הרצאה 1: הקדמה

הקדמה

אלגוריתם הוא שיטה לפתרון **בעיה חישובית**.

במשפט זה מופיעים המושגים **אלגוריתם ובעיה חישובית** אשר כל אחת (אחד) מכם מכירה (מכיר) באופן אינטואיטיבי.

במהלך הסמסטר נלמד סוגים שונים של אלגוריתמים. דרכים להוכחת נכונות של אלגוריתמים המשתייכם לכל אחד מן הסוגים השונים, ודרכים לניתוח הסיבוכיות החישובית של כל אלגוריתם כזה.

המושגים הבסיסיים

בקורס זה נעסוק ב**תורת האלגוריתמים המתמטית**. כל תורה מתמטית מתחילה בהצגת המושגים הבסיסיים בהם היא משתמשת וזוהי מטרת ההרצאה שלנו.

בהרצאה זו נציג, נגדיר ונסביר את המושגים הבאים:

1. בעיה חישובית.
2. אלגוריתם
3. נכונות של אלגוריתם.
4. סיבוכיות של אלגוריתם.

דוגמאות לבעיות חישוביות

אנו מעוניינים באלגוריתמים שיפתרו בעיות חישוביות כגון:

1. הכפלת זוג מספרים.
2. מיון מערך מספרים.
3. כפל מטריצות.

שימו לב:

1. כל אחת מן הבעיות האלה מתארת אינסוף קלטים אפשריים.
2. לכל קלט אפשרי, יש פלט נכון יחיד, או קבוצת פלטים נכונה.

הבהרה חשובה

בקורס זה המושג **קלט** מתייחס למידע שהאלגוריתם מקבל (הארגומנטים) והמושג **פלט** מתייחס למידע שהאלגוריתם מחזיר (ולא לדו שיח עם המשתמש בעזרת מקלדת וטרמינל).

להלן נגדיר את מושג הבעיה החישובית.

הגדרה 1.1: בעיה חישובית

בעיה חישובית - היא שלשה $P=<I,O,f>$ כאשר:

I - **קבוצת הקלט**

O - **קבוצת הפלט**

f - **התאמה (פונקציה) בין קבוצת הקלט לקבוצת הפלט**

הסבר

1. הקבוצה I מכילה את כל הקלטים עמם צריך להתמודד כל אלגוריתם לפתרון P.
2. לכל $i \in I$, $f(i)$ נקרא **הפלט המתאים לקלט** i.

הגדרה 1.2: אלגוריתם

אלגוריתם הוא תיאור של תהליך חישובי אשר ניתן למימוש כתכנית מחשב.

אלגוריתם הפותר בעיה חישובית P מקבל קלט כלשהו $I \in P$, מבצע חישובים תוך שימוש בקלט I, ולאחר ביצוע מספר הוראות סופי מייצר את הפלט $f(I)$.

שימו לב:

כל אלגוריתם לפתרון הבעיה P, אשר מופעל על קלט I, צריך להחזיר את הפלט $f(I)$ (או את אחד מאיברי הקבוצה $f(I)$).

הסבר:

נדגיש את התכונות הבאות:

1. כל ביצוע של אלגוריתם על **קלט חוקי, מסתיים בזמן סופי**.
2. לכל אלגוריתם A, ולכל קלט חוקי I, **כל הביצועים** של A על הקלט I, מסתיימים עם **אותו פלט**.

תיאור אלגוריתמים בעזרת קוד דמה (Pseudo Code)

אנו נתאר אלגוריתמים בעזרת שימוש **בקוד דמה.** השימוש בקוד דמה אינו פורמלי - אנו מביעים כל פעולה בעזרת הוראה בשפה דמוית שפת תכנות ומניחים כי אפשר לבצע מעקב אחר האלגוריתם באופן חד משמעי.

שימו לב

קוד דמה, אינו תוכנה, מטרתו היא ליצור תיאור חד משמעי של האלגוריתם אשר מאפשר יישום האלגוריתם באמצעות תוכנה. במידת הצורך, אפשר לתאר הוראה בשפה טבעית, כל עוד פעולת ההוראה היא **ברורה וחד משמעית.**

<u>נכונות וסיבוכיות של אלגוריתם</u>

איכותו של כל אלגוריתם נשפטת לפי שתי תכונות:

1. **<u>נכונות</u>** - האם האלגוריתם **<u>פותר את הבעיה?</u>**
2. **<u>סיבוכיות</u>** - האם האלגוריתם **<u>פותר את הבעיה ביעילות?</u>**

להלן נטפל בנושא נכונות אלגוריתם באופן פורמלי.

<u>הגדרת נכונות של אלגוריתם</u>

<u>הגדרה 1.3</u>

אלגוריתם A לפתרון **בעיה מתמטית** P הוא **נכון** אם לכל קלט $I \in p$ כל ביצוע של A על I, מניב פלט $f(I)$.

<u>ניסוח אינטואיטיבי יותר</u>

אלגוריתם הוא **נכון** אם לכל **קלט חוקי** האלגוריתם מחשב **פלט נכון**.

<u>שימו לב:</u>

כפי שכבר הדגשנו, אנו דורשים: **לכל קלט** נתון, **כל ביצוע** של האלגוריתם יניב **פלט נכון**.

בהמשך ההרצאה, נסביר ונדגים מהי הוכחת נכונות של אלגוריתם. בנוסף אנו מביאים תזכורת לנושא **סיבוכיות** של **אלגוריתמים** ונדגים מספר ניתוחי סיבוכיות של כמה אלגוריתמים.

<u>דוגמא: חישוב סכום איברי מערך</u>

<u>תיאור קלט:</u> מערך A ובו n מספרים.

<u>תיאור פלט:</u> סכום איברי המערך A.

<u>אלגוריתם 1.1: *Sum*</u>

להלן נציג קוד של אלגוריתם הפותר את הבעיה.

$$
\begin{array}{l}
Sum(A,n) \\
\quad sum \leftarrow 0 \\
\quad \textbf{for } i \leftarrow 1 \textbf{ to } n \\
\qquad sum \leftarrow sum + A[i] \\
\quad \textbf{end(for)} \\
\quad out(sum)
\end{array}
$$

<u>איור 1.1: האלגוריתם *Sum*</u>

<u>שימו לב:</u>

- הבעיה והאלגוריתם הפותר אותה הם פשוטים במיוחד ומטרתם להדגים את עקרונות הטיפול הפורמלי באלגוריתם.
- הארגומנט n מייצג את מספר האיברים במערך A. בדרך כלל, לא נציין את המספר הזה.

<u>הוכחת נכונות של אלגוריתמים</u>

להלן נגדיר את המושגים הדרושים כדי להוכיח את נכונות האלגוריתם באופן פורמלי:

תהי P בעיה חישובית: ויהי Al אלגוריתם לפתרון P.

כדי להוכיח את נכונותו של Al יש להוכיח:

1. **סיום -** לכל קלט $I \in P$, חישוב Al על I **מסתיים** תוך פרק זמן סופי.
2. **נכונות פלט -** לכל קלט $I \in P$, חישוב Al על I מסתיים עם הפלט $f(I)$.

<u>הוכחת סיום</u>

כדי להוכיח סיום של תוכנית איטרטיבית, **מספיק** לבדוק כי **מספר המעברים בכל אחת מן הלולאות הוא חסום**.

<u>הוכחת נכונות פלט</u>

הוכחת **נכונות הפלט** היא לרוב מסובכת ודורשת מיומנות מתמטית. לעתים, הנכונות **ברורה אינטואיטיבית** אולם ההוכחה המתמטית הפורמלית, היא קשה וטכנית.

<u>בניית משפט נכונות – תבנית הטענה</u>

השלב הראשון בהוכחת הנכונות הוא בניית **<u>משפט נכונות</u>**.

כל משפט נכונות נבנית לפי התבנית הבאה:

<u>תבנית משפט נכונות</u>

לכל קלט $I \in P$, חישוב Al על I מסתיים עם הפלט $f(I)$.

כדי להפוך את התבנית למשפט נכונות מלא, יש **להחליף** את הביטוי I בתאור מדוייק של הקלט ואת הביטוי $f(I)$ בתאור מדוייק של הפלט.

3

<u>דוגמא - בניית משפט נכונות</u>

הבעייה החישובית שבדוגמא היא בעית הסכום.

<u>קבוצת הקלטים</u> - קבוצת המערכים שאיבריהם הם מספרים. (כמובן שזוהי קבוצה אינסופית).

<u>משפט הנכונות</u>

לכל **<u>מערך מספרים</u>** A (תיאור הקלט), האלגוריתם Sum מחשב את **<u>סכום</u>** **<u>איברי המערך</u>** A (תיאור הפלט).

<u>שימו לב</u>

משפט הנכונות תלוי אך ורק בבעיה החישובית ואין בו שום התיחסות לאלגוריתם (למעט השם).

<u>הוכחת משפט הנכונות</u>

בטרם ניגש להוכחת נכונות האלגוריתם עלינו להבין באופן אינטואיטיבי את דרך פעולתו ולנסות לזהות שגיאות אפשריות. הדרך להשיג משימה זו היא **<u>לבצע הרצות נסיון.</u>**

<u>הרצות נסיון</u>

1. בוחרים מספר קלטים לדוגמה. כל קלט כזה, צריך לייצג בעיה פוטנציאלית באלגוריתם. לדוגמה:
   1.1 מערך עם איבר יחיד
   1.2 מערך עם איברים שליליים.

<u>הרצות נסיון (המשך)</u>

2. מריצים את האלגוריתם על כל אחד מן הקלטים לדוגמה ומוודאים באופן בלתי תלוי שהאלגוריתם מניב פלט נכון.
3. במקרה שהאלגוריתם המוצע אינו נכון, יש סיכוי טוב שהרצת הנסיון תניב תוצאה שגויה ובכך תחסוך את המאמץ בהוכחת אלגוריתם שגוי.
4. זיהוי השגיאה בדרך כלל מסייע בתיקון הפגם באלגוריתם.

<u>הבנה אינטואיטיבית של האלגוריתם</u>

נניח כי ביצענו מספר הרצות ניסיון ולא זיהינו שגיאה באלגוריתם.

במצב זה, עלינו לנסות ולהבין באופן אינטואיטיבי את דרך פעולת האלגוריתם.

הכוונה היא לנסות ולהבין את תפקידם של המשתנים המופיעים בקוד ואת הדרך בה מחושב הפלט.

בדוגמה (הטריויאלית) שלנו, אנו רואים כי הפלט הוא המשתנה sum.

במקרה זה, ה"אינטואיציה" אומרת לנו כי לאחר המעבר בלולאת האלגוריתם בפעם ה-i, **ערכו של המשתנה** sum **שווה לסכום** i **האיברים הראשונים במערך הקלט.**

<u>הוכחה פורמלית של נכונות האלגוריתם</u>

כעת אפשר לומר כי אנו מבינים את עקרונות הפעולה של האלגוריתם.
בשלב זה, עלינו להביע את האינטואיציה הזו באופן פורמלי.
נתבונן שוב בקוד:

$Sum(A,n)$
  $sum \leftarrow 0$
  for $i \leftarrow 1$ **to** n
    $sum \leftarrow sum + A[i]$
  end(for)
  $out(sum)$

אפשר לראות כי בסיום המעבר ה - i, ($1 \le i \le n$), בלולאת האלגוריתם, מיד לאחר ביצוע הוראת ההשמה מתקיים:

$$\boxed{sum = \sum_{j=1}^{i} A[j]}$$

בסיום המעבר האחרון בלולאת האלגוריתם מתקיים:

$$\boxed{sum = \sum_{j=1}^{n} A[j]}$$

כלומר: **<u>האלגוריתם מחשב את התוצאה המבוקשת.</u>**

<u>הוכחת משפט הנכונות</u>

משפט הנכונות היא **טענה מתמטית** .
כמו כל טענה מתמטית אחרת, הטענה דורשת **הוכחה מתמטית**.
כזכור, עלינו להראות כי עבור כל **קלט חוקי** האלגוריתם מייצר **פלט נכון.**
כדי לבצא את המשימה, נשתמש בתיאורי הקלט והפלט ובקוד האלגוריתם.
כל הוכחת נכונות משתמשת במעקב אחרי פעולות האלגוריתם. באלגוריתם שאינו מכיל לולאה, אנו יכולים (לפחות באופן תיאורטי) להשתמש במעקב ולהגיע לתוצאה הרצויה. באלגוריתם המכיל לולאות אנו נתקלים בבעיה.

<u>הוכחת משפט הנכונות (המשך)</u>

כדי להוכיח את משפט הנכונות עבור אלגוריתם המכיל לולאות (לפחות אחת), עלינו להשתמש באינדוקציה.
אנו נוכיח את משפט הנכונות **<u>באינדוקציה על מספר המעברים בלולאת האלגוריתם.</u>**

איך מוכיחים את הטענה?

ניזכר בדיון שלנו אודות האלגוריתם:

```
Sum(A,n)
  sum ← 0
  for i ← 1 to n
    sum ← sum + A[i]
  end(for)
  out(sum)
```

כבר ראינו כי:

בסיום המעבר ה - i $(1 \le i \le n)$

בלולאת האלגוריתם, מייד לאחר

ביצוע הוראת ההשמה מתקיים:

$$\boxed{sum = \sum_{j=1}^{i} A[j]}$$

שיטת השמורה בשלושה שלבים

נראה כעת, איך מוכיחים את הטענה באופן פורמלי באינדוקציה.

1. **ניסוח השמורה** מנסחים טענה אודות הקשר בין מספר המעברים בלולאה לבין ערך המשתנים בנקודה מסוימת. לטענה קוראים **שמורת הלולאה**, כי היא נשמרת בכל מעבר בלולאה. לנקודה שבה השמורה מתקיימת קוראים **נקודת השמורה.**

2. **בסיס האינדוקציה:** מוכיחים את נכונות השמורה, במעבר הראשון בלולאה, **בעזרת מעקב**.

3. **שלב האינדוקציה:** מניחים כי השמורה נכונה במעבר ה - i, ומוכיחים, **שוב בעזרת מעקב**, את נכונותה במעבר ה $i+1$.

להלן נדגים את השימוש שלושת השלבים הללו בהוכחת משפט הנכונות לאלגוריתם Sum.

בחירת שמורת הלולאה

כדי שהשמורה תאפשר להוכיח את נכונות האלגוריתם, עליה להביע את הקשר בין:

1. הקלט לאלגוריתם.
2. משתני האלגוריתם.
3. הפלט המחושב.

בדרך כלל, השמורה תהיה תלויה בדרך כלשהי במשתנים שערכיהם משתנים במהלך המעבר בלולאה.

באלגוריתמים פשוטים, בחירת השמורה היא טבעית ופשוטה. ככל שרמת הקושי של האלגוריתם הנדון עולה, בחירת השמורה קשה יותר ולעתים מהווה אתגר של ממש.

הוכחת נכונות האלגוריתם Sum

1. בחירת השמורה ונקודת השמורה

```
Sum(A,n)
  sum ← 0
  for i ← 1 to n
    sum ← sum + A[i]
  end(for)
  out(sum)
```

בדוגמה שהבאנו אפשר לבחור כשמורה את: במעבר ה - i בלולאת האלגוריתם, $(1 \le i \le n)$, מייד לאחר ביצוע הוראת ההשמה מתקיים:

$$\boxed{sum = \sum_{j=1}^{i} A[j]}$$

2. הוכחת השמורה

נראה כעת, איך מוכיחים את הטענה הזו **באינדוקציה על מספר המעברים בלולאה.**

בסיס האינדוקציה $(i = 1)$

עלינו להוכיח כי בתום המעבר הראשון בלולאה מתקיים:

$$\boxed{sum = \sum_{j=1}^{1} A[j]}$$

הוכחה

כל הוכחת אלגוריתם משתמשת במעקב אחרי פעולות האלגוריתם.

להלן ההוכחה

לפני הכניסה ללולאה, מתבצעת ההשמה $sum \leftarrow 0$. במהלך המעבר הראשון, מחברים למשתנה sum את $A[1]$. בתום המעבר מתקיים:

$$\boxed{sum = \sum_{j=1}^{1} A[j]}$$

מש"ל.

צעד האינדוקציה

1. הנחת האינדוקציה

נניח כי לאחר $i-1$ מעברים בלולאה מתקיים

$$\boxed{sum = \sum_{j=1}^{i-1} A[j]}$$

2. <u>הוכחת הטענה</u>

לפי בנחת האינדוקציה ידוע כי לאחר [illegible] מעברים בלולאה מתקיים $\boxed{sum=\sum_{j=1}^{i-1}A[j]}$

במהלך המעבר ה i בלולאה, מחברים למשתנה sum את $A[i]$. מכאן: בתום המעבר ה- i בלולאה מתקיים:

$$\boxed{sum=\sum_{j=1}^{i}A[j]}$$

מש״ל

<u>תזכורת:</u> הדוגמה היא **<u>פשוטה במיוחד</u>** ומשמשת להדגמת הטכניקה של ההוכחה באינדוקציה.

<u>סיום ההוכחה</u>

נציב בשמורה את הערך $i=n$ ונקבל:
לאחר n מעברים בלולאה מתקיים:

$$\boxed{sum=\sum_{j=1}^{n}A[j]}$$

או במלים אחרות: נכונות האלגוריתם נובעת מטענת האינדוקציה עבור $i=n$.

מש״ל

<u>שמורות חליפיות</u>

בדרך כלל, אפשר לבחור כמה שמורות חליפיות. לכל שמורה חליפית תתאים נקודת שמורה משלה:

לדוגמא: אפשר גם לבחור כשמורה את הטענה בתחילת המעבר ה - i בלולאת האלגוריתם, ($1\le i\le n$), **לפני ביצוע הוראת ההשמה** מתקיים:

$$\boxed{sum=\sum_{j=1}^{i-1}A[j]}$$

<u>תרגיל:</u> הוכיחו את נכונות האלגוריתם Sum בעזרת השמורה החליפית.

כדאי לבחור את השמורה בהוכחה, לפי פשטות ההוכחה המתקבלת

<u>שמורות גרועות</u>

יש אינסוף ביטויים אשר הם נכונים בכל פעם שבקרת התכנה מגיעה לנקודת השמורה, אך הם חסרי תועלת להוכחת הנכונות.

דוגמה לשמורה כזו היא $1+2=3$.

זה ביטוי נכון, אך הוא אינו קושר בין הקלט לבין משתני התכנית או בין משתני התכנית לבין הקלט.

עוד שמורה גרועה היא

$$1\le i\le n$$

בחירת השמורה הנכונה היא האתגר שבהוכחת הנכונות.

<u>סיבוכיות האלגוריתם</u>

1. הלולאה מתבצעת בדיוק n פעמים.
2. בכל מעבר מתבצע מספר פעולות קבוע (Constant), כלומר מספר פעולות שאינו תלוי בגודל הקלט.

משתי נקודות אלה נובע כי הסיבוכיות היא לינארית, כלומר $\Theta(n)$.

<u>האלגוריתם *Max*: חישוב איבר מרבי במערך חד ממדי</u>

$$\boxed{\begin{array}{l} Max(A)\\ \quad M\leftarrow A[1]\\ \quad \textbf{for } i=2 \textbf{ to } n\\ \quad\quad M\leftarrow \max(M,A[i])\\ \quad \textbf{End for}\\ \textbf{out}(M)\end{array}}$$

<u>איור 1.2: האלגוריתם *Max*</u>

<u>תרגיל:</u>

הוכיחו את נכונות האלגוריתם.

5

<u>אלגוריתם לחישוב איבר מכסימלי</u>

נתון מערך דו-מימדי A מסדר $n\times n$ שאיבריו הם מספרים.

מחפשים אלגוריתם לחישוב האיבר המכסימלי במערך A.

<u>שימו לב:</u> שוב אנו משתמשים בבעיה פשוטה כדי להדגים עקרון בסיסי.

$$\boxed{\begin{array}{l} Max2d(A)\\ \quad \textbf{for } i=1 \textbf{ to } n\\ \quad\quad B[i]\leftarrow Max(A[i])\\ \quad \textbf{End for}\\ \textbf{out}(Max(B))\end{array}}$$

<u>איור 1.3 האלגוריתם *Max2d*</u>

<u>האלגוריתם *Max2d* - תיאור</u>

נסמן את השורה ה-i של המערך A ב-$A[i]$. האלגוריתם מבצע n איטרציות (מעברים בלולאה): באיטרציה ה-i האלגוריתם מחשב את האיבר המרבי בשורה $A[i]$ ומכניס איבר זה אל $B[i]$ לאחר מכן האלגוריתם מפעיל את השיטה $Max(B)$ כדי לחשב את האיבר המרבי במערך B.

הוכחת נכונות

שימו לב, בהוכחת הנכונות של השיטה $i = 1$, עלינו להשתמש בנכונות השיטה $Max(A)$ ובשיטת השמורה.

תרגיל

הציעו שמורה מתאימה ללולאה בקוד השיטה $i = 1$.

פיתוח אלגוריתם כתהליך

פיתוח אלגוריתם הוא תהליך **מחזורי** (איטרטיבי):

- מתחילים בהצעה ראשונית.
- בכל מחזור (איטרציה) **בודקים את ההצעה הנוכחית** בעזרת הרצת דוגמאות.
- אם מזהים שגיאה באלגוריתם.
- מתקנים את השגיאה.

לאחר שלא מוצאים יותר שגיאות בדרך הזו, ניגשים להוכחת האלגוריתם.

ניתוח זמן ריצה (חזרה)

זמן ריצה הוא הגורם החשוב ביותר שיש לבחון כאשר בוחרים אלגוריתם לפתרון בעייה חישובית כלשהי.
לאחר שהשתכנענו שאלגוריתם הוא נכון, יש לנתח את **סיבוכיות הזמן** שלו.
סיבוכיות הזמן של אלגוריתם, היא פונקציה מתמטית המאפשרת לנו לחזות את זמן הריצה של תוכנית המבוססת על האלגוריתם ולהשוות בין אלגוריתמים שונים.
בנספח נביא חזרה קצרה בנושא ניתוח סיבוכיות הזמן של אלגוריתמים.

סיכום

בהצאה זו הגדרנו והדגמנו את המושגים הבסיסיים שישמשו אותנו במהלך הקורס והם:

1. בעיה חישובית.
2. אלגוריתם.
3. נכונות אלגוריתם.
4. סיבוכיות אלגוריתם.

נספח: הגדרת סיבוכיות הזמן של אלגוריתמים

בתחילת הדיון למדוד מהם הגורמים המשפיעים על זמן הביצוע של תכנית לפתרון בעיה כלשהי. הגורמים החשובים הם:

1. יעילות הקידוד.
(תרגום מאלגוריתם לתכנית)
2. יעילות המהדר.
(תרגום משפת על לשפת מכונה)
3. יעילות המעבד.
4. יעילות האלגוריתם.

האם אתם מכירים גורם נוסף?

מה מידת ההשפעה של כל גורם?

1. יעילות הקידוד - כל שורה משורות האלגוריתם תקודד למספר **קבוע** של הוראות בשפה עילית.
יעילות המהדר - כל הוראה בשפה עילית תתורגם למספר **קבוע** של הוראות בשפת מכונה. **ללא תלות באורך הקלט**
מהדר יעיל יותר יכול להקטין מספר זה **בגורם קבוע** כלשהו.
3. הגדלת מהירות המעבד - (לדוגמא מ 1 MHz ל 1 GHz, תקטין את זמן הריצה בגורם קבוע (בדוגמא 1000).

שיפורים בגורם קבוע

כל שיפור באיכות **הקידוד המהדר** או **המעבד**, משפיע על זמן הריצה **אך ורק** על ידי הכפלה בקבוע, שאינו תלוי באורך הקלט.
אין לזלזל בשיפור שיקטין את זמן הריצה למחצית, לשליש, לעשירית, או אף לאלפית (כאשר משפרים מחשב) מערכו הקודם. כאשר מתמודדים עם בעייה קשה, יש למצות את כל אפשרויות השיפור של הפתרון.
לעתים כל האפשרויות גם יחד אינן מספיקות בהתמודדות עם בעיות בגודל (אורך קלט) הולך וגדל.
האם יש אפשרויות שיפור נוספות?

שיפור האלגוריתם

מספר הצעדים שאלגוריתם מסויים מבצע הוא **פונקציה של אורך הקלט לאלגוריתם.** במהלך הקורס אנו נוכח כי אפשר לשפר אלגוריתמים מסויימים בסדר גודל, למשל מסיבוכיות n^2 לסיבוכיות [illegible]. ברור אם כן כי בעזרת שיפור האלגוריתם אפשר לשפר את זמן הריצה בגורם **העולה על קבוע. בשום דרך אחרת אי אפשר להשיג שיפור כזה.**

סיבוכיות זמן ריצה

נתון אלגוריתם A הפותר בעיה P.

נתבונן בסדרת קלטים מ-P

$I_1, I_2, \ldots, I_n, \ldots$

שאורכם הולך וגדל.

בדרך הטבע, ככל שאורך הקלט גדול יותר, האלגוריתם יעבוד זמן רב יותר.

הגדרת סיבוכיות הזמן

סיבוכיות הזמן של האלגוריתם או בפשטות **זמן הריצה** של האלגוריתם תוגדר כפונקציה המתאימה את מספר צעדי האלגוריתם לאורך הקלט.

$t(n) =$ ***מספר צעדים מירבי בריצה של האלגוריתם על קלט באורך*** n

סיבוכיות המקרה הגרוע ביותר

יש אלגוריתמים שזמן הריצה שלהם על קלטים באורך שווה משתנה מקלט לקלט.

דוגמה: במיון מהיר (Quick-Sort), יש קלטים רבים באורך n, שזמן הריצה שלהם הוא $\Theta(n \log n)$ אך עבור קלט **ממוין**, זמן הריצה הוא $\Theta(n^2)$.

במקרה זה, עלינו לבחור מהו זמן הריצה המייצג את הקלטים באורך n.

בדרך כלל, מקובל לבחור עבור כל n את הקלט עבורו האלגוריתם רץ במהירות הנמוכה ביותר (Worst Case).

נימוקים לשימוש בWorst Case

הנימוק המרכזי הוא **מניעת הפתעות.**

כשמשתמשים ב Worst Case, סיבוכיות האלגוריתם היא **חסם עליון** על זמן הריצה עבור כל המקרים.

נימוק חשוב נוסף הוא שיחסית קל לחשב את המקרה הגרוע ביותר.

במקרים מסוימים (כגון במיון מהיר) נתיחס גם **לזמן הריצה הממוצע** (יוגדר מאוחר יותר).

שימו לב: לעיתים משתמשים במושג Best Case לצרכי הסבר.

אין משמעות לסיבוכיות המקרה הטוב ביותר ואין להשתמש במושג זה לצרכי ניתוח זמן ריצה.

הגדרה פורמלית של סיבוכיות זמן

יהי Al אלגוריתם כלשהו. לכל קלט חוקי I נסמן ב $t_{Al}(I)$ את זמן הריצה של Al על הקלט I.

סיבוכיות הזמן של Al מוגדרת על ידי:

$$t_{Al}(n) = \max_{|I|=n}\{t_{Al}(I)\}$$

לכל $n > 0$. כאשר המכסימום נלקח על קבוצת כל הקלטים באורך n.

ניתוח זמן ריצה

מטרתו של ניתוח זמן ריצה של אלגוריתם היא:

1. לסייע לנו **לצפות** את זמן הריצה של תכנית המבוססת על האלגוריתם ללא הרצה של התוכנית.
2. להשוות יעילות של כמה אלגוריתמים.
3. להחליט אם לחפש אלגוריתם נוסף.

שימו לב: מפני שאלגוריתם הוא ברמת הפשטה גבוהה, לעתים קשה לעמוד על משמעות שינוי בגורם קבוע.

מסקנה

בדיון בסיבוכיות זמן של אלגוריתמים נבדוק רק שיפורים שיקטינו את **סדר הגודל** של זמן הריצה **ונתעלם מגורמים קבועים.**

דוגמא

נניח כי אלגוריתם A פותר בעיה מסוימת בזמן $n^2 + 7$ ואלגוריתם B פותר אותה הבעיה בסיבוכיות $n^2 + 12$ אנו נאמר כי לשני האלגוריתמים סיבוכיות זמן $\Theta(n^2)$.

השוואת קצבי גידול

הסימונים O, Θ, וΩ מאפשרים לנו להשוות בין קצבי הגידול של פונקציות שונות כאשר n שואף לאינסוף.

דוגמא

נניח כי זמן הריצה של תוכנית כלשהי מבוטא על ידי הפונקציה $T(n)$ המוגדרת על ידי:

$$T(n) = 31n^3 + 17.5n^2 - 3.2n + 1077$$

קל לראות (איך?) כי כאשר $n \to \infty$ ערך $T(n)$ נשלט על ידי n^3. במקרה כזה נאמר כי $T(n) = \Theta(n^3)$.

<u>הגדרות</u>

נאמר כי $f(n)=O(g(n))$ (הפונקציה $g(n)$ **חוסמת מלמעלה באופן אסימפטוטי** את הפונקציה $f(n)$) אם קיימים קבועים n_0 ו c_0 המקיימים:

לכל $n>n_0$, $f(n)\le c_0 g(n)$.

נאמר כי $f(n)=\Omega(g(n))$ (הפונקציה $g(n)$ **חוסמת מלמטה באופן אסימפטוטי** את הפונקציה $f(n)$) אם קיימים קבועים n_1 ו c_1 המקיימים:

לכל $n>n_1$, $f(n)\ge c_1 g(n)$

שימו לב: אם $f(n)=O(g(n))$ אז $g(n)=\Omega(f(n))$.

אלגוריתמים א׳ הרצאה 1 - פרופ׳ עמוס ישראלי

<u>המשך ההגדרות</u>

הפונקציה $f(n)$ היא $\Theta(g(n))$ (הפונקציה $f(n)$ **שוה אסימפטוטית** לפונקציה $g(n)$) אם

1. $f(n)=O(g(n))$

וגם

2. $g(n)=O(f(n))$

שימו לב: אם $f(n)=\Theta(g(n))$ אז גם $g(n)=\Theta(f(n))$

© כל הזכויות שמורות לפרופסור עמוס ישראלי

אלגוריתמים א׳ הרצאה 1 - פרופ׳ עמוס ישראלי

<u>דוגמא</u>

תהי

$$T(n)=31n^3+17.5n^2-3.2n+1077$$

נראה כי $T(n)$ היא $O(n^3)$.

$$T(n)\le 31n^3+17.5n^2+1077$$

עבור $n>1077$ נקבל כי

$$T(n)\le 31n^3+n^3+n^3=33n^3$$

מש״ל.

© כל הזכויות שמורות לפרופסור עמוס ישראלי

אלגוריתמים א׳ הרצאה 1 - פרופ׳ עמוס ישראלי

<u>סקלה למדידת קצב הגידול של פונקציות</u>

אנו מחפשים דרך לתאר התנהגות של פונקציות. הדרך שנבחרה היא להשוות כל פונקציה לקבוצת פונקציות בסיסיות המוכרות לכל.

הפונקציות הבסיסיות הן:

1. פונקציות מעריכיות: $f(n)=a^{n^k}$, $k=1,2,\dots$.
2. מונומים $f(n)=n^k$, $k=1,2,\dots$
3. פונקציות לוגריתמיות $f(n)=(\log n)^k$ $k=1,2,\dots$.

שימו לב: כל הפונקציות הללו הן **חיוביות**.

© כל הזכויות שמורות לפרופסור עמוס ישראלי

אלגוריתמים א׳ הרצאה 1 - פרופ׳ עמוס ישראלי

<u>הסדר האסימפטוטי</u>

להלן הסדר האסימפטוטי של כמה פונקציות מוכרות:

$$1<\log n<(\log n)^k<\log n^{k+1}$$
$$<n^{\varepsilon}(\forall\,\varepsilon>0)<n^{1/2}<$$
$$n<n\log n<n^k<n^{k+1}<$$
$$<2^n<n!$$

© כל הזכויות שמורות לפרופסור עמוס ישראלי

אלגוריתמים א׳ הרצאה 1 - פרופ׳ עמוס ישראלי

<u>השוואת סדרי גודל</u>

בהינתן פונקציה כלשהי נרצה להשוות אותה לאחת הפונקציות התקניות. ברובם המכריע של המקרים איננו צריכים לחשב גבולות כלל אלא להשתמש בהגיון ובכמה חוקים בסיסיים.

כדוגמא נביא שאלה מתוך בוחן:

קבע את יחסי סדר הגודל בין

$$f(n)=n(\log n)^2$$

לבין

$$g(n)=\frac{n^2}{\log n}$$

© כל הזכויות שמורות לפרופסור עמוס ישראלי

אלגוריתמים א׳ הרצאה 1 - פרופ׳ עמוס ישראלי

בדוגמא זו, קל מאוד לראות, למשל על ידי הכפלת שתי הפונקציות בפונקציה $\log n$ כי $g(n)$ גדלה מהר יותר מ $f(n)$ כלומר $f(n)=O(g(n))$.

שימו לב:

לעתים אנו אומרים כי סיבוכיות האלגוריתם היא $O(f(n))$ כאשר בעצם אנו מתכוונים לומר $\Theta(f(n))$.

© כל הזכויות שמורות לפרופסור עמוס ישראלי

אלגוריתמים א׳ הרצאה 1 - פרופ׳ עמוס ישראלי

<u>דוגמא נוספת</u>

נתון כי הפונקציה $f(n)$ מתנהגת באופן שונה עבור ערכי n שונים כמתואר בטבלה הבאה:

n	0-25	26-250	>250
$f(n)$	n^4	n^3	n^2

מהי סיבוכיות הפונקציה f?

<u>תשובה:</u> מן ההגדרות נובע מייד כי מה שקובע את סיבוכיות הפונקציה היא התנהגותה כאשר ערכי n שואפים לאינסוף. להתנהגות f עבור מספר סופי של ערכי n אין שום חשיבות ולכן

$$f(n)=\Theta(n^2)$$

© כל הזכויות שמורות לפרופסור עמוס ישראלי

<u>הרצאה 2: אלגוריתמי מיון פשוטים</u>

בעיית המיון היא אחת הבעיות החשובות והנחקרות ביותר בכל מדעי המחשב. בהרצאה זו נעסוק באלגוריתמי מיון פשוטים שסיבוכיותם $\Theta(n^2)$. במהלך ההרצאה נציג שלושה אלגוריתמי מיון קלאסיים :

1. מיון הכנסה (Insertion Sort).
2. מיון בחירה (Selection Sort).
3. מיון בועות (Bubble Sort) .

לאחר דיון קצר בהוכחת נכונות של אלגוריתמים למיון, נוכיח את נכונות מיון הכנסה וננתח את הסיבוכיות שלו.

בחלק השני של ההרצאה נציג שתי גרסאות של מיון בועות (Bubble Sort) , נוכיח את נכונותן וננתח את סיבוכיות הזמן שלהם במקרה הגרוע ביותר.

<u>הגדרה אינטואיטיבית</u>

אלגוריתם מיון מקבל כקלט מערך מספרים, ומחזיר מערך המכיל את אותם המספרים, מסודרים בסדר עולה או יורד. להלן נגדיר את בעית המיון באופן מתמטי :

<u>הגדרה</u>

תהי U קבוצה כלשהי. איברי הקבוצה U הם **<u>ניתנים להשוואה</u>**, אם לכל שני איברים $a,b \in U$ מתקיימת אחת מן האפשרויות הבאות :

1. $a < b$
2. $a = b$.
3. $a > b$.

<u>הגדרת תמורה</u>

מערך B הוא תמורה של מערך A אם המערכים A ו-B מכילים <u>בדיוק אותם איברים</u> וכל איבר מופיע בדיוק אותו מספר פעמים ב-A וב-B.

<u>לדוגמא</u>

1,2,3 היא תמורה של 3,1,2.
4,1,9,7 היא תמורה של 1,9,4,7
5,1,1 היא תמורה של 1,5,1
3,2,7 <u>אינה תמורה</u> של 3,5,9
4,7,7 <u>אינה תמורה</u> של 4,4,7
5,1,2,2 <u>אינה תמורה</u> של 1,2,5,5

<u>הגדרת פורמלית לבעיית המיון</u>

<u>קלט</u>

מערך A בן n איברים (כגון מספרים) ניתנים להשוואה המסודרים בסדר שרירותי.

<u>פלט</u>

מערך B ובו איברי A מסודרים לפי גודלם.

<u>במלים אחרות</u>: הפלט הוא תמורה מסודרת (ממוינת) של איברי הקלט.

<u>הערה</u>

למען הפשטות נניח כי כל איברי המערך A <u>נבדלים</u>, כלומר : שונים זה מזה (באנגלית distinct).

<u>מיון בחירה (Selection Sort)</u>

קוד האלגוריתם מופיע באיור 2.1 כותרת האלגוריתם היא $SelectionSort(A,n)$ כאשר נתון ש-A הוא מערך שאיבריו ניתנים להשוואה ומספרם הוא n. האלגוריתם משתמש במערך עזר B. לולאת האלגוריתם מבוצעת $n-1$ פעמים. במעבר ה-i בלולאה, האלגוריתם מעביר את האיבר המינימלי בתת המערך $A[i...n]$ אל $B[i]$.

<u>מיון בחירה - הקוד</u>

```
SelectionSort(A)
for i = 1 to n-1
    ind <- index of minimal
           element of A[i...n]
    Swap(A[i], A[ind])
  end for
Out(B)
```

<u>איור 2.1: מיון בחירה</u>

<u>שימו לב</u>

האלגוריתם משתמש בשגרה $Swap(A[i],A[j])$ אשר מחליפה את יקומם של האיברים $A[i]$ ו-$A[j]$.

תזכורת: הוכחת נכונות של אלגוריתם

כדי להוכיח נכונות של אלגוריתם עלינו להוכיח:

1. האלגוריתם מסתיים בכל ריצה עם קלט חוקי.
2. בכל ריצה עם קלט חוקי, הפלט נכון.

כדי לענות על שתי הדרישות הללו, עלינו להתיחס במדויק אל פעולת האלגוריתם כולל:

1. מספר המעברים בכל לולאה.
2. ערכי המשתנים בכל שלב של החישוב.
3. הצלחה או כישלון בביצוע טסטים כגון if ו while.

מיון בחירה - הוכחת סיום

כמו במקרים רבים, הוכחת הסיום של האלגוריתם היא מיידית ונובעת מכך שהאלגוריתם מכיל לולאה מקוננת אחת ושמספר המעברים בכל לולאה חסום על ידי גודל הקלט n.

הערה

אפשר להוכיח כי סיבוכיות האלגוריתם היא $\Theta(n^2)$, כלומר, בכל ריצה עם קלט שארכו n מספר הצעדים חסום על ידי cn^2, עבור איזשהו קבוע c. מכאן נובע מיידית כי האלגוריתם מסתיים.

איך מוכיחים נכונות אלגוריתמי מיון?

כדי להוכיח נכונות פלט של אלגוריתם מיון עלינו:

1. להוכיח כי מערך הפלט הוא תמורה של מערך הקלט.
2. להוכיח כי מערך הפלט ממוין.

נכונות מיון בחירה

משפט הנכונות

אלגוריתם *InsertionSort* מקבל כקלט מערך של איברים בני השוואה ומחזיר תמורה ממוינת של מערך הקלט.

שימו לב:

טענת הנכונות לכל אלגוריתם מיון היא זהה, למעט שם האלגוריתם.

הוכחה

נתבונן שוב בקוד

```
SelectionSort(A)
for i = 1 to n−1
    ind ←index of minimal
          element of A[i...n]
    Swap(A[i], A[ind])
end for
Out(B)
```

בטרם נציע שמורה נגדיר סימונים:

m_1 - האיבר המינימלי במערך A.

m_2 - האיבר השני בקטנו במערך A.

...

m_i - האיבר ה-i בקטנו במערך A.

השמורה

בתום המעבר ה-i, $0 \le i \le n$, מתקיים

$$B[1...i] = [m_1, m_2, ...m_i]$$

הוכחת משפט הנכונות

נכונות השמורה נותרת כתרגיל לקורא.

לאחר n מעברים בלולאה מתקיים:

$$B[1...n] = [m_1, m_2, ...m_n]$$

ומכאן נובע כי המערך B הוא תמורה ממוינת של המערך A.

מש"ל

סיבוכיות מיון הכנסה

בכל ביצוע מבצעים $n-1$ איטרציות.

סיבוכיות האיטרציה ה-i, $0 \le i \le n$, היא $\Theta(i)$.

מכאן נובע כי סיבוכיות האלגוריתם שווה לטור החשבוני

$$T(n) = \sum_{i=1}^{n} i = \Theta(n^2)$$

מיון בועות (Bubble Sort)

מיון בועות הוא אלגוריתם קלאסי ששנים ארוכות נחשב כאלגוריתם המיון היעיל ביותר הקיים.

יתרונו הגדול של האלגוריתם הוא קלות הקידוד וחסרונו הגדול, כפי שנוכיח, הוא סיבוכיות זמן גבוהה מדי.

תחילה נציג אלגוריתם בסיסי, נוכיח את נכונותו וננתח את הסיבוכיות שלו.

נמשיך בשיפור האלגוריתם וננתח את הסיבוכיות הממוצעת של האלגוריתם המשופר. האלגוריתם נקרא **מיון בועות** שכן בכל איטרציה האיבר המכסימלי עולה לראש המערך כבועה הצפה על פני המים.

תיאור כללי

האלגוריתם נערך ב $n-1$ איטרציות.
באיטרציה ה-i האלגוריתם מעביר את האיבר ה-i בגודלו אל $A[n-i+1]$.
כדי לבצע זאת, האלגוריתם בוחן את זוגות האיברים הסמוכים הבאים:

$(1,2),(2,3),...,(n-i,n-i+1)$

בכל פעם שזוג איברים כזה מסודר **בסדר יורד**, האלגוריתם מפעיל את השגרה $Swap(A[i],A[j])$ אשר מחליפה את יקומם של האיברים $A[i]$ ו-$A[j]$.

קוד האלגוריתם הבסיסי

```
BaseBubble(A)
for Lim = n downto 2 do
    for j = 1 to Lim - 1 do
        if A[j] > A[j+1]
            Swap(A[[j], A[j+1])
        end if
    end for
end for
```

איור 2.2: מיון בועות

שימו לב

המיון נערך **במקום** (באנגלית in Place), כלומר בסיום ביצוע האלגוריתם אברי **מערך הקלט** ממוינים.

משפט נכונות האלגוריתם

אלגוריתם $BaseBubble$ המקבל כקלט מערך A שאיבריו ניתנים להשוואה וממיין את איברי המערך.

שימו לב

כזכור עלינו להוכיח:

1. האלגוריתם מסתיים.
2. הפלט הוא תמורה של הקלט.
3. הפלט ממוין.

הוכחת שני הסעיפים הראשונים

סעיף 1

האלגוריתם מכיל שתי לולאות מקוננות, מספר המעברים בכל לולאה חסום על ידי n ולכן ברור כי האלגוריתם מסתיים.

סעיף 2

בכל מהלך ביצוע האלגוריתם, איברי המערך מוזזים אך ורק על ידי החלפה בין שניים מהם ועל כן ברור כי מערך הפלט הוא תמורה של מערך הקלט.
נותר לנו להוכיח אך ורק כי מערך הפלט ממוין.

אינטואיציה לנכונות הפלט

אנו נוכיח כי בתחילת כל איטרציה (האלגוריתם מתנהל ב-$n-1$ איטרציות) המערך A מחולק ל:

1. חלק עליון ממוין.
2. חלק תחתון שמצבו (מבחינת המיון) אינו ידוע.

כמו כן מתקיים: כל איבר בחלק העליון גדול ממש מכל איבר בחלק התחתון. המשתנה LIM מצביע על האיבר העליון בחלק התחתון. בנוסף: בכל איטרציה, עובר איבר אחד מן החלק התחתון אל החלק העליון וערך המשתנה LIM קטן ב 1.

הוכחת הנכונות

סימונים

לכל $1 \le i \le n$ נסמן ב M_i את האיבר ה-i בגודלו בA, כלומר M_1 הוא האיבר המכסימלי ב A, M_2 האיבר השני בגודלו ב-A וכן הלאה.

טענה – הגדרת השמורה

לכל i, $1 \le i \le n$, לאחר האיטרציה ה-i מתקיים:

$A[n]=M_1,...,A[n-i+1]=M_i$.

שיוויונים אלה נשמרים עד סוף ביצוע האלגוריתם.
להלן יקראו אברים אלה (כולל M_i) בשם: החלק הממוין של המערך A.
החלק השני של המערך A:

$A[1],A[2],...,A[n-i]$

יקרא: אזור אי הודאות

הוכחת הנכונות

ההוכחה באינדוקציה על המעברים בלולאה.

בסיס ($i=1$)

עלינו להוכיח כי בתום האיטרציה הראשונה, $A[n]=M_1$ והאיבר $A[n]$ לא ישתנה יותר.
יהי j האינדקס של M_1 במערך הקלט.
במהלך האיטרציה הראשונה, M_1 לא זז ממקומו עד אשר האיבר $A[j]$ מושווה עם $A[j+1]$ ואז הם מוחלפים.
מרגע זה ועד תום האיטרציה, האיבר M_1 מושווה עם כל איבר הנמצא מעליו. מאחר שהוא האיבר המכסימלי

ב-A, M_1 מחליף את מקומו בכל השוואה ובסוף האיטרציה מתקיים $A[n] = M_1$.
מן הקוד נובע כי באיטרציות הבאות, האיבר $A[n]$ לא יושווה יותר עם אף איבר נוסף ומקומו לא ישתנה עד תום ביצוע האלגוריתם.

מש"ל

<u>צעד האינדוקציה</u>

<u>הנחה</u>

נניח נכונות עבור $1,2,\ldots,i$ ונוכיח עבור $i+1$. לפי הנחת האינדוקציה, בתחילת האיטרציה ה-$i+1$, המשתנה LIM מצביע על האיבר $A[n-i+1]$, האיברים $M_1,\ldots,M_i$ מאכלסים את החלק הממוין של המערך A ו-M_{i+1} הוא האיבר המכסימלי באיזור אי הוודאות.

<u>הוכחת טענת האינדוקציה</u>

מהנחת האינדוקציה ידוע כי בתחילת האיטרציה ה-$i+1$, האיברים $M_1,\ldots M_i$ מאוחסנים בחלק הממוין, בסדר עולה, ומן הקוד נובע כי באיטרציה הנוכחית האיבר M_{i+1} יושווה רק עם איברים הקטנים ממנו. יהי j, $j \le n-(i+1)$, האינדקס של M_{i+1} בתחילת האיטרציה ה-$i+1$.
בדומה להוכחת מקרה הבסיס, M_{i+1} לא יזוז ממקומו עד שתתקיים ההשוואה בין $A[j+1]$ לבין $A[j]$ ואיברים אלה יוחלפו. לאחר מכן M_{i+1} יושווה לכל האיברים שמעליו, והוא יוחלף עם כל אחד מהם, עד שהוא יגיע,

בסוף האיטרציה ה-$i+1$, אל המקום ה-$n-(i+1)$ במערך A. באותו רגע, הנחת האינדוקציה מתקיימת.

מש"ל

הוכחת הנכונות מושלמת על ידי המשפט הבא:

<u>משפט</u>

בתום פעולת מיון בועות הפלט ממוין.

<u>הוכחה</u>

ההוכחה נובעת מיידית מנכונות טענת האינדוקציה עבור $i = n$.

מש"ל

<u>ניתוח סיבוכיות הזמן</u>

בקרת התכנית, למעט הקריאות לשגרה *Swap*, אינה תלויה באברי הקלט אלא באורכו בלבד. בריצה עם מערך באורך n, מתבצעות $n-1$ איטרציות, עבור $LIM = n,\ldots,2$.
בכל איטרציה כזו מתבצעות $LIM - 1 = n-1,\ldots,1$ השוואות וכל השוואה גוררת לכל היותר חילוף אחד.
סך כל ההשוואות אם כן הוא

$$\sum_{1}^{n-1} i = \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \Theta(n^2)$$

מכאן, סיבוכיות הזמן של האלגוריתם היא $\Theta(n^2)$.

<u>מיון בועות משופר</u>

<u>הגדרה</u>

היפוך (Inversion) במערך A, הוא זוג ערכים, **<u>לא בהכרח צמודים</u>**, i, j, המקיימים: $i < j$ אך $A[i] > A[j]$.

<u>דוגמא</u>

במערך

1	5	3	2

הזוג 5 ו-3, הזוג 3 ו-2, וגם הזוג 5 ו-2 הם היפוכים.

<u>שימו לב</u>

מערך המכיל היפוך אינו ממוין.

<u>מוטיבציה לאלגוריתם המשופר</u>

אלגוריתם מיון בועות מבוסס על ביצוע חילופים (*Swap*) בין איברים צמודים. כאשר מחליפים בין שני איברים צמודים, מספר ההיפוכים במערך קטן בדיוק ב-1. יהי I קלט למיון בועות.
מספר הצעדים בביצוע מיון בועות על I חסום מלמטה על ידי <u>מספר ההיפוכים</u> במערך I.

<u>דוגמא</u>

כדי למיין את המערך

1	5	3	2

יש להחליף את האיבר 5 עם 3, אחר כך מחליפים את 5 עם 2 ולבסוף את 3 עם 2.

<u>מוטיבציה לאלגוריתם המשופר</u>

קיימים מערכים שמספר ההיפוכים בהם קטן בהרבה מ-n^2. בפרט, עבור מערך ממוין, מספר ההיפוכים הוא 0. נכנה קלטים בהם מספר ההיפוכים קטן בשם קלטים נוחים.

<u>דוגמא לקלט נוח:</u>

1	5	3	2	4	7	6	8

ההיפוכים בקלט הם:

$(5,3),(5,2),(5,4)(3,2),(7,6)$

האלגוריתם המשופר מוריד את זמן הריצה עבור קלטים נוחים.

האלגוריתם המשופר

$$
\begin{array}{l}
ImpBubble(A) \\
Lim \leftarrow n \\
\textbf{while } Lim > 1 \textbf{ do} \\
\quad newLim \leftarrow 1 \\
\quad \textbf{for } j = 1 \text{ to } Lim - 1 \textbf{ do} \\
\quad\quad \textbf{if } A[j] > A[j+1] \\
\quad\quad\quad Swap(A[j], A[j+1]) \\
\quad\quad\quad newLim \leftarrow j \\
\quad\quad \textbf{End if} \\
\quad \textbf{end for} \\
\quad Lim \leftarrow newLim \\
\textbf{End while}
\end{array}
$$

איור 2.3: מיון בועות משופר

הערות לאלגוריתם המשופר

האלגוריתם המשופר מסתמך על העובדה שעבור קלטים נוחים, הגבול בין החלק הממוין, לבין אזור אי הוודאות, יכול לקטון בכל איטרציה ביותר מ-1.

בכל איטרציה, חלק המערך העליון, עליו עוברים לאחר ביצוע החילוף האחרון, הוא ממוין. לעומת זאת, אין לנו כל ערובה לגבי איזור אי הוודאות.

באלגוריתם המשופר, ערך המשתנה LIM נקבע עם תום כל איטרציה, כך שיצביע על האיבר התחתון של החילוף האחרון שהתבצע באיטרציה.

<u>דוגמא</u>

המשתנה Lim מצביע על התא המוצלל בהיר והמשתנה $newLim$ מצביע על התא המוצלל כהה:

1	5	3	2	4	7	6	8

1	3	5	2	4	7	6	8

1	3	2	5	4	7	6	8

1	3	2	4	5	7	6	8

1	3	2	4	5	6	7	8

1	3	2	4	5	6	7	8

1	3	2	4	5	6	7	8

1	2	3	4	5	6	7	8

1	2	3	4	5	6	7	8

ברור מספר הצעדים שהאלגוריתם מבצע על הקלט שבדוגמה קטן מאוד.
להלן נוכיחכונות האלגוריתם המשופר וננתח את הסיבוכיות שלו.

הוכחת סיום האלגוריתם המשופר

<u>טענה</u>

בכל מעבר בלולאה הראשית הערך של המשתנה Lim קטן לפחות ב-1.

<u>הוכחה</u>

נכונות הטענה נובעת מן הקוד (הוכיחו זאת על ידי מעקב).

מן הטענה נובע כי האלגוריתם מסתיים לאחר לכל היותר $n-1$ מעברים בלולאה הראשית.

הוכחת נכונות האלגוריתם המשופר

בטענת הנכונות שהוכחנו קודם השתמשנו בעובדה שבסיום האיטרציה ה-i ערך המשתנה Lim הוא $n-i$.

הטענה הבאה מתקבלת מן הטענה הקודמת על ידי שימוש בערך המשתנה Lim עצמו.

<u>טענה</u>

לכל $1 \le i \le n$, לאחר האיטרציה ה-i מתקיים:

$$A[n] = M_1, \ldots, A[n - Lim + 1] = M_{Lim}$$

שיוויונים אלה נשמרים עד סוף ביצוע האלגוריתם.

להלן יקראו אברים אלה (כולל M_{Lim}) בשם: החלק הממוין של המערך A.

החלק השני של המערך A:

$A[1], A[2], \ldots, A[n - Lim]$

יקרא: אזור אי הודאות

הוכחת הטענה מתקבלת על ידי התאמת ההוכחה הקודמת והיא נשארת כתרגיל לקורא.

נכונות מיון בועות (המשך)

הוכחת הנכונות מושלמת על ידי המשפט הבא:

<u>משפט</u>

בתום פעולת מיון בועות הפלט ממוין.

<u>הוכחה</u>

ההוכחה נובעת מיידית מנכונות טענת האינדוקציה בסיום המעבר האחרון בלולאת האלגוריתם.

מש"ל

סיבוכיות הזמן באלגוריתם המשופר

באלגוריתם המשופר, זמן הריצה תלוי במספר ההיפוכים בקלט ולא רק באורכו:

1. עבור קלט ממוין בסדר עולה, סיבוכיות הזמן היא לינארית. זהו המקרה הטוב ביותר.
2. סיבוכיות הWorst Case מתקבלת על ידי קלט בממוין בסדר יורד. במקרה זה, באיטרציה ה i מתבצעים בדיוק $n-i$ חילופים ובסך הכל, מספר החילופים הכולל הוא, כמו באלגוריתם הקודם, $O(n^2)$. אפשר להראות כי זהו גם מספר הצעדים המכסימלי שהאלגוריתם מבצע עבור קלט כלשהו.

סיכום

בהרצאה זו הצגנו שלושה אלגוריתמי מיון קלאסיים:

1. מיון הכנסה (Insertion Sort).
2. מיון בחירה (Selection Sort).
3. מיון בועות (Bubble Sort).

לכל אלגוריתם, הוכחנו את נכונותו ונתחנו את הסיבוכיות שלו במקרה הגרוע ביותר. לאכזבתנו, סיבוכיות שלושת האלגוריתמים הללו היא $\Theta(n^2)$

הרצאה 3: בנייה של אלגוריתמי מיון יעילים בשיטת הפרד ופתור (Divide and Conquer)

הרצאה זו מוקדשת להכרות עם **הפרדיגמה** (שיטה כללית) לבניית אלגוריתמים הקרויה **הפרד ופתור (Divide and Conquer)**. זוהי שיטה כללית לבניית אלגוריתמים רקורסיביים. אנו נציג את השיטה באופן כללי ולאחר מכן נדגים אותה על ידי בנייה של אלגוריתם מיון יעיל הקרוי **מיון באמצעות מיזוג (Merge-sort)**. סיבוכיות הזמן של האלגוריתם היא $\Theta(n\log n)$.

הפרדיגמה -הפרד ופתור

פרדיגמה (paradigm) היא שיטה כללית לפתרון סוג מסוים של בעיות. במדעי המחשב אנו מדברים על פרדיגמות של תכנון אלגוריתמים כגון: הפרד ופתור, תכנון דינמי ועוד, ועל פרדיגמות של תכנות כגון Top Down, תכנות מונחה עצמים וכו"ו.

בהרצאה זו, נדון בפרדיגמה **הפרד ופתור (Divide and conquer)**, המשמשת לתכנון **אלגוריתמים רקורסיביים**.

שלבי הפרדיגמה **הפרד ופתור** מוצגים בשקף הבא:

הפרד ופתור - הצגה אינטואיטיבית

1. הצג שיטה (*DirectSolve*) לפתרון ישיר של קבוצת קלטים. איברי קבוצה זו נקראים: **מקרי קצה**.
2. הצג שיטה (*Break*) לפרוק כל קלט מורכב, אשר אינו נמנה על מקרי הקצה, למספר תת קלטים שיקראו: **תת-קלטים ראשיים**.
3. הפעל את *GenericD & W* על כל אחד מתת הקלטים הראשיים.
4. הצג שיטה (*Combine*) לחישוב פתרון לקלט המקורי בעזרת פתרוני תת הקלטים הראשיים.

הפרד ופתור קוד דמה גנרי

$GenericDC(I)$
  if $I \in Base$
    $out(DirectSolve(I))$
  $Break(I, I_1, \dots, I_l)$
  for $\leftarrow 1$ **to** l
    $O_i \leftarrow GenericDW(I_i)$
  end for
    $O \leftarrow Combine(O_1, \dots, O_l)$
  $out(O)$

איור 3.1: הפרד ופתור גנרי

הערה

תת הקלטים הראשיים, הם מקרי קצה או שהם ניתנים לפרוק נוסף. אם נעקוב אחרי שרשרת הקריאות הרקורסיביות, נוכח כי למעשה, כל בעיה מתפרקת למספר מקרי קצה אשר כל אחד מהם נפתר בעזרת קריאה ל *DirectSolve*. הפלטים של הקריאות האלה משולבים לפלט לבעיה המקורית בעזרת קריאות ל *Combine*.

דוגמא: חישוב מספר הצמתים בעץ

יש להציג אלגוריתם לפתרון הבעיה הבאה:

קלט: עץ בינארי.

פלט: מספר הצמתים, צמתים פנימיים ועלים, בעץ.

רעיון האלגוריתם:

1. אם העץ **ריק** יש בו 0 צמתים (מקרה קצה).
2. בעץ **שאינו ריק** מספר הצמתים שוה לסכום מספר הצמתים בתת העץ השמאלי (תת בעיה מס' 1), מספר הצמתים בתת העץ השמאלי (תת בעיה מס' 2) ועוד אחד (השורש).

אלגוריתם 3.1

$Count(T)$
  if $T = \phi$ $out(0)$
  $c_l \leftarrow Count(T_l)$
  $c_r \leftarrow count(T_r)$
  $out(c_l + c_r + 1)$

איור 3.2: האלגוריתם Count

נכונות אלגוריתמים רקורסיביים

מאחר ש**האלגוריתם** הוא **רקורסיבי** ואין בו לולאות, אין אפשרות להוכיחו באינדוקציה על המעברים בלולאה. במקום זאת, נשתמש **באינדוקציה על אורך הקלט**.

ה"מתכון" להוכחות כאלה, מתואר בשקף הבא.

שימו לב: ה"מתכון" הזה, מתאר שיטה לבניית הוכחות נכונות. אי-אפשר להבטיח כי השיטה תעבוד בכל מקרה. עבור כל הוכחה, צריך לוודא כי היא אכן אינה שגויה.

"מתכון" להוכחת נכונות של אלגוריתמים רקורסיביים

בסיס האינדוקציה: יש להוכיח ישירות, כי האלגוריתם מחזיר פתרון נכון כאשר הוא נקרא עם כל אחד ממקרי הקצה.

צעד האינדוקציה: יש להניח כי בכל קריאה רקורסיבית לתת-בעיה ראשית, המוכתבת על-ידי הקוד, האלגוריתם מחזיר פתרון נכון. בהסתמך על הנחה זו, יש להוכיח כי האלגוריתם מחזיר פתרון נכון לבעיה המקורית.

אלגוריתמים א' הרצאה ג - פרופ' עמוס ישראלי

הוכחת נכונות האלגוריתם *Count*

משפט 3.1

אלגוריתם *Count* מקבל כקלט עץ בינארי T ומחשב את מספר הצמתים בעץ T.

הוכחה

נוכיח את נכונות הטענה באינדוקציה על מספר הצמתים בעץ T.

בסיס: $(|T|=0)$

זהו מקרה הקצה. מן הקוד נובע כי האלגוריתם מחזיר 0.

אלגוריתמים א' הרצאה ג - פרופ' עמוס ישראלי

צעד האינדוקציה

נניח כי האלגוריתם מחשב באופן נכון את מספר הצמתים בכל עץ שמספר הצמתים בו קטן מ-n.

נניח כי האלגוריתם *Count* מופעל על עץ בינארי T המקיים $|T|=n$. מן הקוד נובע כי במהלך ביצוע האלגוריתם $Count(T)$ מתבצעות הקריאות הרקורסיביות $Count(T_l)$ ו-$count(T_r)$.

שורש העץ T לא נכלל באף אחד משני העצים האלה ולכן מתקיים: $|T_l|<n$ וגם $|T_r|<n$. מספר הצמתים בכל אחד מתת העצים הללו קטן ממש מ-n.

אלגוריתמים א' הרצאה ג - פרופ' עמוס ישראלי

לפי הנחת האינדוקציה הקריאות הרקורסיביות מחזירות את מספר הצמתים בכל אחד מתת העצים. נכונות האלגוריתם נובעת מיידית מן הקוד.

מש"ל

אלגוריתמים א' הרצאה ג - פרופ' עמוס ישראלי

סיבוכיות האלגוריתם *Count*

להלן נראה ב-3 שלבים כי סיבוכיות האלגוריתם היא לינארית:

1. בכל ביצוע של האלגוריתם מתבצע מספר קבוע של צעדים - זכרו שכאן אנו סופרים כל קריאה רקורסיבית כצעד חישוב אחד, כי הצעדים שהקריאות הרקורסיביות מבצעות נספרים בקריאה עצמה.
2. כאשר האלגוריתם *Count* נקרא עם קלט T (T הוא עץ בינרי) מספר הקריאות לשיטה המתבצעות הוא $2|T|+1$. כדי לראות זאת, נשים לב כי כל קריאה, למעט הקריאה הראשונה, מתאימה לאחת

אלגוריתמים א' הרצאה ג - פרופ' עמוס ישראלי

הקשתות של העץ T, כולל קשתות המובילות לצמתי *Null*. מכאן נובע כי מספר הקריאות הרקורסיביות הוא בדיוק $2T$. למספר זה יש להוסיף את הקריאה הראשונה וכך מתקבל המספר $2|T|+1$.

3. נניח מספר צעדי החישוב שכל קריאה רקורסיבית מבצעת, חסום על ידי קבוע c. מכאן נובע מיידית כי סיבוכיות האלגוריתם חסומה על ידי

$$2(|T|+1)c=\Theta(|T|)$$

אלגוריתמים א' הרצאה ג - פרופ' עמוס ישראלי

אלגוריתמים א' הרצאה ג - פרופ' עמוס ישראלי

סיבוכיות אלגוריתם רקורסיבי

אפשר לחשב סיבוכיות של אלגוריתם רקורסיבי על ידי מציאת פונקציה רקורסיבית המתאימה למספר הצעדים שהאלגוריתם מבצע. במקרים רבים נח יותר לנקוט בשיטה בה חישבנו את סיבוכיות האלגוריתם *Count*. שלבי השיטה הם:

1. הערכת מספר הפעולות שקריאה יחידה לאלגוריתם מבצעת, בתלות באורך הקלט. בהערכה זו, כל קריאה רקורסיבית תחושב כצעד יחיד. במקרים רבים מספר הפעולות הזה הוא קבוע.

2. הערכת מספר הקריאות הרקורסיביות לאלגוריתם המתבצעות, בתלות באורך הקלט.
3. חסימת מספר צעדי החישוב המתבצעים על ידי האלגוריתם כולו, על ידי הכפלת מספר הקריאות לאלגוריתם, כפי שחושב בשלב 2, במספר צעדי החישוב המתבצעים בקריאה יחידה, כפי שחושב בסעיף 1.

<u>הערות</u>

כל בעיה הניתנת לפתרון בעזרת אלגוריתם רקורסיבי, אפשר לפתור גם בעזרת אלגוריתם איטרטיבי.

אלגוריתם כזה, יצטרך לפרק את הקלט לתת הקלטים הראשיים, ולחזור על התהליך, עבור כל תת-קלט, עד שכל תת-קלט יפורק למקרי הקצה.

לאחר מכן, יהיה על האלגוריתם לפתור את כל מקרי הקצה ולבסוף להשתמש בפתרונות מקרי הקצה כדי לבנות פתרון לכל אחד מתת הקלטים עד להשגת פתרון לקלט המקורי.

תכנות מוחצן (Explicit) של שלבים אלה הוא מייגע וצפוי לשגיאות רבות הנחסכות על ידי האופי ה״אוטומטי״ של תכנות רקורסיבי.

מצד שני, השימוש הכבד שאלגוריתמים רקורסיביים עושים במנגנון הקריאה לשגרה, אינו זול ולכן אלגוריתמים רקורסיביים הם, בדרך כלל, קצת פחות יעילים.

לאחר שהצגנו את השיטה, נפנה להשתמש בה כדי להציג את אלגוריתם **מיון באמצעות מיזוג (Merge Sort)** .

<u>מיון באמצעות מיזוג (Merge Sort)</u>

אלגוריתם מיון באמצעות מיזוג (Merge Sort) מבוסס על תבנית הפרד ופתור. כדי למיין מערך A בן n איברים יש לבצע:

1. **מקרי קצה -** מערך באורך 1, תמיד ממוין (*Directsolve*) .
2. **לחלק** את A לשני תת מערכים באורך $n/2$ כל אחד (*Break*).
3. **למיין באופן רקורסיבי** כל אחד מתת המערכים.
4. **למזג (merge)** את שני תת המערכים הממוינים למערך ממוין אחד. (*Combine*).

<u>אלגוריתם מיזוג (Merge)</u>

אלגוריתם המיזוג הוא אלגוריתם **איטרטיבי**, אשר משמש כשיטת איחוד תת הפתרונות (*Combine*) באלגוריתם *MergeSort* .

הנחות: A ו-B הם מערכים ממוינים שגודלם m ו-n בהתאמה. ממזגים את A ואת B אל תוך מערך C שארכו $m+n$.

<u>תיאור האלגוריתם</u>

בתחילת ביצוע האלגוריתם מאתחלים שלושה מצביעים (Pointers) כך שיצביעו אל תחילת שלושת המערכים. בכל שלב של האלגוריתם משווים בין שני האיברים אליהם מכוונים המצביעים ל-A ול-B ומעבירים את הקטן מבין השנים לאיבר עליו מכוון המצביע ל - C. לאחר מכן, מקדמים את המצביע שהצביע על האיבר שהועבר ואת המצביע של מערך C. כדי למנוע גלישה מעבר לגבולות A ו-B, מוסיפים לכל אחד מהם את **איבר חוסם** שערכו ∞.

<u>אלגוריתם *Merge*</u>

```
merge(A,B)
  A[m+1] ← ∞, B[n+1] ← ∞
  ap ← 1, bp ← 1, cp ← 1
  while (cp < m+n+1)
    if A[ap] ≤ B[bp]
      C[cp] ← A[ap]; ap++
      Else C[cp] ← B[bp]; bp++
    end if
    cp++
  end while
out(C)
```

<u>איור 3.3: האלגוריתם *merge*</u>

<u>הערה</u>

הוספת האיברים החוסמים מפשטת את הצגת האלגוריתם.

<u>נכונות אלגוריתם *merge*</u>

<u>משפט 3.2</u>

לכל $1 \le i \le m+n$, לאחר i מעברים בלולאה הראשית, המערך $C[1...i]$ מכיל תמורה ממוינת של i האיברים הקטנים ביותר מבין איברי המערכים A ו-B. המצביע pa (בהתאמה pb) מצביע על האיבר הראשון במערך A (בהתאמה B) שטרם העבר למערך C. ערכו של המצביע pc הוא $i+1$.

<u>הוכחה</u>

תרגיל בית

<u>משפט 3.3</u>

אלגורתם המיזוג הוא נכון.

<u>הוכחה</u>

תרגיל בית.

סיבוכיות אלגוריתם *merge*

משפט 3.4

סיבוכיות אלגוריתם המיזוג היא לינארית בסכום אורכי המערכים $t_{merge}(n,m)=\Theta(n+m)$.

הוכחה

תרגיל בית.

אלגוריתם המיון - תאור

מקרי קצה

מדרך הטבע, כל מערך עם איבר אחד הוא **ממוין**. מקרי הקצה של אלגוריתם המיון הם אם כן **מערכים עם איבר יחיד.**

חלוקה לתת-קלטים ראשיים

נחלק את מערך הקלט לחלק תחתון ובו $\lfloor n/2 \rfloor$ איברים, ולחלק עליון ובו $n-\lfloor n/2 \rfloor$ איברים. בהנתן פתרונים, לשני תת-הקלטים הישירים, כלומר שני תת-מערכים ממוינים, נשתמש באלגוריתם המיזוג, שהצגנו קודם, כדי לקבל מהם מערך ממוין המכיל את איברי הקלט המקורי.

אלגוריתם המיון - הקוד

$$
\begin{array}{l}
mergesort(A) \\
\quad \textbf{if}\ |A|=1\ out(A) \\
\quad A_l = A[1...\lfloor n/2 \rfloor] \\
\quad A_r = A[\lfloor n/2 \rfloor+1...n] \\
\quad B_l = mergesort(A_l) \\
\quad B_r = mergesort(B_r) \\
\quad C \leftarrow merge(B_l, B_r) \\
\quad out(C)
\end{array}
$$

איור 3.4: מיון מיזוג (Merge Sort)

דוגמא

בשקפים הבאים מוצגת דוגמא מפורטת לשימוש במיון מיזוג. בדוגמא זו, קריאות רקורסיביות מוצגות כאילו נעשו במקביל. למעשה, קריאות אלה מתבצעות באופן סדרתי.

	Sort(1)							
In (sort1)	2	7	5	3	6	8	4	1
	Sort(2)				Sort(2)			
In (sor2)	2	7	5	3	6	8	4	1
	Sort(3)		Sort(3)		Sort(3)		Sort(3)	
In (sort3)	2	7	5	3	6	8	4	1
	Sort	Sort	Sort	Sort	Sort	Sort	Sort	Sort
In(sort4)	2	7	5	3	6	8	4	1
Out(sort4)	2	7	5	3	6	8	4	1

	Mer(3)		Mer(3)		Mer(3)		Mer(3)	
In(merge3)	2	7	5	3	6	8	4	1
Out(merge3)	2	7	3	5	6	8	1	4
Out(sort3)	2	7	3	5	6	8	1	4
	Merge(2)				Merge(2)			
In(merge2)	2	7	3	5	6	8	1	4
Out(merge2)	2	3	5	7	1	4	6	8
Out(sort2)	2	3	5	7	1	4	6	8
	Merge(1)							
In(merge1)	2	3	5	7	1	4	6	8
Out(merge1)	1	2	3	4	5	6	7	8
Out(sort1)	1	2	3	4	5	6	7	8

איור 3.5: סדר הקריאות בהפעלת מיון מיזוג

נכונות מיון מיזוג

משפט 3.5:

לכל $n>0$, אלגוריתם מיון-מיזוג ממיין כל קלט חוקי באורך n.

הוכחה

נוכיח את המשפט באינדוקציה על **אורך הקלט** n.

בסיס $(n=1)$

במקרה זה, הקלט ממוין באופן טריויאלי ולכן הקלט שווה לפלט. מן הקוד נובע כי הבקרה תוחזר מייד עם פלט שווה לקלט.

מש״ל

צעד האינדוקציה $(n>1)$

נניח כי מיון מיזוג מחזיר פלט ממוין בכל קריאה עם קלט שארכו **קטן** מ-n ונניח כי האלגוריתם נקרא עם קלט שארכו n.

מן הקוד נובע כי, אם $n>1$, מערך הקלט מחולק ל**שני** תת-קלטים ראשיים: A_l, שארכו $\lfloor n/2 \rfloor$ ו-A_r שארכו $n-\lfloor n/2 \rfloor$. ארכו של כל תת-קלט ראשי **קטן** מ-n ולכן, לפי הנחת האינדוקציה, מערכי הפלט B_l ו-B_r ממוינים בסדר עולה.

צעד האינדוקציה (המשך)

לאחר מכן, *mergesort* מפעיל את השיטה *merge* על המערכים B_l ו-B_r ומחזיר את המערך הממוזג.

מאחר שהמערכים B_l ו-B_r ממוינים, נובע מנכונות האלגוריתם *merge* כי מערך הפלט ממוין בסדר עולה.

מש״ל

ניתוח סיבוכיות מיון-מיזוג

כדי לחשב את סיבוכיות האלגוריתם, נציג **תחילה את הסיבוכיות מערכת משוואות רקורסיביות**

נסמן ב $t(n)$ את הזמן הדרוש למיון-מיזוג מערך בן n איברים.

הזמן הנדרש לכל קריאה רקורסיבית הוא כמובן $t\left(\frac{n}{2}\right)$.

כבר הוכחנו כי מיזוג שני מערכים ממוינים באורך $n/2$ כל אחד אורך זמן ליניארי dn כאשר d הוא קבוע הפרופורציה.

הצגת סיבוכיות מיון-מיזוג על ידי מערכת משוואות רקורסיביות

נניח כי c הוא הזמן הקבוע הנדרש למיון מערך באורך 1 ונקבל:

$$t(n) = 2t\left(\frac{n}{2}\right) + dn$$

$$t(1) = c$$

כאשר d הוא קבוע הפרופורציה באלגוריתם המיזוג ו c הוא הזמן הקבוע הנדרש למיון מערך באורך 1.

סיבוכיות מיון מיזוג

משפט 3.6

סיבוכיות מיון מיזוג היא $\Theta(n\log n)$.

הוכחה

מטעמים טכניים, נניח כי $n = 2^m$ עבור m שלם כלשהו. (שימו לב: $m = \log n$)

נוכיח באינדוקציה על m כי

$$t(2^m) = a2^m + dm2^m$$

עבור a שלם כלשהו ו- d הוא הקבוע המופיע במשוואות הרקורסיה.

בסיס ($m = 0$, $n = 2^m = 1$)

במקרה זה, נתון כי $t(1) = c$ והטענה נובעת מייד.

צעד האינדוקציה

נניח כי עבור $n = 2^m$ מתקיים

$$t(2^m) = a2^m + dm2^m$$

נתבונן ב $t(2^{m+1})$. לפי המשוואה הרקורסיבית מתקיים:

$$t(2^{m+1}) = 2t(2^m) + d2^{m+1}$$

נציב במשוואה את הביטוי עבור $t(2^m)$ מטענת האינדוקציה ונקבל:

$$t(2^{m+1}) = 2\left[a2^m + dm2^m\right] + d2^{m+1}$$
$$= a2^{m+1} + dm2^{m+1} + d2^{m+1}$$
$$= a2^{m+1} + d(m+1)2^{m+1}$$
$$= an + dn\log n$$

משי"ל

מיון מיזוג - סיכום

מיון מיזוג משיג סיבוכיות $\Theta(n\log n)$.

בהרצאה הבאה, אנו נוכיח כי זוהי הסיבוכיות האופטימלית עבור כל אלגוריתם מיון **מבוסס השוואות**.

אלגוריתם מיון נוסף המשיג סיבוכיות זו הוא **מיון ערמה (Heap Sort)** שאותו לא נלמד בקורס.

סיכום ההרצאה

פתחנו את ההרצאה בהצגת שיטת **הפרד ופתור** המהווה בסיס לבניית אלגוריתמים רקורסיביים.

בהמשך ההרצאה, הדגמנו את השימוש בשיטה על ידי בניית אלגוריתם **מיון באמצעות מיזוג** .

המשכנו את ההרצאה בהוכחת נכונות האלגוריתם תוך שימוש באינדוקציה על **אורך הקלט**.

בסיום ההרצאה, הוכחנו כי סיבוכיות האלגוריתם היא $\Theta(n\log n)$. זהו אלגוריתם המיון שהסיבוכיות שלו היא הטובה ביותר מבין כל האלגוריתמים שנכיר.

הרצאה 4: מיון מהיר (Quicksort)

הרצאה זו מוקדשת לאלגוריתם **מיון מהיר (Quick Sort)**. במהלך ההרצאה נציג את האלגוריתם, נוכיח את נכונותו וננתח את הסיבוכיות שלו.

מיון מהיר מקובל היום כאלגוריתם המיון ש**מהירותו הממוצעת** היא הגבוהה ביותר מבין כל אלגוריתמי המיון המוכרים.

בנוסף, אלגוריתם זה, קל ביותר לתכנות. כתוצאה, רוב המערכות המציעות אלגוריתם מיון מערכתי משתמשות במיון מהיר.

חסרוננו העיקרי של המיון המהיר: סיבוכיות הזמן במקרה הגרוע ביותר היא $\Theta(n^2)$.

מיון מהיר – תכונות וסימונים

1. לאורך כל ההרצאה אנו מניחים כי הקלט למיון הוא המערך A ובו n איברים **נבדלים** (שונים זה מזה). התאמת האלגוריתם למקרה שבמערך הקלט יש כמה איברים שווים זה לזה אינה קשה.
2. הרעיון במיון מהיר הוא לחסוך ככל האפשר בצעדי חישוב. כדי לעשות זאת, נבצע את כל החישובים תוך שימוש במערך A בלבד, ללא שימוש במערכי עזר.
3. לכל $1 \le i \le j \le n$ נסמן את תת המערך A, בין המשתנה i למשתנה j על ידי $A[i...j]$

שלבי מיון מהיר

1. בחר איבר כלשהו מן המערך **כציר** (באנגלית **Pivot**). (תכף נסביר איך הציר יבחר)
2. השתמש בציר שנבחר כדי לבצע **חלוקת ציר**: כל האיברים הקטנים מן הציר בתת המערך התחתון. כל האיברים הגדולים מן הציר או שווים לו (כולל הציר עצמו) בתת המערך העליון.
3. הפעל מיון מהיר באופן רקורסיבי על כל אחד מתת המערכים.
4. האלגוריתם מתבצע **במקום** (באנגלית **in place**), כלומר בתוך מערך הקלט A.

חלוקת ציר

חלוקת הציר, השגרה *partition* מקבלת כארגומנטים את מערך קלט ואת איבר הציר p.

במהלך חלוקת הציר מחלקים את מערך הקלט A לשני תת מערכים:

1. תת המערך התחתון, $A_<$ - מכיל את כל איברי המערך A הקטנים מ-p.
2. תת המערך העליון, $A_\geq$ - מכיל את כל איברי המערך A הגדולים או שווים מ-p.

תכונות חלוקת הציר

1. יחסי הגודל בין $A_\geq$ לבין $A_<$ אינם קבועים מראש (תלויים ב-p).
2. כל אחד מאיברי $A_<$ קטן מכל אחד מאיברי $A_\geq$.

לאחר ביצוע חלוקת הציר, קוראים למיון מהיר באופן רקורסיבי על $A_<$ ועל $A_\geq$.

תכונות חלוקת הציר מבטיחות כי המערך המתקבל על ידי שרשור הפלטים של הקריאות הרקורסיביות מהווה פתרון לבעיה המקורית ללא צורך בעיבוד נוסף.

מיון מהיר – קוד האלגוריתם

להלן קוד האלגוריתם.

```
quicksort(A,i,j)
 If (i = j) out(A)
  p ← choosepivot(A,i)
   boundary ←
              partition(A,i,j,p)
  quicksort(A,i,boundary − 1)
  quicksort(A,boundary,j)
  out(A)
```

איור 4.1: מיון מהיר (Quick Sort)

בחירת הציר

השיטה לבחירת הציר היא מאוד פשוטה: נבחר כציר את האיבר הגדול מבין שני איברי המערך הראשונים. בדרך זו מבטיחים כי בכל אחד מתת המערכים יהיה לפחות איבר אחד:

- האיבר **הקטן** מבין השניים, בחלק התחתון.
- האיבר **הגדול** מביניהם (הציר עצמו), בחלק העליון.

השיטה *choosepivot*

להלן הקוד:

```
choosepivot(A,i)
  out(max(A[i], A[i+1]))
```

איור 4.2: השגרה *choosepivot*

הסיבוכיות

סיבוכיות השגרה *choosepivot* היא (כמובן) קבועה.

שימו לב

ישנן כמה דרכים לבחור ציר. לכל קלט נתון, כל שיטה לבחירת ציר מניבה ביצוע שונה.

אלגוריתם חלוקה פשוט

לאחר בחירת הציר, המערך מחולק לפי ערך הציר. הדרך הפשוטה יותר דורשת שני מעברים על מערך הקלט ושימוש במערך עזר.

במעבר הראשון על מערך הקלט, משוים כל אחד מאיבריו לציר וכל איבר הקטן מן הציר, או שוה לו מעבירים אל מערך העזר. את האיברים הגדולים מן הציר, או שווים לו, לא מזיזים. האיברים שהועברו מהווים את תת המערך התחתון.

אלגוריתם חלוקה פשוט (המשך)

במעבר השני, מעבירים את איברי הקלט הנותרים להמשך מערך העזר. איברים אלה מהווים את תת המערך העליון.

במהלך ביצוע האלגוריתם מבצעים **שני מעברים** על הקלט. סיבוכיות הזמן היא $\Theta(n)$.

שאלות

האם אפשר לשפר את הסיבוכיות?

האם אפשר לשפר את זמן הביצוע?

תרגיל: הראו כיצד אפשר לוותר על המעבר השני.

אלגוריתם חלוקה יעיל

אפשר לבצע את אלגוריתם החלוקה ללא שימוש במערך עזר (In Place) תוך חסכון בהזזת איברים ממקומם. בדרך זו חוסכים הן בזמן והן במקום, אך החסכון הוא בגורם קבוע בלבד.

החלוקה נערכת בשלבים, בכל שלב, מוצאים זוג איברים אשר אינם נמצאים בחלק המתאים. האיבר הראשון, נמצא בחלק התחתון אך הוא גדול או שווה מן הציר. האיבר השני, נמצא בחלק העליון אך הוא קטן מן הציר. לאחר מציאת זוג האיברים מבצעים החלפה *Swap* והשלב נגמר.

חלוקה במקום - תאור

באלגוריתם זה, משתמשים בשני מחוונים המכונים l ו-r.

לפני תחילת החלוקה, l מצביע אל האיבר הראשון (השמאלי ביותר) של המערך ו-r מצביע אל האיבר האחרון (הימני ביותר) של המערך.

בכל שלב, l מקודם ימינה עד שנמצא איבר הגדול מן הציר או שוה לו. לאחר מכן, r מקודם שמאלה עד שנמצא איבר קטן ממש מן הציר. כאשר נמצאו שני איברים כאלה, בודקים האם l ו-r חלפו זה על פני זה, כלומר האם $l \geq r$, אם כן, החלוקה הסתיימה. אם לא, מבצעים החלפה (*Swap*) בין $A[l]$ לבין $A[r]$ וממשיכים לשלב הבא.

חלוקה במקום - הדגמה

קלט:

שימו לב: הציר הוא 4

1	4	9	5	3	1	2	4	8	2	7	3	2	1
l													r

שלב 1.1: זיהוי האיברים להחלפה

1	4	9	5	3	1	2	4	8	2	7	3	2	1
	l												r

שלב 1.2: ביצוע ההחלפה

1	1	9	5	3	1	2	4	8	2	7	3	2	4
	l												r

שלב 2.1: זיהוי האיברים להחלפה

1	1	9	5	3	1	2	4	8	2	7	3	2	4
		l										r	

שלב 2.2: ביצוע ההחלפה

1	1	2	5	3	1	2	4	8	2	7	3	9	4
		l										r	

שלב 3.1: זיהוי האיברים להחלפה

1	1	2	5	3	1	2	4	8	2	7	3	9	4
			l								r		

שלב 3.2: ביצוע ההחלפה

1	1	2	3	3	1	2	4	8	2	7	5	9	4
			l								r		

שלב 4.1: זיהוי האיברים להחלפה

1	1	2	3	3	1	2	4	8	2	7	5	9	4
								l	r				

שלב 4.2: ביצוע ההחלפה

1	1	2	3	3	1	2	2	8	4	7	5	9	4
							l	r					

שלב 5: סיום

החלק העליון מסומן, $boundary = l$.

1	1	2	3	3	1	2	2	8	4	7	5	9	4
							r	l					

אלגוריתם החלוקה - הפרמטרים

A - מערך הקלט.

i ו j - מציינים את גבולות מערך הקלט.

$pivot$ - הציר, נמסר לשגרה על ידי השגרה הקוראת.

הערך המוחזר: מציין את האינדקס בו מתחיל תת המערך העליון.

אלגוריתם החלוקה - הקוד

```
Partition(A,i,j,pivot)
l ← i; r ← j
do forever
   while (A[l] < pivot) l++
   while (A[r] ≥ pivot) r--
   if (l ≥ r) out(l)l
      else swap(A[l], A[r])
   end if
end do
end
```

איור 4.3: אלגוריתם החלוקה

הוכחת סיום אלגוריתם החלוקה

טענה 4.1

אם קוראים לאלגוריתם *partiton* כאשר $l < r$ האלגוריתם מסתיים.

הוכחה

בכל מעבר בלולאה הראשית, למעט אולי במעבר הראשון מתקיים:

- המשתנה l גדל לפחות ב-1
- המשתנה r קטן לפחות ב-1

מכאן, בכל מעבר בלולאה הראשית, המרחק בין l ל-r קטן. מאחר שתנאי הסיום הוא $l \geq r$, סיום האלגוריתם מובטח.

הוכחת נכונות אלגוריתם החלוקה

טענה 4.2

עבור כל מערך A שאיבריו ניתנים להשוואה מתקיים: לאחר ביצוע אלגוריתם החלוקה:

- כל איברי המערך עד $boundary - 1$ קטנים מן הציר.
- כל איברי המערך מ$boundary$ והלאה גדולים מן הציר

הוכחה

בכל מעבר בלולאה הראשית, מיד לאחר ביצוע שתי הלולאות הפנימיות מתקיימים שני הביטויים הבאים:

1. כל האיברים בתחום $A[i...l-1]$ קטנים מן הציר. $A[l] \geq pivot$.
2. כל האיברים בתחום $A[r+1...j]$ גדולים מן הציר או שווים לו. $A[r] < pivot$.

שימו לב: במהלך המעבר הראשון יתכן כי התחומים $A[i...l-1]$, ו-$A[r+1...j]$ ריקים.

הוכחה (המשך)

בכל מעבר בלולאה הראשית, לאחר ביצוע החילוף, מתקיימים שני הביטויים הבאים:

1. כל האיברים בתחום $A[i...l]$ קטנים מן הציר.
2. כל האיברים בתחום $A[r...j]$ גדולים מן הציר או שווים לו.

בכל מעבר בלולאה שתי השמורות (ארבעת הביטויים) מתקיימות (**תרגיל:** הוכיחו את נכונות שתי השמורות בעזרת מעקב).

הוכחה (המשך)

ביצוע האלגוריתם מסתיים כאשר מתקיים $l \geq r$. שוויון בין l ל r לא ייתכן שכן במקרה זה נובע מן השמורה כי $pivot \leq A[l] = A[r] < pivot$

סתירה!

מכאן נובע כי בסיום הביצוע מתקיים $l > r$. אפשר לראות מן הקוד (הוכיחו זאת) כי בסיום ביצוע האלגוריתם מתקיים: $l = r + 1$.

עם סיום הביצוע, ערך המשתנה *boundary*, המצביע על תחילת תת המערך העליון, נקבע ל-l. נכונות המשפט נובעת מיידית מן הקוד ומקיום השמורה בסיום הביצוע.

מש"ל

מיון מהיר - הוכחת נכונות

משפט

בסיום ביצוע אלגוריתם המיון המהיר על מערך A, מערך הפלט ממוין.

הוכחה

ההוכחה באינדוקציה על אורך מערך הקלט A.

בסיס $|A| = 1$

במקרה זה, קל לראות כי שגרת המיון אינה מבצעת שום פעולה וכמובן מערך הפלט ממוין.

צעד האינדוקציה

נניח כי אלגוריתם *QuickSort* ממיין כל מערך באורך קטן או שווה n ונוכיח לגבי מערך באורך $n+1$.

יהי A מערך המקיים $|A| = n+1$ ונניח כי אלגוריתם מיון מהיר מופעל על המערך A. בתחילת הביצוע נבחר ציר ומתבצעת חלוקה.

לפי הטענה בדבר נכונות החלוקה, לאחר ביצוע החלוקה, מערך הפלט מחולק לשני חלקים לא ריקים כאשר אברי תת המערך העליון גדולים כולם מאיברי תת המערך התחתון.

צעד האינדוקציה (המשך)

לפי האלגוריתם, לאחר ביצוע החלוקה, מופעל מיון מהיר על כל אחד מתת המערכים. מאחר ששני תת המערכים אינם ריקים, אורך כל תת מערך קטן או שווה n. אנו נמצאים בתנאי טענת האינדוקציה, ומן הטענה נובע כי בסיום ההפעלה הרקורסיבית, שני תת המערכים ממוינים. מאחר שכל איברי תת המערך התחתון קטנים מכל איברי תת המערך העליון אפשר להסיק כי לאחר סיום הקריאות הרקורסיביות, המערך כולו ממוין.

מש"ל.

מיון מהיר - סיבוכיות זמן

כל ביצוע של חלוקה הוא לינארי באורך המערך המחולק וסיבוכיות הזמן תלויה ישירות בטיב הצירים שנבחרו.

במקרה הגרוע ביותר, הציר הנבחר הוא האיבר הגדול ביותר (או הקטן ביותר) במערך הממוין. במקרה זה המערך מתחלק לחלק עליון בגודל 1 ולחלק תחתון בגודל $n-1$.

אם מקרה זה חוזר על עצמו, זמני ריצת שלבי המיון הם:

$n, n-1, \ldots, 2, 1$

בסך הכל, במקרה הגרוע ביותר, סיבוכיות הזמן היא $\Theta(n^2)$.

מיון מהיר - סיבוכיות ממוצעת

נסמן ב- $T(n)$ את זמן הריצה הממוצע של מיון מהיר על קלט באורך n. נניח כי קבוצת הקלטים עליהם מחושב זמן הריצה הממוצע היא קבוצת התמורות מעל $1,2,\ldots,n$ וכי כל התמורות הן שוות הסתברות.

תזכורת:

קבוצת התמורות S_n היא קבוצת הוקטורים באורך n שאיבריהם הם $1,2,\ldots,n$. ידוע כי $|S_n| = n!$.

הסיבוכיות הממוצעת מתקבלת מן הנוסחה:

$$T(n) = \sum_{v \in S_n} p(v)t(v)$$

כאשר $t(v)$ מסמן את זמן הריצה של מיון מהיר על התמורה v. לכל תמורה $v \in S_n$ אנו מניחים כי $p(v) = 1/n!$

ואז מקבלים כי $T(n) = \frac{1}{n!}\sum_{v \in S_n} t(v)$

כדי לחשב ביטוי זה, נחלק את הקבוצה S_n ל- $n-1$ תת קבוצות כדלהלן:

חלוקת המשנה

לכל i, $1 \le i \le n$, תהי $S_n^i \subset S_n$ קבוצת כל התמורות המקיימות: אורך תת המערך התחתון שאיבריו קטנים מן הציר, $A_<$, הוא i, ואורך תת המערך העליון, $A_\ge$ הוא $n-i$. כעת אפשר לרשום כי

$$T(n) = \sum_{v \in S_n} p(v)t(v) = \sum_{i=1}^{n-1} \frac{|S_n^i|}{n!} T(S_n^i)$$

כאשר $T(S_i^n)$ מסמן את זמן הריצה הממוצע בביצוע quicksort על כל המערכים בקבוצה S_n^i.

טענה

לכל i, $1 \le i \le n$, זמן הריצה הממוצע עבור כל התמורות ב - S_n^i הוא:

$$T(S_n^i) = \underbrace{T(i)}_{\substack{\text{זמן}\\\text{דרוש}\\\text{למיון}\\\text{חלק}\\\text{תחתון}}} + \underbrace{T(n-i)}_{\substack{\text{זמן}\\\text{רושמ}\\\text{למיון}\\\text{מע}\\\text{עליון}}} + \underbrace{cn}_{\substack{\text{זמן}\\\text{דרוש}\\\text{להלוקה}}}$$

הסבר

$T(i)$ הוא הזמן הממוצע הנדרש למיון תמורה בת i איברים. נכונות הטענה נובעת מן העובדה שאם מתבצעת חלוקה, ההסתברות (הסיכוי) לכל אחת מן התמורות ב S_i, זהה.

ההסבר עבור $T(n-i)$ - זהה.

הוכחה

נתבונן בקבוצת החלקים התחתונים המתקבלים מכל הוקטורים ב - S_n^i, שאורכם הוא תמיד i. אפשר להוכיח כי בקבוצה זאת, כל התמורות ב - S_i מתקבלות באותה השכיחות. מטענה זו נובע כי הזמן הממוצע הנדרש למיון החלק התחתון בקריאה הרקורסיבית הוא $T(i)$ (המוגדר כזמן הממוצע הנדרש למיון כל התמורות ב-S_i). באופן דומה, אפשר להוכיח כי $T(n-i)$ הוא הזמן הממוצע הנדרש בקריאה הרקורסיבית למיון החלק העליון.

ברור כי זמן החלוקה הוא $\Theta(n)$.

מש"ל.

הסיבוכיות הממוצעת (המשך)

כדי לחשב את ערך $T(n)$ בדרך נכונה, עלינו לשים לב כי לכל $1 \le i < j \le n-1$ מתקיים $|S_n^i| \ne |S_n^j|$

מן הטענה הקודמת ומהגדרת הסיבוכיות הממוצעת נובע כי:

$$T(n) = \sum_{i=1}^{n-1}\left[\frac{|S_n^i|}{n!}\left(T(i)+T(n-i)\right)+cn\right]$$

$$= \sum_{i=1}^{n-1}\frac{|S_n^i|}{n!}\left(T(i)+T(n-i)\right)+ \sum_{i=1}^{n-1}\frac{|S_n^i|}{n!}cn$$

$$= \sum_{i=1}^{n-1}\left[\frac{|S_n^i|}{n!}\left(T(i)+T(n-i)\right)\right]+cn$$

טענה

$$\frac{|S_n^i|}{n!} = \frac{2i}{n(n-1)}$$

הוכחה

נשים לב לעובדה ש כדי שמערך $A \in S_n$ יקיים $A \in S_n^i$, הציר חייב להיות $i+1$.

לכל i, $1 \le i < n$, כדי ש-$i+1$ יבחר כציר צריך להתקיים:

1. $A[1]=i+1$ ו-$A[2]<i+1$ **או**
2. $A[1]<i+1$ ו-$A[2]=i+1$.

נתבונן במקרה 1:

קיימות בדיוק i אפשרויות לבחירת $A[2]$ ועבור כל בחירה כזו, יש

$(n-2)!$ אפשרויות עבור
$A[3], A[4], \ldots, A[n]$. מכאן נקבל כי
קיימים בדיוק $i(n-2)!$ המקיימים
את מקרה 1.
באופן סימטרי, קיימים בדיוק
$i(n-2)!$ המקיימים את מקרה 2.
ומשתי מסקנות אלה נקבל:

$$\frac{|S_n^i|}{n!} = \frac{2i(n-2)!}{n!} = \frac{2i}{n(n-1)}$$

<u>הסבר</u>

קל לראות כי $(n-2)!/n! = 1/n(n-1)$

<u>הסיבוכיות הממוצעת (המשך)</u>

כעת נציב את הערכים שקיבלנו
בנוסחה לחישוב $T(n)$ ונקבל:

$$T(n) = \sum_{i=1}^{n-1}\left\{\frac{2i}{n(n-1)}[T(i)+T(n-i)+cn]\right\}$$
$$= \frac{1}{2}\sum_{i=1}^{n-1}\frac{2i}{n(n-1)}[T(i)+T(n-i)+cn]$$
$$+ \frac{1}{2}\sum_{i=1}^{n-1}\frac{2(n\text{-}i)}{n(n-1)}[T(n-i)+T(i)+cn]$$

נכונות המעבר האחרון נובעת מן
העובדה שבשתי השורות האחרונות
מסכמים אותם האיברים **בסדר הפוך**.

<u>הסיבוכיות הממוצעת (המשך)</u>

נמשיך בפיתוח:

$$T(n) = \sum_{i=1}^{n-1}\left(\frac{2iT(i)+2iT(n-i)}{n(n-1)}\right.$$
$$+ \frac{2nT(n-i)+2nT(i)}{n(n-1)} +$$
$$\left.- \frac{2iT(n-i)+2iT(i)}{n(n-1)}\right) +$$
$$\frac{1}{2n(n-1)}\sum_{i=1}^{n-1}2i$$

כעת נשים לב שהביטוי העליון
בסוגריים בשורה הראשונה והשלישית
מבטלים זה את זה, וערכו של הביטוי
בשורה הרביעית הוא cn ונקבל:

$$T(n) = \frac{1}{2}\left(\sum_{i=1}^{n-1}\frac{2n[T(n-i)+T(i)]}{n(n-1)}\right) + cn$$

$$= \frac{1}{(n-1)}\left(\sum_{i=1}^{n-1}T(i)+T(n-i)\right) + cn$$

מאחר ששני הביטויים בתוך הסכום
מסתכמים לאותו מספר נקבל: את
נוסחת הנסיגה הבאה:

$$T(n) = \frac{2}{n-1}\left(\sum_{i=1}^{n-1}T(i)\right) + cn$$

<u>נוסחת הנסיגה</u>

הגענו לנוסחת נסיגה עבור הסיבוכיות
הממוצעת של מיון מהיר:

$$T(n) = \frac{2}{n-1}\left(\sum_{i=1}^{n-1}T(i)\right) + cn$$

בשקפים הבאים נוכיח כי
$T(n) = \Theta(n\log n)$.

<u>למה</u>

סיבוכיות נוסחת הנסיגה

$$T(n) = \frac{2}{n-1}\sum_{i=1}^{n-1}T(i) + cn$$

עבור $c \ge 1$ היא $\Theta(n\log n)$.

<u>הוכחה</u>

נגדיר d=4c ונוכיח באינדוקציה על n
כי לכל n מתקיים, $T(n) \le dn\log n$
בסיס: עבור n=2 ובהנחה ש $T(1)=1$,

$$T(2) = 2T(1) + 2c = 2\left(\underbrace{1}_{=T(1)} + c\right) \le$$
$$\le 8c = 2d\log 2$$

צעד: נניח כי לכל $i < n$ מתקיים
$T(i) \le di\log i$ ונתבונן ב $T(n)$:

$$T(n) = \frac{2}{n-1}\sum_{i=1}^{n-1}T(i) + cn$$

כעת נשתמש בהנחה ונקבל:

$$T(n) \le \frac{2d}{n-1}\sum_{i=1}^{n-1}i\log i + cn$$

כדי להעריך את הסכום, נפרק אותו
לשני חלקים ונקבל:

$$T(n) \le \frac{2d}{n-1}\left(\sum_{i=1}^{n/2}i\log\frac{n}{2} + \sum_{i=n/2+1}^{n-1}i\log n\right) + cn =$$

ומאחר ש $\log(n/2) = \log n - 1$ נקבל:

$$= \frac{2d}{n-1}\left(\sum_{i=1}^{n/2}i(\log n - 1) + \sum_{i=n/2+1}^{n-1}i\log n\right) + cn =$$

$$= \frac{2d}{n-1}\left(\sum_{i=1}^{n-1}i\log n - \sum_{i=1}^{n/2}i\right) + cn$$

$$= \left(\frac{2d\log n}{n-1}\right)\left(\frac{n(n-1)}{2}\right) -$$
$$\left(\frac{2d}{n-1}\right)\left(\frac{\frac{n}{2}\left(\frac{n}{2}-1\right)}{2}\right) + cn$$

$$= dn\log n - \frac{dn^2}{4(n-1)} - \frac{dn}{2(n-1)} + cn$$

$$\le dn\log n - \frac{dn^2}{4n} - \frac{dn}{2n} + cn$$

$$= dn\log n - \Bigg(\underbrace{\frac{d}{4} - c}_{=0}\Bigg)n - \frac{d}{2} \le dn\log n$$

<u>הערות נוספות</u>

1. הוכחנו כי $T(n) = O(n\log n)$.
2. אפשר להראות כי המקרה הטוב ביותר מתקבל כאשר בכל שלב, כל קובץ מתחלק בדיוק ל2. במקרה זה מתקיים:

$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$

משוואה זו היא $\Theta(n\log n)$ ולכן מתקיים $T(n) = \Theta(n\log n)$.

3. קריאות רקורסיביות גורמות לפעולות מחסנית רבות שהן אינן יעילות עבור ערכי n קטנים. לכן,

עבור $n<10$ (למשל) מוצע להשתמש במיון ריבועי כגון מיון בועות.

4. בספר מוצעת גירסה שונה למיון מהיר. בגירסה זו, הציר נבחר באופן אקראי ומוכחת תוצאה דומה לסיבוכיות הממוצעת.
5. בחיים האמיתיים מיון מהיר הוא **יעיל יותר** ממיון מיזוג. הסיבה לתופעה זו היא כי במיון מיזוג, בכל איטרציה מוזזים כל האיברים, בעוד שבמיון מהירת בכל איטרציה מוזזים בממוצע, מחצית האיברים. סימולציות מראות כי מיון מהיר יעיל ממיון מיזוג **פי שתים.**

<u>סיכום</u>

בהרצאה זו, הצגנו את אלגוריתם **מיון מהיר**, הוכחנו את נכונותו והראנו כי סיבוכיות האלגוריתם היא $\Theta(n^2)$. כאמור, **הסיבוכיות הממוצעת** של מיון מהיר היא $\Theta(n\log n)$. זהו אלגוריתם יעיל מאוד וגם פשוט מאוד והוא משמש פעמים רבות כאלגוריתם מערכת למיון.

<u>הרצאה 5: סיבוכיות של בעיות - סיבוכיות בעית המיון כדוגמא</u>

בהרצאה זו, נעסוק ב**סיבוכיות של בעיות**. בתחילת ההרצאה, נגדיר סיבוכיות של בעיות באופן כללי וכדוגמא נחשב את סיבוכיות בעית המיזוג.

עיקר ההרצאה, יוקדש לניתוח סיבוכיות בעית המיון. במהלכה נוכיח חסם תחתון $\Omega(n\log n)$ לבעית המיון, עבור אלגוריתמים מבוססי השוואות.

בסיכום ההרצאה ונסיק כי סיבוכיות בעיית המיון, עבור אלגוריתמי מיון **מבוססי השוואות** היא $\Theta(n\log n)$.

<u>סיבוכיות של בעיות</u>

תהי P בעיה חישובית כלשהי.

הסיבוכיות של P היא הסיבוכיות של האלגוריתם היעיל ביותר (מהיר ביותר) לפתרון P.

שימו לב: הכוונה אינה לסיבוכיות האלגוריתם יעיל ביותר **המוכר לנו** אלא לסיבוכיות האלגוריתם היעיל ביותר האפשרי.

כדי לקבוע סיבוכיות של בעיה, P עלינו להוכיח **חסם תחתון** על סיבוכיות כל האלגוריתמים לפתרון P.

כאשר קיים אלגוריתם לינארי, הוכחה כזאת אינה קשה.

<u>דוגמא 4.1 - סיבוכיות בעית המיזוג</u>

סיבוכיות בעיית המיזוג של שני מערכים ממוינים בגודל n ו-m בהתאמה, היא $\Theta(n+m)$.

<u>**הוכחה**</u>

כבר ראינו כי קיים אלגוריתם מיזוג שהסיבוכיות שלו היא $\Theta(n+m)$.

כדי להוכיח את המשפט, עלינו להראות כי **לא קיים אלגוריתם מיזוג שהסיבוכיות שלו נמוכה מ-** $n+m$.

מאחר ש**כל אלגוריתם מיזוג יוצר את וקטור הפלט** גדלו $n+m$ המשפט נובע מיידית.

מש"ל

<u>דוגמא 4.1 - סיבוכיות בעית הסכום</u>

סיבוכיות בעיית סכום האיברים במערך בגודל n, היא $\Theta(n)$.

<u>**הוכחה**</u>

ברור שקיים אלגוריתם לחישוב סכום מיזוג שהסיבוכיות שלו היא $\Theta(n)$.

כדי להוכיח את המשפט, עלינו להראות כי **לא קיים אלגוריתם מיזוג שהסיבוכיות שלו נמוכה מ-** n.

מאחר ש**כל אלגוריתם סכום חייב לבחון את כל אברי מערך הקלט** שגדלו n המשפט נובע מיידית.

מש"ל

המשפט הבא מכליל את שתי הדוגמאות שהבאנו:

<u>משפט 4.3</u>

תהי P בעיה חישובית.

1. אם לכל n קיים קלט בגודל n, כך שהפלט המתאים הוא בגודל n, או בגודל $\Omega(n)$, הסיבוכיות של כל אלגוריתם לפתרון P, היא $\Omega(n)$.
2. אם אם לכל n טבעי קיים קלט I_n, $|I_n|=n$, כך שכל אלגוריתם לפתרון P חייב לבחון כל איבר ב-I_n, הסיבוכיות של כל אלגוריתם לפתרון P, היא $\Omega(n)$.

<u>**הוכחה**</u>

ההוכחה מיידית ונשארת כתרגיל לקורא.

<u>מחלקות של בעיות בסיבוכיות לינארית</u>

קימות אם כן, שתי מחלקות גדולות של בעיות בסיבוכיות לינארית:

1. מחלקת הבעיות שקיים אלגוריתם לינארי לפתרונן, ואשר יוצרות פלט שגדלו הוא $\Theta(n)$. לדוגמא: בעיית המיזוג.
2. מחלקת הבעיות שקיים אלגוריתם לינארי לפתרונן, וכל אלגוריתם נדרש לבחון את כל הקלט. לדוגמא: בעיית הסכום.

<u>חסם תחתון לסיבוכיות בעיית המיון</u>

מה המעמד של בעיות שהאלגוריתם הטוב ביותר המוכר לנו הוא סופר לינארי (כלומר יותר מלינארי)?

בדרך כלל, הוכחת חסמים תחתונים לבעיות כאלה היא קשה ביותר.

בהרצאה זו, נכיר את מודל עץ ההחלטה, והשימוש בו, כדי לקבוע את סיבוכיות בעיית המיון מבוסס ההשוואות.

<u>סיבוכיות בעיית המיון</u>

נתבונן בבעיית המיון. לכל מופע בגודל n, גודל הפלט הוא n ולכן, סיבוכיות הזמן של בעיית המיון היא $\Omega(n)$.

בעבר, האלגוריתם הטוב ביותר הידוע למיון היה מיון בועות (Bubble Sort) שזמן הריצה שלו הוא $\Theta(n^2)$.

בזמן זה, הידע האנושי בדבר סיבוכיות בעיית המיון היה:

$$\boxed{\Omega(n)\le \text{סיבוכיות מיון} \le\Theta(n^2)}$$

במלים אחרות: פער אי הוודאות בנושא סיבוכיות מיון היה בין n לבין n^2 .

<u>סיבוכיות בעיית המיון</u>

פער אי הוודאות המוזכר לעיל נסגר על ידי שתי ההתפתחויות הבאות :

1. פותחו אלגוריתמי מיון יעילים יותר, כגון מיון בעזרת מיזוג (Merge-Sort) שהסיבוכיות שלהם היא $\Theta(n\log n)$.
2. בהרצאה זו נוכיח חסם תחתון הקרוי **"חסם תורת האינפורמציה"** של $\Omega(n\log n)$ על סיבוכיות הזמן של אלגוריתמי מיון מבוססי השוואות (זהו סוג מסוים של אלגוריתמי מיון).

אלו חסמים **הדוקים** (tight). כתוצאה נקבל: **סיבוכיות בעיית המיון מבוסס השוואות היא** $\Theta(n\log n)$.

<u>הערות</u>

1. לאורך כל ההרצאה נניח כי כל איברי מערך הקלט הם נבדלים.
2. החסם **תופס** רק עבור אלגוריתמים מבוססי השוואות.
3. החסם **אינו** מוכיח כי כל הריצות הנן באורך $n\log n$. לדוגמא : מיון בועות עבור קובץ ממוין יתבצע בזמן לינארי.

נפתח את ההרצאה במספר הגדרות :

<u>הגדרה 4.4: תמורה מעל n איברים</u>

וקטור ובו n איברים שהם המספרים 1,2,…,n מסודרים בסדר שרירותי.

<u>הגדרה 4.5: חבורת התמורות מעל n איברים</u> קבוצת כל התמורות מעל n איברים, כלומר קבוצת כל הוקטורים שאיבריהם הם $1,2,\ldots,n$ בסדר כלשהו. קבוצה זו מכונה S_n. ידוע כי $|S_n| = n!$.

<u>אלגוריתמי מיון מבוססי השוואות (המשך)</u>

<u>הגדרה 4.6:</u>

1. יהי Al אלגוריתם כלשהו המקבל כקלט מערך A. **<u>השוואה</u>** ב-Al היא הביטוי $A[i] < A[j]$, כי משווים שניים מאיברי מערך הקלט. תוצאת ההשוואה יכולה להיות T או F.

<u>הערה 1:</u> אפשר כמובן להפוך את סדר הביטוי כלומר לכתוב $A[i] > A[j]$. לשם האחידות אנו נשתדל להצמד לצורה הראשונה.

<u>הערה 2:</u> הביטוי $A[i] < 0$ אינו מוגדר כהשוואה, כי משווים איבר ממערך הקלט עם קבוע מספרי.

<u>אלגוריתמי מיון מבוססי השוואות (המשך)</u>

<u>הגדרה 4.7:</u>

אלגוריתם מיון הוא **<u>מבוסס השוואות</u>** אם כל אחת מהחלטות הבקרה שלו מתקבלת אך ורק על ידי ביצוע השוואה.

<u>הערה 1:</u>

הביטוי $A[i] < 0$ אינו מוגדר כהשוואה, כי משווים איבר ממערך הקלט עם קבוע מספרי.

<u>הערה 2:</u>

כל אלגוריתמי המיון שלמדנו עד כה הם מבוססי השוואות.

<u>אלגוריתמי מיון מבוססי השוואות (המשך)</u>

<u>הגדרה 4.8: מערך שקול</u>

מערך A בגודל n שאיבריו מספרים שלמים הוא **<u>שקול</u>** לתמורה α אם A ו-α שומרים על אותם יחסי גודל. התמורה α נקראת **<u>התמורה המייצגת</u>** את המערך A.

<u>לדוגמא:</u>

המערך 7,8,9 שקול לתמורה 1,2,3 והמערך 30,20,40 שקול למערך 2,1,3.

כל חישוב של אלגוריתם מבוסס השוואות נקבע **<u>אך ורק על פי תוצאות של השוואה</u>**, לכן כל אלגוריתם כזה, מתנהג באופן זהה על כל המערכים **<u>השקולים לתמורה מסוימת.</u>**

<u>תכונות אלגוריתמי מיון מבוססי השוואות</u>

<u>משפט 4.9</u>

תהיינה $\sigma, \pi \in S_n$ שתי תמורות שרירותיות ויהי Al אלגוריתם מיון מבוסס השוואות כלשהו. תהי $c_1, c_2, \ldots$ ($c'_1, c'_2, \ldots$ בהתאמה). סדרת השוואות המתבצעות בחישוב Al עם σ (עם π בהתאמה). אזי

1. $c_1 = c'_1$.
2. קיים אינדקס מינימלי i_0, המקיים ש $c_{i_0} = c'_{i_0}$ אך תוצאות ההשוואות הללו מנוגדות.

<u>הוכחת 1</u>

ההשוואה הראשונה מוכתבת על ידי קוד האלגוריתם ואינה מושפעת כלל מן התמורה שבארגומנט.

<u>הוכחת 2</u>

מאחר ש-Al הוא אלגוריתם מבוסס השוואות, כל החלטות הבקרה של האלגוריתם Al תלויות אך ורק בהשוואות הנערכות על ידי האלגוריתם. נתבונן בסדרות ההשוואות המתבצעות על ידי האלגוריתם בחישובים עם σ ועם π. אם שתי הסדרות זהות וגם התוצאות זהות, אזי כל פעולות האלגוריתם בשני החישובים זהות. במקרה זה מתקיים :

1. האיברים $\sigma[1]$ ו-$\pi[1]$ מגיעים לאותו מקום בפלט.
2. האיברים $\sigma[2]$ ו-$\pi[2]$ מגיעים לאותו מקום בפלט.
3. לכל i, $2 < i < n$, האיברים $\sigma[i]$ ו-$\pi[i]$ מגיעים לאותו מקום בפלט.
4. האיברים $\sigma[n]$ ו-$\pi[n]$ מגיעים לאותו מקום בפלט.

וזה כמובן לא ייתכן מפני ש-AI הוא **אלגוריתם מיון.**

מש"ל

דוגמא

אם לדוגמא נתון

σ	...	3	8	2	6	4	5	1

π	...	3	8	2	6	7	5	1

אז לאחר המיון נקבל

$out(\sigma)$	...				4	3	2	1

$out(\pi)$	...				7	3	2	1

וזאת סתירה כי הפלט $out(\pi)$ **אינו ממוין.**

עץ החלטה

בטרם ניגש להוכחת החסם התחתון עלינו להכיר מבנה נתונים תיאורטי בשם **עץ החלטה למיון n איברים** או בקיצור **עץ מיון ל- n איברים.** להלן נגדיר עץ מיון בשני שלבים.

הגדרה 4.10: עץ מסומן

עץ מסומן הוא עץ בינרי אשר כל אחד מן הצמתים שלו מסומן כדלהלן:

- כל צומת פנימי בעץ מסומן בהשוואה בין שנים מאיברי תמורת הקלט.
- שתי הקשתות היוצאות מצומת פנימי כזה מסומנות האחת ב-T והשניה ב-F.
- כל עלה בעץ מסומן על ידי תמורה $\pi \in S_n$. להלן נסמן את המסלול משורש העץ ועד לעלה המסומן בתמורה π בשם α_π.

הגדרה 4.11: מסלול תקין

מסלול α_π בעץ מסומן T נקרא **תקין** אם התמורה π **מספקת** (מאמתת) כל אחת מן ההשוואות ב-α_π.

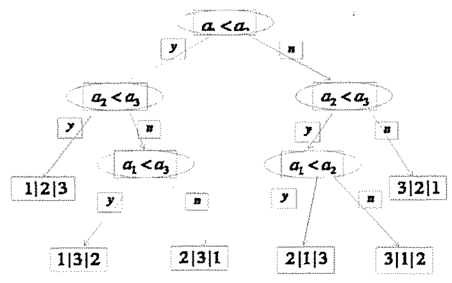
הגדרה 4.12: עץ החלטה

עץ החלטה למיון n איברים, הוא עץ מסומן T המקיים את הדרישות הבאות:

1. בעץ T יש בדיוק $n!$ עלים.
2. כל תמורה $\pi \in S_n$ מסמנת **עלה יחיד** בעץ T.
3. כל אחד מ - $n!$ המסלולים בעץ T הוא **תקין.**

דוגמא: עץ החלטה עבור S_3

בעץ שלפנינו יש להתייחס לסימון a_i כאילו כתוב $A[i]$.



איור 4.1: עץ החלטה למיון 3 איברים

בטרם תמשיכו בקריאה, ודאו שכל המסלולים בעץ T הם **תקינים.**

הערות

1. עץ החלטה אינו אלגוריתם מיון. זהו מודל תיאורטי בלבד.
2. אפשר להתייחס לעץ החלטה כאל **אלגוריתם לזיהוי תמורת הקלט.**
3. כל עץ החלטה ל - n איברים אמור לזהות כל אחת מתמורות הקלט. זיהוי תמורה π, מתבצע על ידי מעבר על המסלול α_π.
4. מאחר שמספר תמורות הקלט האפשריות הוא $n!$, לכל עץ החלטה למיון n איברים, יש $n!$ עלים.
5. בהנתן וקטור V, שכל איבריו **נבדלים** עץ המיון מזהה את התמורה המייצגת את V.

חסם תחתון לסיבוכיות אלגוריתמים מבוססי השוואות למיון

להלן נוכיח כי הסיבוכיות של כל אלגוריתם מיון מבוסס השוואות היא $\Omega(n \log n)$, כלומר הסיבוכיות חסומה מלמטה על ידי $n \log n$.

במלים פחות מדויקות אפשר לומר כי סיבוכיות בעית המיון מבוסס השוואות היא $\Omega(n \log n)$.

סקירת הוכחת החסם

הוכחת החסם תתקבל על ידי הוכחת הטענות הבאות:

טענה 1: כל אלגוריתם מבוסס השוואות למיון n איברים, Al_n, **משרה** עץ החלטה ל- n איברים, T_{Al_n} ובו $n!$ עלים.

טענה 2: סיבוכיות האלגוריתם Al_n, גדולה או שווה מעומק העץ המושרה T_{Al_n}.

טענה 3: הגובה של כל עץ החלטה למיון n איברים הוא $\Omega(n \log n)$.

בניית עץ המיון המושרה

ההוכחה הפורמלית לטענה 1, מתווה אלגוריתם לבנית עץ החלטה.

אלגוריתם הבנייה, שיכונה להלן Bal מקבל כקלט, אלגוריתם מיון Bal מבוסס השוואות Al_n ובונה את עץ ההחלטה T_{Al_n}.

שימו לב:

1. אלגוריתם הבניה, מקבל כקלט **אלגוריתם מיון.**
2. **מבנה** עץ ההחלטה המושרה, תלוי באלגוריתם המיון המשרה אותו.

תאור האלגוריתם Bal

קלט: אלגוריתם מבוסס השוואות למיון n איברים - Al_n.

פלט: עץ החלטה למיון n איברים - T_{Al_n}.

אלגוריתם Bal בונה את T_{Al_n}, תוך כדי הרצת Al_n על כל $n!$ התמורות של n איברים.

לכל תמורה $\pi \in S_n$, האלגוריתם Bal הבנייה בונה את α_π על ידי **מעקב** אחר ביצוע האלגוריתם Al_n על הקלט π.

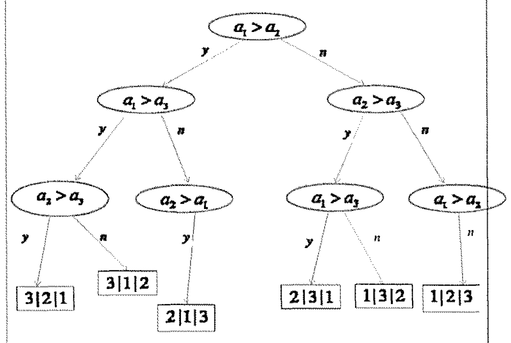
דוגמא: עץ מיון המושרה על ידי מיון בועות לשלושה איברים

בטרם נציג את Bal, נדגים את פעולתו על ידי בניית עץ המיון T_{BS_3}, המושרה על ידי מיון בועות, בגרסה הראשונה, ל-3 איברים:

```
BubbleSort(A,3)
  for Lim ← 3 downto 2
    for j = 1 to Lim − 1
      if (A[j] > A[j + 1])
        Swap(A[j], A[j + 1])
    end for
  end for
end
```

איור 4.2: מיון בועות ל-3 איברים

העץ המושרה



איור 4.3: עץ ההחלטה המושרה על ידי אלגוריתם מיון בועות ל-3 איברים

נא בדקו שכל המסלולים הם תקינים.

תיאור אלגוריתם Bal לבניית $\alpha(\Pi)$

אלגוריתם Bal עוקב אחר חישוב A_n על הקלט Π, לפי הנקודות המפורטות להלן:

1. כל הרצה, ללא תלות בתמורת הקלט, Π, נפתחת בהשוואת זוג מאיברי Π. השוואה זו מסמנת את שורש העץ T_{A_n}.
2. לאחר ביצוע ההשוואה הראשונה, אלגוריתם Bal עוקב אחרי α_Π על ידי בחירת הקשת המתאימה לתשובה שהתקבלה בחישוב A_n על Π (התלויה ב Π), כדלהלן:

אם ההשוואה מתקיימת, נבחרת הקשת המסומנת ב-y; אם לא- נבחרת הקשת המסומנת ב-n.

3. אם קשת שנבחרה, אינה מופיעה ב T_{A_n}, אלגוריתם Bal מוסיף אותה ל T_{A_n} וממשיך בבניית α_Π כאשר כל השוואה נוספת ש-A_n מבצע, תורמת צומת נוסף ל α_Π.
4. בסיום הרצת A_n על Π, אלגוריתם Bal מוסיף עלה ומסמנו על ידי Π.

שימו לב (הערה חשובה)

כל ההשוואות נעשות בין איברי תמורת הקלט ולא מתחשבים בחילופי האיברים הנעשים במהלך ביצוע האלגוריתם.

משפט 4.4

אלגוריתם Bal בונה עץ השוואה למיון n איברים.

הוכחה

אלגוריתם Bal מפעיל את האלגוריתם שבקלט A_n על כל התמורות ב-S_n אחת אחרי השניה. לפי סעיף 1 במשפט 4.3, כל הפעלה כזו נפתחת בביצוע השוואה מסוימת, ללא תלות בתמורת הקלט. ההשוואה הזאת היא כמובן ההשוואה המופיעה בשורש העץ הנבנה. מהגדרת Bal נובע כי כל התמורה הראשונה שעליה מופעל, A_n, בונה מסלול תקין בעץ. נניח כי עד לביצוע A_n על תמורה

כלשהי Π, אלגוריתם Bal בנה עץ שכל מסלוליו תקינים. לפי סעיף 2 במשפט 4.3 כאשר A_n מופעל על Π, לכל תמורה Σ, עליה הפעל A_n עד כה, המסלול הנבנה בעץ עבור Π, מבצע השוואה אחת הכלולה בביצוע עם Σ, אך תוצאת ההשוואה שונה. מכאן נובע כי המסלול הנבנה עבור Π <u>**שונה מכל אחד מן המסלולים שנבנו בעץ עד כה.**</u> כתוצאה מכך, המסלול הנבנה עבור Π הוא תקין וסך הכל אלגוריתם הבניה בונה בדיוק $n!$ מסלולים כלומר:

<u>**העץ הנבנה הוא עץ השוואה.**</u> להלן נכנה את עץ ההשוואה הזה ב- T_{A_n}.

מש"ל

<u>**הוכחת טענה 2**</u>

טענה 2: סיבוכיות האלגוריתם A_n, גדולה או שווה מגובה העץ המושרה T_{A_n}.

ההוכחה מסתמכת על האלגוריתם לבנית T_{A_n}: לכל תמורה Π, המסלול $\alpha(\Pi)$ שומר את סדרת השוואות הנערכות במהלך החישוב על Π.

תהי Σ התמורה המסמנת את המסלול הארוך ביותר ב T_{A_n}. מאחר שחישוב A_n על Σ, כולל את כל ההשוואות המאוחסנות לאורך α_Σ ביחד עם צעדי חישוב נוספים, מספר הצעדים בביצוע A_n, בריצה עם Σ, גדול או שווה מאורכו של α_Σ. מכאן: סיבוכיות המקרה הגרוע ביותר של A_n גדולה או שווה מ $|\alpha_\Sigma|$.

מש"ל

<u>**חסימת גובה עץ ההשוואה**</u>

הוכחת החסם לסיבוכיות בעיית המיון תושלם על ידי חסימת גובה עץ המיון.

<u>**משפט**</u>

יהי T עץ בינארי כלשהו עם n עלים. גבהו של T הוא $\Omega(\log n)$.

<u>**הוכחה**</u>

הוכחת המשפט מופיעה בנספח להרצאה.

<u>**חסימת גובה העץ (המשך)**</u>

<u>**תוצאה**</u>

נזכר כי לכל עץ מיון יש בדיוק $n!$ עלים.

מן המשפט שהוכחנו נובע כי גבהו של כל עץ מיון הוא:

$$\Omega(\log n!)$$

מה ערכו של $\log n!$?

אי השוויונות הבאים מעידים כי

$$\log n! = \Theta(n \log n)$$

$$n \log n \geq \log(n^n) \geq \log(n!) \geq$$

$$\geq \log\left(\left(\frac{n}{2}\right)^{\frac{n}{2}}\right) = \frac{n}{2}(\log n - \log 2)$$

<u>**חסימת גובה העץ (המשך)**</u>

מנוסחת Stirling לחישוב $n!$ נובע:

$$n! \approx \left(\frac{n}{e}\right)^n$$

$$\log n! \approx n(\log n - \log e) \approx$$

$$\approx n \log n - 1.44n$$

<u>**סיכום ההוכחה**</u>

לפי **טענה 1**, כל אלגוריתם מיון **מבוסס השוואות** למיון n איברים A_n משרה עץ מיון כלשהו, T_{A_n}.

מ**טענה 2** נובע כי גובה העץ המושרה T_{A_n}, הוא חסם תחתון על סיבוכיות המקרה הגרוע ביותר של האלגוריתם המשרה, A_n.

מן המשפט האחרון שהוכחנו, נובע כי גבהו המינימלי של עץ מיון ל-n איברים חסום על ידי

$$\Omega(\log n!) = \Omega(n \log n)$$

<u>**הוכחת חסמים תחתונים בעזרת רדוקציה**</u>

סיימנו את הוכחת הטענה כי סיבוכיות כל אלגוריתם השוואות למיון היא $\Omega(n \log n)$.

משפט זה מאפשר לנו הוכחה של חסמים תחתונים רבים בשיטת הרדוקציה. בשיטה זו, אפשר להוכיח כי עבור בעיה חישובית P, לא קיים אלגוריתם **מבוסס השוואות** לפתרון P בסיבוכיות קטנה ממש מ $n \log n$.

בשלב זה, נדגים את שיטת הרדוקציה על ידי הוכחת חסם תחתון על בעיה אופינית:

<u>**דוגמה: בעיית המערך הקמור**</u>

<u>**מערך קמור**</u> הוא מערך A שמספר איבריו זוגי והם מקיימים:

$$A[i] < A[i+1] \quad \text{if} \quad 1 \leq i \leq n/2$$

$$A[i] > A[i+1] \quad \text{if} \quad n/2+1 \leq i \leq n$$

תהי P הבעיה החישובית הבאה:

קלט: מערך A שאיבריו ניתנים להשוואה.

פלט: תמורה של איברי המערך A המהווה מערך קמור.

<u>**משפט**</u>

לא קיים אלגוריתם <u>**מבוסס השוואות**</u> לפתרון הבעיה P שהסיבוכיות שלו נמוכה מ-$\Theta(n \log n)$.

הוכחת החסם התחתון בשיטת הרדוקציה

נניח בשלילה כי קיים אלגוריתם Al_n המקבל כקלט מערך A ובו n איברים בני השוואה ומארגנם למערך קמור, וכי סיבוכיות האלגוריתם Al_n, $f(n)$ מקיימת $f(n) < n\log n$.

להלן נציג את אלגוריתם *Stira* אשר פותר את בעיית המערך הקמור תוך שימוש באלגוריתם Al_n כשגרת עזר.

אלגוריתם *Stira* - הקוד

```
Stira(A)
1. A ← Al_n(A)
2. for i ← n/2+1 to n
      B[i] ← A[n−i+1]
3.  merge(B[1..n/2], B[n/2+1.
4.  out(B)
```

איור 4.4: אלגוריתם *Stira*

אלגוריתם *Stira* - תיאור

מהלך האלגוריתם הוא כדלהלן:

בתחילה, האלגוריתם מפעיל את Al_n והופך את מערך הקלט A למערך קמור.

לאחר מכן, המערך הקמור מוסב למערך ממוין על ידי **מיזוג** שני החצאים של המערך A למערך ממוין.

המשך הוכחת החסם התחתון

נכונות *Stira* נובעת מיידית מתכונות Al_n ומנכונות *Merge*.

קל לראות כי *Stira* הוא אלגוריתם מונחה השוואות.

סיבוכיות שלב 1 היא $f(n)$ וסיבוכיות שאר השלבים היא לינארית.

מכאן נובע כי סיבוכיות האלגוריתם $Stira(A)$ היא $f(n) < n\log n$.

כעת נשים לה לכך ש *Stira* הוא אלגוריתם מיון מבוסס השוואות שהסיבוכיות שלו קטנה ממש

מ-$\Theta(n\log n)$, וזאת כמובן **סתירה** **לחסם התחתון שהוכחנו** בחלקה הראשון של ההרצאה.

מש"ל

סיבוכיות בעיית המערך הקמור

בטרם נציג את שיטת הרדוקציה באופן כללי, נחסום את סיבוכיות בעיית המערך הקמור, על ידי הצגת אלגוריתם מבוסס השוואות לפתרון הבעיה בסיבוכיות $n\log n$.

משפט

סיבוכיות הבעיה P, עבור אלגוריתמים מבוססי השוואות,היא $\Theta(n\log n)$.

הוכחה

כדי להוכיח את המשפט עלינו להציג אלגוריתם מבוסס השוואות הפותר את הבעיה בסיבוכיות $\Theta(n\log n)$.

בהקשר זה, נכנה לעתים אלגוריתם כזה בכינוי **חסם עליון**, וזאת בניגוד להוכחת **החסם התחתון**.

הוכחת החסם העליון: האלגוריתם

```
1. A ← MergeSort(A)
2. for i ← 1 to n/2
      B[i] ← A[i]
3. for i ← 1 to n/2
      B[n/2+i] ← A[n−i+1]
4. out(B)
```

קל לראות כי מערך הפלט B הוא קמור וכי סיבוכיות האלגוריתם המוצג היא $\Theta(n\log n)$.

שיטת הרדוקציה

כעת נציג את שיטת הרדוקציה באופן כללי. להלן פירוט שלבי השיטה :

1. נניח כי ברצוננו להוכיח כי הסיבוכיות של כל אלגוריתם מבוסס השוואות לפתרון בעיה חישובית P היא $\Omega(n\log n)$.
2. נניח בשלילה כי קיים אלגוריתם מבוסס השוואות, Al_p, לפתרון הבעיה P בסיבוכיות $f(n)$ המקיימת, $f(n)<n\log n$.

שיטת הרדוקציה (המשך)

3. נראה כי אפשר להשתמש באלגוריתם Al_p, כדי לקבל אלגוריתם מיון מבוסס השוואות שהסיבוכיות שלו היא $\Omega(n\log n)$ סתירה לחסם התחתון שהוכחנו.

הדרך לקבלת אלגוריתם המיון

אלגוריתם המיון שבו משתמשים בשיטת הרדוקציה מתקבל בדרך הבאה :

1. הסב את הקלט I_s לקלט לבעיה P, I_p.
(הערה : במקרים מסוימים, שלב זה מיותר)
2. הפעל את אלגוריתם Al_p על הקלט I_p וקבל את הפלט O_p.
3. הסב את הפלט O_p לפלט $O_s=sort(I_s)$.

מסקנה

אם שלושת השלבים הללו מתבצעים באלגוריתמים מבוססי השוואות שהסיבוכיות שלהם קטנה ממש מ $n\log n$ - קיבלנו אלגוריתם מיון מבוסס השוואות בסיבוכיות $f(n)<n\log n$ בסתירה למשפט שהוכחנו.

מכאן אפשר להסיק שהנחתנו הראשונה בדבר קיומו של Al_n אינה נכונה.

מש"ל

שיטת הרדוקציה (המשך)

לאור העובדה שאנו משתמשים באלגוריתם לפתרון P כדי להציג פתרון לבעיית המיון, אנו אומרים כי הבעיה P **ניתנת לרדוקציה** לבעיית המיון.

סיכום

בהרצאה זו, עסקנו בסיבוכיות של בעיות חישוביות. פתחנו בהגדרת מושג **סיבוכיות של בעיה חישובית**. הסברנו כי, בניגוד להוכחת סיבוכיות של אלגוריתם, הוכחת סיבוכיות של בעיה כרוכה בהוכחת **חסם תחתון** לסיבוכיות **כל אלגוריתם אפשרי לפתרון הבעיה.**

לאחר מכן, הוכחנו חסמים פשוטים לסיבוכיות של בעיות **לינאריות.**

עיקר ההרצאה הוקדש לניתוח של סיבוכיות בעית **מיון מבוסס השוואות.** במסגרת זו הוכחנו כי סיבוכיות בעיה זו היא $\Theta(n\log n)$. לסיום ההרצאה

פיתחנו את שיטת הרדוקציה להוכחת חסמים תחתונים לבעיות אחרות. הדגמנו את שיטת הרדוקציה בעזרת הוכחה כי סיבוכיות בעית חישוב וקטור קמור היא $\Theta(n\log n)$.

נספח: עצים בינאריים

נפתח בחזרה על כמה מושגים הקשורים לעצים בינאריים כלליים :

צומת - רשומה המכילה נתונים ומחוונים אל צמתים אחרים. כל מחוון לצומת אחרת נקרא גם **קשת.**

שימו לב: יתכנו צמתים ללא מידע או ללא מחוונים.

עץ בינארי - מבנה נתונים המכיל צמתים וקשתות. **שורש העץ** r הוא צומת אליו לא נכנסת אף קשת. לכל צומת בעץ הבינארי, למעט השורש יש **אב** יחיד. לכל צומת יש 0-2 **בנים.** אחד הבנים של הצומת יקרא בן **שמאלי** והשני בן **ימני.** אם לצומת יש רק בן אחד, הוא יכול להיות שמאלי או ימני.

מושגים בעצים בינאריים (המשך)

רמה של צומת היא **מספר הקשתות** במסלול (היחיד) מן השורש אל הצומת.
עלה - צומת שאין לו בנים.
גובה של עץ - רמה מכסימלית של עלה.
עץ בינארי מלא - עץ בינארי בו לכל צומת שאינו עלה יש **שני בנים.**
עץ בינארי מאוזן - עץ בינארי מלא בו כל העלים נמצאים באותה רמה.
עץ בינארי כמעט מאוזן - עץ בינארי בו כל הרמות מלאות למעט הרמה האחרונה. הרמה האחרונה מלאה משמאל עד לנקודה כלשהי.
כל עץ בינארי מתחלק ל:
1. תת עץ שמאלי (יתכן שהוא ריק).
2. תת עץ ימני (יתכן שהוא ריק).

אלגוריתמים א' הרצאה 5 - פרופ' עמוס ישראלי

תכונות עצים בינאריים

טענה 1

יהי T_1 עץ בינארי מאוזן בגובה h. אזי מתקיים:
1. מספר הצמתים ברמה l, $l = 0,1,\ldots,h$ הוא 2^l.
2. מספר הצמתים ב T_1 הוא $2^{h+1}-1$.

הוכחת טענה 1

תחילה נוכיח את 1 באינדוקציה על l.
בסיס: $l = 0$ - ברמה 0 יש צומת יחיד ואכן מתקיים.
צעד: נניח כי ברמה l יש 2^l צמתים. העץ T_1 מאוזן, ולכן, לכל צומת ברמה l

אלגוריתמים א' הרצאה 5 - פרופ' עמוס ישראלי

יש בדיוק שני בנים, ברמה $l+1$. מכאן נובע כי מספר הצמתים ברמה $l+1$ של T_1 הוא $2 \cdot 2^l = 2^{l+1}$.
כדי להוכיח את סעיף 2 נשים לב כי אם n הוא מספר צמתי T_1 אזי מתקיים:

$$n = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1$$

אלגוריתמים א' הרצאה 5 - פרופ' עמוס ישראלי

טענה 2

יהי T_2 עץ בינארי כמעט מאוזן שגובהו h. נסמן את מספר איברי T_2 ב - n. אזי n מקיים:

$$2^h \le n \le 2^{h+1} - 1$$

מכאן נובע כי:

$$h \le \log n < h+1$$

או

$$h = \lfloor \log n \rfloor$$

הוכחה

מיידית.

אלגוריתמים א' הרצאה 5 - פרופ' עמוס ישראלי

טענה 3

יהי T עץ בינארי כלשהו עם n עלים. גבהו של T הוא $\Omega(\log n)$.

הוכחה

אם T הוא כמעט מאוזן, נכונות הטענה נובעת באופן מיידי מטענה 2.
נניח אם כן כי T אינו כמעט מאוזן ונראה כי קיים עץ בינארי כמעט מאוזן עם n עלים, T_{ab}, אשר גבהו קטן או שווה מגבהו של העץ T.

אלגוריתמים א' הרצאה 5 - פרופ' עמוס ישראלי

הוכחה (המשך)

העץ T_{ab} יתקבל בשני שלבים כדלהלן:
שלב 1: מציאת עץ בינארי מלא, T_c, עם n עלים, אשר גבהו $h(T_c)$ מקיים:

$$h(T) \ge h(T_c)$$

שלב 2: מציאת עץ בינארי כמעט מאוזן, עם n עלים, T_{ab}, אשר גבהו, $h(T_{ab})$ מקיים:

$$h(T) \ge h(T_c) \ge h(T_{ab})$$

מאחר ש T_{ab} הוא עץ בינארי כמעט מאוזן עם n עלים, גבהו הוא $\Theta(\log n)$ והמשפט נובע מאי השוויונים שצוינו.
נותר לנו להראות כיצד נבצע את שלבים 1 ו 2.

אלגוריתמים א' הרצאה 5 - פרופ' עמוס ישראלי

ביצוע שלב 1

העץ T_c ימצא כך: אם T מלא, אז $T_c = T$. אם T אינו מלא, נתבונן בסדרת העצים הבינאריים

$$T = T_0, T_1, \ldots, T_k = T_c$$

כאשר T_0 הוא העץ הנתון T, T_k הוא העץ המלא T_c, ולכל i, $1 \le i \le k$, T_i מתקבל מ T_{i-1} על ידי מחיקת צומת v שדרגתו 1, ביחד עם הקשת (v,u) היוצאת ממנו. הצומת u, בנו של v, "יתלה" על אביו של v.

אלגוריתמים א' הרצאה 5 - פרופ' עמוס ישראלי

ביצוע שלב 1 (המשך)

בדרך זו, מספר העלים של T_i שווה למספר העלים של T_{i-1} וגבהו של T_i קטן או שווה מגבהו של T_{i-1}. כל עוד T_i אינו מלא, קיים בו צומת שדרגת היציאה שלו שווה ל 1 והוא ניתן להסרה ולכן התהליך יעצר רק כאשר נגיע לעץ בינארי מלא.

<u>ביצוע שלב 2</u>

העץ T_{ab} ימצא כך: אם T_c הוא כמעט מאוזן, הטענה נובעת מייד. אם T_c אינו כמעט מאוזן, נתבונן בסדרת העצים הבינאריים

$$T_c = T_0, T_1, \ldots, T_m = T_c$$

כאשר T_0 הוא העץ T_c ו T_m הוא עץ בינארי כמעט מאוזן T_{ab}.

לכל i, $1 \le i \le m$, T_i מתקבל מ T_{i-1} על ידי אחד משני השינויים הבאים:

1. העברת זוג עלים מצומת שמרחקו מן השורש הוא l (לכל $l > 1$), להיות בניו של עלה אשר מרחקו מן השורש קטן מ $l - 1$.

<u>ביצוע שלב 2(המשך)</u>

2. העברת עלה מצומת שמרחקו מן השורש l אל צומת שמאלי יותר שמרחקו מן השורש הוא l.

כל אחד משני שינויים אלה משמר את מספר העלים בעץ ואינו מגדיל את גובה העץ. כל עוד T_i אינו כמעט מאוזן, אפשר לבצע לפחות אחד משני השינויים המתוארים ולכן, התהליך המתואר יעצור רק כאשר נגיע לעץ כמעט מאוזן.

<u>הוכחה (המשך)</u>

מאחר ש T_{ab} הוא כמעט מאוזן ויש לו n עלים נובע מטענה 2 בנספח להרצאה זו, כי גבהו של T הוא $\Theta(\log n)$. מאחר שגבהו של T גדול או שווה מגבהו של T_{ab}, נובע כי גבהו של T הוא $\Omega(\log n)$.

מש"ל

הרצאה 6: מיון שאינו מבוסס השוואות

בהרצאה זו, נדון באלגוריתמים שאינם מבוססי השוואות. כזכור, הוכחנו בהרצאה הקודמת כי הסיבוכיות של בעית ה**מיון מבוסס ההשוואות** היא $\Theta(n\log n)$.

בהרצאה זו נדון בשאלה הבאה: האם ניתן ל"שבור" את החסם התחתון $\Omega(n\log n)$ לסיבוכיות הזמן של מיון?

תשובה: עבור מקרים מיוחדים בהם הערכים הממוינים **חסומים** אפשר למיין בזמן $\Theta(n)$.

דוגמא

מיון של **מערך בינארי** (מערך שאיבריו הם אך ורק 0 ו-1).

1	0	0	1	1	0	1	1	0	0	0	1

איור 6.1: מערך בינארי

תיאור האלגוריתם

1. ספור כמה 0-ים מופיעים במערך.
2. ספור כמה 1-ים מופיעים במערך.
3. בנה את הפלט.

מיון מניה - תאור האלגוריתם

יהי m מספר טבעי (חיובי ושלם) ונניח כי כל איברי המערך A מקימים:

$$1\le A[i]\le m$$

מיון מניה משתמש **במערך מונים** שיקרא $Counter$ ובו m איברים.

האלגוריתם מתבצע ב 3 שלבים:

1. איתחול מונים.
2. ביצוע סטטיסטיקה מלאה של הקלט לפי ערכי תחום ההגדרה.
3. בניית הפלט.

מיון מניה- הקוד

```
CountingSort(A)
/* איתחול מונים */
for i ← 1 to m  Counter[i] ← 0
/* ביצוע סטטיסטיקה על הקלט */
for j ← 1 to n
  elt ← A[j]
  Counter[elt]++
End for
j ← 1
/* בניית הפלט */
for i ← 1 to m
  while Counter[i] ≠ 0
     outV[j] ← i; j++
                    Counter[i]--
  end while
end for
out(outV)
```

איור 6.2: מיון מניה

שימו לב:

מיון מניה **אינו מיון מבוסס השוואות.**

הסיבה: אם נבטא את האלגוריתם רק על ידי הוראות **if** נקבל השוואה של כל איבר קלט לכמה איברים בתחום $1,\dots,m$ ובמיון מבוסס השוואות מותרות רק השוואות מן הצורה $A[i]<A[j]$. בנוסף, בקוד שלפנינו החריגה מתבטאת גם בהוראת ה-**while**.

משפט הנכונות עבור מיון מניה זהה למשפט הנכונות לכל אלגוריתם מיון אחר. להלן המשפט:

משפט 6.1

אלגוריתם מיון מניה (Counting Sort) המופעל על מערך קלט A, שערכיו הם מספרים טבעיים בין 1 ל-m, מחשב תמורה ממוינת של המערך A.

הוכחה

נכונות האלגוריתם מוכחת בעזרת שלוש טענות פשוטות שההוכחות שלהן מיידיות.

טענה 6.2

לכל i, $1\le i\le m$, נסמן ב-n_i את מספר המופעים של הערך i במערך הקלט A. לאחר ביצוע שתי הלולאות הראשונות מתקיים:

1. $Counetr[i]=n_i$
2. $n=\sum_{i=1}^{m}counter[i]=\sum_{i=1}^{m}n_i$

הוכחה

מיידית.

טענה 6.3

לכל i, $1\le i\le m$, נסמן ב-j_i את ערך המשתנה j ברגע בו המעבר ה-i בלולאת ה-**while** מסתיים. לצורך זה נניח כי $j_0=1$ וגם ש-$n_0=1$. אזי באותו רגע הטענה הבאה מתקיימת: $out[j_i-n_i\dots j_i-1]=[i,i,\dots,i]$.

הוכחה

מיידית.

טענה 6.4

המערך out הוא תמורה ממוינת של מערך הקלט A.

הוכחה

נוכיח באינדוקציה על מספר המעברים בלולאה החיצונית השלישית, כי לאחר המעבר ה-i, $1 \le i \le m$, בנקודת הסיום של הלולאה הזו מתקיים:

1. $j = 1 + \sum_{k=1}^{i} n_i$
2. $j-1$ האיברים הראשונים במערך $outV$ הם תמורה ממוינת של $j-1$ האיברים המינימליים במערך הקלט.
3. ערכם של i האיברים הראשונים במערך $Counter$ הוא 0.

נוכיח את הטענה באינדוקציה על מספר המעברים בלולאה החיצונית:

המשך ההוכחה

בסיס ($i = 0$) **ברגע הכניסה ללולאה:**

1-2. הטענה נכונה באופן ריק ($j = 1$).

3. הטענה נכונה באופן ריק ($i = 0$).

צעד האינדוקציה:

נניח נכונות טענות 1-3 עבור i ונוכיח עבור $i+1$

הוכחה

מיידית ממעקב על ביצוע הלולאה הפנימית עבור $Counter[i+1]$.

נכונות האלגוריתם

נכונות האלגוריתם נובעת מיידית מנכונות טענה 6.4 (סעיף 2) עבור $i = m$. **מש"ל**

ניתוח סיבוכיות האלגוריתם

להלן נוכיח כי סיבוכיות האלגוריתם היא $\Theta(n+m)$.

ברור שסיבוכיות החלק הראשון והחלק השני הן $\Theta(n)$ ו- $\Theta(m)$ בהתאמה.

ניתוח סיבוכיות החלק השלישי מסובך קצת יותר:

הלולאה החיצונית מתבצעת m פעמים, אך מהי סיבוכיות הלולאה הפנימית?

לכל i, $1 \le i \le n$, מתקיים $0 \le counter[i] \le n$. מכאן נובע כי בכל מעבר בלולאה החיצונית, הלולאה הפנימית תתבצע **לכל היותר** n פעמים.

מכאן נובע כי סיבוכיות האלגוריתם חסומה מלמעלה על ידי $O(n \cdot m)$.

כדי להוכיח כי הסיבוכיות היא בדיוק $\Theta(n+m)$, נשים לב לעובדה כי במעבר ה-i בלולאה החיצונית הוראת ה-**while** מתבצעת **בדיוק** $Counter[i]+1$ פעמים. **נמכאן נובע כי** מספר הצעדים הכולל שהוראת ה-**while** מתבצעת הוא

$$\sum_{i=1}^{m}(counter[i]+1) =$$

$$= \sum_{i=1}^{m} counter[i] + m = n + m$$

מש"ל

מתי כדאי להשתמש במיון מנייה?

כל עוד $m < \Theta(n \log n)$ עדיף להשתמש במיון מניה.

שימו-לב:

אפשר לענות על שאלה זו אך ורק על ידי "השאפת" גודל הקלט לאינסוף.

אם $m = \Theta(n \log n)$ עלינו לשקול גם את גודל הזכרון הנדרש.

מוטיבציה למיון דליים (Bucketsort)

אלגוריתם מיון ספירה *CountingSort* יכול לשמש כאשר ממיינים מערך ובו מספרים טבעיים בעלי ערך חסום. נניח כעת כי נתון מערך קלט שאיבריו הם **רשומות עם כמה שדות** (לדוגמה רשומות המתארות סטודנטים).

ברצוננו למיין את הרשומות הללו לפי **מפתח** שהוא אחד השדות. אם המפתח הוא **מספר טבעי** שערכיו **חסומים** (למשל גיל בשנים, או ממוצע ציונים מעוגל לערך שלם), עולה הרעיון להשתמש במיון לינארי.

מוטיבציה למיון דליים (המשך)

לרוע המזל, אם רוצים לשמור על מלוא המידע הכלול בכל רשומה אין אפשרות להשתמש במיון מניה. אלגוריתם **מיון דליים** (**Bucket Sort**) המתואר להלן משתמש ברעיון זהה, בדרך שונה במקצת.

מיון דליים (Bucketsort)

נניח כי נתון מערך של רשומות ורוצים למיין את הרשומות לפי אחד השדות. שדה זה מכונה **מפתח**.

לדוגמא: אפשר למיין רשומות המייצגות סטודנטים לפי המפתח ממוצע מצטבר.

הרעיון: נשתמש במערך שבו יהיו m מבני נתונים המכונים **דליים** (**buckets**). נעבור על הקלט ו"נשליך" כל רשומה עם מפתח שערכו i אל הדלי ה-i. לאחר מעבר זה, נקבל את הפלט על ידי שירשור את האיברים בכל הדליים שאינם ריקים.

מיון דליים בעזרת רשימות מקושרות

באלגוריתם זה, כל "דלי" ממומש בעזרת **רשימה מקושרת**. כל הרשומות בעלות מפתחות שווים, מאוחסנות ברשימה מקושרת אחת.

באמצעות רעיון זה נקבל אלגוריתם מיון אשר משתמש אך ורק בהזזת מצביעים (Pointers). סיבוכיות הזמן של האלגוריתם היא $\Theta(n+m)$ וסיבוכיות המקום גם היא $\Theta(n+m)$.

מיון דליים - מבנה הנתונים

נניח כי מערך הקלט, A, מכיל מצביעים אל רשומות וכי אנו רוצים למיין רשומות אלו לפי השדה *key*. נסמן את ערך השדה *key* ברשומה ש-$A[i]$ מצביע אליה על ידי $key[i]$. (אפשר גם לסמנו על ידי $A[i].key$)

האלגוריתם משתמש בשני מערכים של מצביעים, H (Heads) ו T (Tails) שבכל אחד מהם יש m איברים.

המערכים H ו-T מאחסנים m ראשי רשימות, כלומר מצביעים אל האיבר הראשון ברשימה, ו-m סופי רשימות, בהתאמה.

מיון דליים - מהלך האלגוריתם

שלב 1: איתחול מונים

לכל i, $1 \le i \le m$, האלגוריתם מאתחל את המשתנים $H[i]$ ו-$T[i]$ ל-*null*.

שלב 2: ביצוע סטטיסטיקה על הקלט

האלגוריתם עובר על הקלט ומכניס כל איבר שערך השדה *key* שלו הוא j, $1 \le j \le m$, אל קצה הרשימה שראשה מאוחסן ב $H[j]$.

שימו לב: כדי לבצע הכנסה זו בזמן שאינו תלוי באורך הרשימה, משתמשים במצביע $T[j]$ המצביע אל קצה הרשימה הזו.

שלב 3: בניית הפלט

לאחר ביצוע שלב הסטטיסטיקה, **וקטור הפלט** *outV* מתקבל על ידי **שירשור** הרשימות החל ב-$H[1]$ וכלה ב-$H[m]$.

כל פעולות האלגוריתם מתבצעות **אך ורק** על ידי **פעולות על מצביעים.**

מיון דליים - הקוד

```
BucketSort(A)
/* אתחול רשימות */
for j ← 1 to m
    init H[j] and T[j] to empty
/* ביצוע סטטיסטיקה על הקלט */
for i ← 1 to n
    insert A[i] to list H[key[i]]
                using T[key[i]]
/* בניית הפלט */
j ← 1
/* בניית הפלט */
for i ← 1 to m
  while list H[i] not empty
     out[j] ← remove(H[i])
      j++
  end while
```

איור 6.3: מיון דליים

הערות

1. אלגוריתם מיון דליים משתמש ברעיון זהה לרעיון המונח ביסוד מיון מניה.
2. בקוד מיון דליים, שמרנו את ההערות שהשתמשנו בהן בקוד של מיון מניה, וזאת לצורך הבלטת הדמיון בין שני האלגוריתמים.
3. ההבדל בקוד נובע מן ההבדל באיברי מערך הקלט ומן הרצון לשמור על כל המידע הכלול באיברים אלה.

תכונות מיון דליים

טענה 6.5

לאחר ביצוע שתי הלולאות הראשונות, הרשימה $H[i]$ מכילה את כל הרשומות המקיימות $key[j]=i$.

הוכחה

מיידית.

טענה 6.6

המערך *outV* מכיל רשימה מקושרת של כל רשומות הקלט. המערך *outV* ממוין לפי סדר עולה של השדות *key*.

הוכחה

מיידית.

סיבוכיות

$\Theta(n+m)$

יציבות של אלגוריתם מיון

כאשר ממינים רשומות, לפי מפתח, יש משמעות לשאלה מהו הסדר הפנימי בין כל הרשומות בעלות מפתח שווה.

לדוגמא: מה יהיה הסדר בפלט של כל הסטודנטים שלהם ממוצע ציונים 75.

בהקשר זה נגדיר: אלגוריתם מיון הוא **אלגוריתם יציב** אם הסדר **בפלט** של איברים בעלי מפתחות שווים, **זהה** לסדרם של אותם איברים **בקלט.**

שימו לב

כאשר האיברים הממוינים הם **סקלרים** אין משמעות ליציבות המיון. היציבות מתבטאה כאשר האיברים הממוינים הם מבנים מורכבים יותר.

יציבות אלגוריתמי מיון (המשך)

כדי לבין זאת טוב יותר נתבונן בבעיה הבאה:

נניח כי בידינו קובץ רשומות המיצגות אנשים. בכל קובץ יש שדה המציין שם ושדה נוסף המציין גיל והקובץ ממוין לפי השם.

דוגמא

אברהם	20
דוד	20
חיים	30
יצחק	20
משה	30
פנחס	18

דוגמא (המשך)

נניח וכי אנו רוצים למיין את הקובץ לפי גיל כאשר עבור כל גיל, הקובץ יהיה ממוין לפי השם.

הפלט יהיה:

פנחס	18
אברהם	20
דוד	20
יצחק	20
חיים	30
משה	30

הסבר

אנו מחפשים אלגוריתם אשר ימיין את הקובץ לפי הגיל, כאשר עבור כל קבוצת גיל הסדר המקורי בקובץ הקלט ישמר.

אם נמיין את הקובץ, לפי הגיל, במיון לא יציב, יתכן שבפלט אברהם יופיע אחרי יצחק ודוד.

שימוש במיון יציב, ישמור את הסדר האלפביתי של השמות הפרטיים של אנשים אם אותו שם משפחה ויבטיח את קבלת הפלט הרצוי ללא מאמץ נוסף.

יציבות מיון דליים

יציבות מיון דליים תלויה במימוש הדליים. במימוש שתיארנו, כל דלי ממומש על ידי **רשימה מקושרת** בה מאוחסנים כל איברי הקלט שלהם מפתחות זהים. כדי להשיג יציבות, סדר האיברים המאוחסנים בכל צריך להיות זהה לסדר בו האיברים הללו מופיעים במערך הקלט. בדרך זו, הסדר יישמר גם בעת בניית מערך הפלט. כדי להבטיח את שמירת סדר ההופעה, יש להכניס כל איבר לקצה הרשימה המקושרת תוך שימוש במצביעים $T[i]$.

לאחר שהגדרנו את מושג היציבות נקבל:

משפט 6.7

אלגוריתם מיון דליים ממיין רשומות באופן **יציב** בסיבוכיות $\Theta(m+n)$.

מיון שארית (Radix Sort)

האלגוריתם האחרון בו נדון בהרצאה זו הוא **מיון שארית (Radix Sort)**.

אלגוריתם זה משמש כאשר נתון מערך (גדול מאוד) שאיבריו הם רשומות בנות בני k תווים (למשל ספרות או אותיות). הגישה האינטואיטיבית (והשגויה) היא אחת האפשרויות היא למיין את הקבצים, תו אחר תו, לפי רמת המשמעותיות של התווים.

דוגמא

נניח שהקלט הוא מערך מספרים בני k ספרות, כגון מערך של מספרי זהות.

האלגוריתם האינטואיטיבי

1. מיין קובץ מספרים לפי הספרה הראשונה.
2. בשלב השני יש למיין (כל אחד מ10 קבצי הביניים שהתקבלו כפלט מן השלב הראשון) לפי הספרה השניה.
3. בשלב ה i, $1 \le i \le k$, יש למיין כל אחד מ 10^{i-1} קבצי הביניים לפי הספרה ה i.

הבעייתיות באלגוריתם שהוצע

כל שלב במיון זה, אפשר לבצע על ידי מיון דליים.

הבעיה המתגלה במיון זה היא:

במהלך המיון, עלינו לזהות את גבולות הקבצים, באופן דינמי, או לשמור על כמות של $m \cdot k$ מצביעים שיאורגנו בהיררכיה מסובכת למדי.

מצב זה, מסבך את התיכנות ומגביר מאוד את כמות הבגים הצפויה.

להלן נציג את האלגוריתם הנקרא **מיון שארית** אשר מתגבר על בעיה זו.

מיון שארית - הרעיון

במיון לפי הסדר המילוני (לכסיקוגרפי), המיקום היחסי של כל שני איברים a ו b (כלומר מי מהם מופיע ראשון בקובץ הפלט) נקבע לפי האות (ספרה) המשמעותית ביותר שבה האיברים שונים זה מזה.

תכונה זו מנוצלת כך :

נשתמש בגרסה **יציבה** של מיון דליים, נמיין את הקובץ כולו k פעמים, בכל פעם אך ורק לפי הספרות בעמודה מסוימת ובסדר **הפוך** לסדר המשמעותיות, כלומר החל בספרת היחידות (הראשונה מימין) וכלה בספרה המשמעותית ביותר, האחרונה מימין (הראשונה משמאל).

מיון שארית - הרעיון (המשך)

האלגוריתם יערך ב-k איטרציות. במעבר הראשון ימוינו כל הרשומות לפי הספרה הכי פחות משמעותית, כלומר הראשונה מימין. במעבר השני ימוינו **שוב** כל הרשומות לפי הספרה השניה בסדר המשמעותיות (ספרת העשרות), וכך לכל i, $3 \le i \le k$, במעבר ה-i ימוינו **שוב** כל הרשומות לפי הספרה ה-i בסדר המשמעותיות.

מיון שארית - הרעיון (המשך)

בדרך זו, היחס בין כל שני איברים a ו-b נקבע אך ורק לפי הספרה המשמעותית ביותר עבורה a ו-b שונים זה מזה, ללא תלות במיקומם לפני איטרציה זו.

כל הפעולות האחרות, לא משפיעות כלל על המצב היחסי a ו-b.

תאור האלגוריתם

עבור מפתחות בני k תווים (k קבוע), $b_k, b_{k-1}, \ldots, b_1$ האלגוריתם יעבוד ב - k איטרציות. באיטרציה ה i, $1 \le i \le k$, הקובץ כולו ימוין במיון דליים, לפי הספרה ה - i.

נשתמש בגרסה יציבה של מיון דליים ובכך נבטיח כי הסדר היחסי הנכון בין המפתחות הממוינים ישמר.

דוגמא - הקלט

2534				
1769				
4932				
5136				
6492				
1186				
7293				
8116				
3229				
2478				

דוגמא - לאחר האיטרציה הראשונה

2534	4932			
1769	6492			
4932	7293			
5136	2534			
6492	5136			
1186	1186			
7293	8116			
8116	2478			
3229	1769			
2478	3229			

דוגמא - לאחר האיטרציה השניה

2534	4932	8116		
1769	6492	3229		
4932	7293	4932		
5136	2534	2534		
6492	5136	5136		
1186	1186	1769		
7293	8116	2478		
8116	2478	1186		
3229	1769	6492		
2478	3229	7293		

דוגמא - לאחר האיטרציה השלישית

2534	4932	8116	8116	
1769	6492	3229	5136	
4932	7293	4932	1186	
5136	2534	2534	3229	
6492	5136	5136	7293	
1186	1186	1769	2478	
7293	8116	2478	6492	
8116	2478	1186	2534	
3229	1769	6492	1769	
2478	3229	7293	4932	

דוגמא - לאחר האיטרציה הרביעית

2534	4932	8116	8116	1186
1769	6492	3229	5136	1769
4932	7293	4932	1186	2478
5136	2534	5136	3229	2534
6492	5136	2534	7293	3229
1186	1186	1769	2478	4932
7293	8116	2478	6492	5136
8116	2478	1186	2534	6492
3229	1769	6492	1769	7293
2478	3229	7293	4932	8116

הוכחת נכונות

נכונות האלגוריתם נובעת מן הטענה הבאה:

טענה 6.8:

לאחר i איטרציות $0 \le i \le k$ איברי המערך ממוינים לפי i הספרות **הפחות משמעותיות** שלהם.

הוכחה:

נכונות הטענה נובעת מיידית מיציבות המיון.

סיכום

בהרצאה זו, דנו במיונים אשר אינם מבוססי השוואות.

בתחילת ההרצאה הצגנו שתי גרסאות של מיון דליים והוכחנו את נכונותן.

לאחר מכן, הגדרנו מהו **מיון יציב** והסברנו מה התועלת ביציבות.

סיימנו את ההרצאה בהצגת **מיון שארית** (Radix Sort) המשתמש **במיון דליים יציב** כשגרה.

סיכום נושא המיון

בארבע ההרצאות האחרונות עסקנו בבעית המיון. הנושאים שלמדנו הם:

1. אלגוריתמים פשוטים למיון בסיבוכיות ריבועית: מיון בחירה (Selection Sort) ומיון בועות (Bubble Sort).
2. מיון מיזוג (Merge Sort) בסיבוכיות $\Theta(n \log n)$.
3. מיון מהיר (Quick Sort) בסיבוכיות $\Theta(n^2)$ ובסיבוכיות ממוצעת $\Theta(n \log n)$.

סיכום נושא המיון (המשך)

4. חסם תחתון (חסם תורת האינפורמציה) בגובה $\Omega(n \log n)$ על סיבוכיות אלגוריתמי מיון מבוססי השוואות.
5. אלגורית מיון לינאריים (שאינם מבוססי השוואות).

הרצאה 7: תכנות דינמי

(Dynamic Programming)

בהרצאה זו נלמד פרדיגמה חדשה לבניית אלגוריתמים.

הפרדיגמה קרויה **תכנות דינמי** והיא משמשת לפתרון בעיות שאינן ניתנות לפתרון יעיל בשיטת הפרד ופתור.

בתחילת ההרצאה נציג בעיה פשוטה ונפתור אותה תוך שימוש בתכנות דינמי. לאחר מכן נציג את עיקרי השיטה באופן כללי.

בעיה 7.1

נתון מערך A ובו n איברים חיוביים. מחפשים תת מערך (לא רציף) שסכום איבריו מרבי כאשר מכל שני איברים סמוכים – מותר לקחת לכל היותר אחד.

דוגמאות

נתבונן בדוגמאות הבאות:

3	**12**	6

5	4	**9**

הפתרון: 14 12

13	3	5	**9**

12	5	6	**17**	9

הפתרון: 39 22

מהדוגמאות אפשר להסיק מספר האיברים הרצופים שאינם כלולים בפתרון הוא 2. המסקנה הזו מתבטאת בלמה הבאה:

למה 7.2

לכל קלט חוקי לבעיה 7.1 מתקיים:

1. הפתרון האופטימלי מכיל לפחות איבר אחד מכל שלושית איברים רצופים.
2. הפתרון האופטימלי מכיל תמיד את $A[n-1]$ או את $A[n]$, אך כמובן לא את שניהם.

הוכחה

1. נניח בשלילה כי קיים אינדקס i המקיים: הפתרון האופטימלי לא מכיל את $A[i]$, $A[i+1]$, וגם את $A[i+2]$. מאחר שנתון כי $A[i+1]>0$, אפשר לראות כי צרוף $A[i+1]$ לפתרון הנתון, מניב פתרון חוקי שסכום איבריו גדול יותר מסכום איברי הפתרון הקודם - סתירה.
2. באופן דומה אפשר להראות כי אם $A[n-2]$ לא נכלל בפתרון כלשהו אז $\max(A[n-1],A[n])$ נכלל בפתרון הזה ואם $A[n-1]$ לא נכלל בפתרון, אז $A[n]$ נכלל בו.

מש"ל

תיכון אלגוריתם רקורסיבי

כדי להגיע לאלגוריתם לפתרון הבעיה נציג תחילה אלגוריתם רקורסיבי לפתרון הבעיה:

נסמן ב-$B(A,i)$ את פתרון הבעיה עבור תת המערך $A[1...i]$ תחת האילוץ ש-$A[i]$ שייך לסכום זה.

דוגמא

עבור המערך $(3,12,6)$

$$B(A,1)=A[1]+A[3]=9$$

להלן נציע אלגוריתם רקורסיבי לחישוב $B(A,n)$. נכונות האלגוריתם נובעת מלמה 7.2.

אלגוריתם רקורסיבי לחישוב $B(A,i)$

```
B(A,i)
   if ((i=1)||(i=2)) return A[i]
return A[i]+B(A,i-2),B(A,i-3)
```

איור 7.1: אלגוריתם רקורסיבי

נכונות האלגוריתם נובעת מסעיף 1 בלמה 7.2.

פתרון בעיה 7.1 מתקבל מ:

```
Rec(A)
   Return Max(B(A,n),B(A,n-1))
```

איור 7.2: האלגוריתם הסופי

נכונות האלגוריתם נובעת מסעיף 2 בלמה 7.2.

סיבוכיות האלגוריתם הרקורסיבי

אם נתבונן בעץ הקריאות, נוכל לראות כי הוא מכיל עץ בינארי מאוזן שגובהו הוא $n/3$. מכאן נובע כי מספר הקריאות הרקורסיביות הוא **אקספוננציאלי**.

הגורם לסיבוכיות הוא מספר הפעמים הרב שאנו קוראים לאלגוריתם עם כל אחד מן הארגומנטים $1,2,...,n$.

כדי להשיג אלגוריתם **לינארי** נשתמש **טבלת מעקב (LookUp Table)** כמפורט בהמשך ההרצאה.

תיכון אלגוריתם בתכנות דינמי

נתבונן בקלט

5	2	3	8	4	6	9	1	8	4

נשתמש **במערך עזר** B באורך n באיבר ה- $B[i]$ נרשום את $B(A,i)$. כלומר פתרון בעיה 7.1 עבור המערך $A[1:i]$ תחת האילוץ: האיבר $A[i]$ **נכלל בסכום**.

הקלט:	5	2	3	8	4	6	9	1	8	4
מערך עזר:	5	2	8	13	12	19	22	20	30	26

חישוב איברי מערך העזר

ברור כי $B[1]=5$, $B[2]=2$, ו - $B[3]=5+3=8$.

כעת נשים לב כי

$B[4]=A[4]+\max(B[1],B[2])$

הסיבה לכך: לפי סעיף 1 בלמה 7.2 אחד מבין האיברים $A[1]$, $A[2]$, או $A[3]$ חייב להיות בתת המערך המרבי. נתון לנו כי $A[4]$ שייך לתת המערך ולכן לא יתכן כי $A[3]$ נמצא בו. מכאן נובע כי איבר אחד בדיוק מבין $A[1]$ ו $A[2]$ שייך לתת המערך המרבי. הנוסחה נובעת.

באופן דומה אפשר לראות כי לכל $5\le i\le n$ מתקיים

$B[i]=A[i]+\max(B([i-3],B[i-2])$

המשך תיכון האלגוריתם

מהי התשובה הסופית?

נחזור אל המערך שחישבנו

הקלט:	5	2	3	8	4	6	9	1	8	4
מערך עזר:	5	2	8	13	12	19	22	20	30	26

ברור כי איבר אחד מבין $A[n-1]$ ו-$A[n]$ משתייך לתת המערך המרבי. הפתרון מתקבל על ידי:

$final=\max\{B(n-1),B(n)\}$

ובדוגמה שלנו הפתרון הוא 30.

אלגוריתם לינארי לפתרון בעיה 7.1

הקוד המופיע באיור 7.3 מהווה אלגוריתם לינארי לפתרון בעיה 7.1:

```
MaxVec(A)
  B[1] ← A[1]; B[2] ← A[2];
  B[3] ← B[1] + A[3];
  for i ← 4 to n
      B[i] ← A[i] +
              + max(B[i-3], B[i-2])
  endfor
  return(max(B[n], B[n-1]))
```

איור 7.3: האלגוריתם הלינארי

משפט נכונות

האלגוריתם $MaxVec$ מחשב את סכום תת הוקטור המירבי בהתאם לדרישות.

הוכחה

צריך להוכיח באינדוקציה על i כי לאחר חישוב האיבר $B[i]$ ערכו שווה לסכום תת הוקטור $A[1:i]$ המרבי **אשר מכיל את** $A[i]$.

ההוכחה נובעת באופן מיידי מן הלמה, כפי שהסברנו את הדרך בה חישבנו את האיבר $B[4]$.

שימו לב: במקרה זה, השמורה שווה להגדרת מערך העזר.

הסיבוכיות

לינארית.

חישוב תת המערך המרבי

כעת נחשב את תת המערך המרבי עצמו ולא רק את סכום האיברים שלו. נרשום מחדש את מערך הקלט ואת מערך העזר ונוסיף להם מערך עזר נוסף: באיבר ה- i נרשום את האינדקס שהניב את המכסימום בעזרתו חשבנו את $B[i]$.

האינדקס:	1	2	3	4	5	6	7	8	9	10
הקלט:	5	2	3	8	4	6	9	1	8	4
מערך עזר:	5	2	8	13	12	19	22	20	30	26
המקור:	-	-	1	1	3	4	4	6	7	7

חישוב תת המערך המרבי(המשך)

אפשר לראות כי הסכום $B[9]=30$ התקבל על ידי $B[9]=B[7]+A[9]$ אם נמשיך בדרך זו נקבל:

$B[9]=B[7]+A[9]=B[4]+A[7]+A[9]$

ומאחר ש $B[4]=A[1]+A[4]$ נקבל

$B[9]=A[1]+A[4]+A[7]+A[9]$

סיכום הצעדים שביצענו

1. הגדרנו את מערך העזר: "סכום תת הוקטור המירבי עבור הוקטור $A[1:i]$ כאשר נתון ש-$A[i]$ שייך למערך המירבי".
2. חישבנו את איברי מערך העזר.
3. מצאנו את סכום הוקטור המרבי עבור המערך A.
4. חישבנו את האינדקסים של האיברים של תת המערך המרבי.
5. הוכחת הנכונות מתבססת על השמורה: "בסיום המעבר ה-i בלולאה, מתקיים:

$B[i]=B(A,i)$

תרגילים

1. השלימו את הוכחת הנכונות.
2. השלימו את קוד האלגוריתם כך שיחשב את האינדקסים של האיברים מהם מתקבל תת הוקטור המירבי.
3. הציעו אלגוריתם לפתרון הבעיה הבאה: נתון מערך A ובו n איברים חיוביים. מחפשים תת מערך (לא רציף) שסכום איבריו **מינימלי** כאשר מכל שני איברים סמוכים - חייבים לקחת לפחות אחד.

בעיה 7.2

נתון מערך A ובו n איברים חיוביים. מחפשים את הסכום המרבי אשר אפשר לקבל כאשר - מכל שני איברים סמוכים מותר לקחת לכל היותר אחד וחצי מן השני.

תיכון האלגוריתם

נשתמש בתכנות דינמי. נחשב שני וקטורי עזר:

$B[i]$ - הסכום המרבי המקבלים מן המערך $A[1:i]$ כאשר $A[i]$ כלול בסכום.

$C[i]$ - הסכום המרבי המקבלים מן המערך $A[1:i]$ כאשר $1/2 \cdot A[i]$ כלול בסכום.

תיכון האלגוריתם (המשך)

$B[1] = A[1]$

$C[1] = 1/2 \cdot A[1]$

$B[2] = A[2] + 1/2 \cdot A[1].$

$C[2] = 1/2 \cdot A[2] + A[1].$

$B[3] = A[3] + C[2]$

$C[3] = 1/2 A[3] + \max(C[2], B[2])$

$B[4] = A[4] + C[3]$

ובאופן כללי

$C[i] = 1/2 A[i] + \max(C[i-1], B[i-1])$

$B[i] = A[i] + C[i-1]$

והפלט הוא

$\max(B[n], C[n])$

דוגמא

A	5	2	3	8	4	6	9	8
B	5	4.5	9	15.5	17	23.5	29	36
C	2.5	6	7.5	13	17.5	20.5	28	33

והפלט הוא $\max(B[n], C[n]) = 36$

חישוב תרומת כל איבר

לכל i ברצוננו לדעת האם בסכום המירבי מופיע $A[i]$ או $1/2 \cdot A[i]$.

נוסיף עוד מערך עזר ונסמן בו לכל i, $3 \le i \le n$, האם $C[i]$ התקבל מ $B[i-1]$ או מ $C[i-1]$.

	1	2	3	4	5	6	7	8
A	5	2	3	8	4	6	9	8
B	5	4.5	9	15.5	17	23.5	29	36
C	2.5	6	7.5	13	17.5	20.5	28	33
	-	-	C	B	B	C	B	B

ונקבל

$$out = A[8] + 1/2 \cdot A[7] + A[6] + 1/2 \cdot A[5]$$
$$A[4] + 1/2 \cdot A[3] + 1/2 \cdot A[2] + A[1]$$

תרגילים

1. הציגו את האלגוריתם בעזרת קוד דמה.
2. הוכיחו את נכונות האלגוריתם.
3. השלימו את האלגוריתם כך שיחשב את תרומתו של כל איבר של מערך הקלט.
4. הציעו אלגוריתם לפתרון בעיה 7.2 תוך שימוש במערך עזר יחיד.

בעיה 7.3:

נתון מערך A ובו n איברים לוא דווקא חיוביים. מחפשים את הסכום המרבי אשר אפשר לקבל כאשר - מכל שני איברים סמוכים מותר לקחת לכל היותר אחד, כמו כן נתון שחייבים לבחור לפחות באיבר אחד מן המערך.

הרעיון

שוב נשתמש בתכנות דינמי. נחשב את מערך העזר B אשר מגדר כך:

$B[i]$ - הסכום המרבי המקבלים מן המערך $A[1:i]$ כאשר $A[i]$ כלול בסכום.

$B[1] = A[1]$

$B[2] = A[2]$

בעיה 7.3 (המשך)

ההבדל בין בעיה 7.1 (המערך A מכיל רק איברים חיוביים) לבין בעיה 7.3 (יתכן שהמערך A מכיל איברים שליליים) מתבטא בחישוב $A[3]$: אם $A[1] < 0$ וגם $A[2] < 0$, מתקיים לפי ההגדרה כי $B[3] = A[3]$. לכן ערכו של $B[3]$ מתקבל כך:

$B[3] = A[3] + \max\{A[1], 0\}$

בהמשך נקבל:

$B[4] = A[4] + \max\{B[1], B[2], 0\}.$

באופן דומה אפשר לראות כי

$B[i] = A[i] + \max\{B[1], \ldots, B[i-2], 0\}$

<u>בעיה 7.3 (המשך)</u>

לאחר חישוב המערך B כולו, נקבל את הפלט על ידי החזרת האיבר המרבי במערך B. (נמקו זאת!).

לרוע המזל, סיבוכיות האלגוריתם שהתקבל היא **ריבועית**.

שאלה: האם אפשר לתכן אלגוריתם בעל סיבוכיות לינארית?

תשובה: נחשב מערך עזר נוסף C אשר יוגדר כך:

$C[i]$ - פתרון הבעיה עבור המערך $A[1:i]$ כאשר $A[i]$ לא כלול בסכום.

<u>תרגילים</u>

1. כתבו קוד דמה לאלגוריתם בהתאם לקווים המוצעים.
2. הציעו אלגוריתם לינארי לפתרון בעיה 7.3.
3. הציעו דרך לחישוב האיברים המופיעים בפתרון.
4. פתרו את בעיה 2 ללא האילוץ שכל האיברים חיוביים.

<u>בעיה 7.4 - בעית התשלום המינימלי</u>

נתון כביש מהיר ובו n תחנות תשלום.

בכל תחנה התשלום שונה.

מכונית העוברת בכביש צריכה לשלם לפחות בכל שער שלישי. נניח כי יש 10 תחנות, המכונית יכולה לשלם לפי הסדרות הבאות:

3,6,9

1,4,7,8

2,3,5,6,7,10

2,3,4,7,9,10

סדרת התשלום הבאה אינה מותרת:

1,4,8,10

כי לא משלמים בשערים מסי 5,6,7

<u>בעית התשלום המינימלי (המשך)</u>

יש להציע אלגוריתם לפתרון הבעיה הבאה:

קלט - מערך pay עם n איברים, $pay[i]$ הוא התשלום בתחנה i.

פלט - התשלום המינימלי הנדרש למעבר הכביש.

בניית אלגוריתם רקורסיבי

ננסה לפתור את הבעיה בעזרת פרוצדורה רקורסיבית $Mpay$.

לכל $1 \le i \le n$, נגדיר את $Mpay(i)$ כתשלום המינימלי שעלינו לשלם מתחנה מסי 1 ועד תחנה i כאשר ידוע לנו כי עלינו לשלם בתחנה i.

<u>חישוב רקורסיבי של</u> $Mpay(i)$

<u>תנאי הקצה</u>

מן ההגדרה נובע באופן מיידי כי:

$Mpay(1) = pay[1]$

$Mpay(2) = pay[2]$

$Mpay(2) = pay[3]$

ואלו הם **תנאי הקצה**.

הרקורסיה

לכל $4 \le i \le n$, כדי לחשב את $Mpay(i)$ נזכור כי נתון **שאנו חייבים לשלם בתחנה** i והסכום הוא $pay[i]$.

כעת עלינו לברר מהי התחנה האחרונה בה נשלם לפני שנשלם בתחנה i.

האפשרויות הן: תחנה $i-1$, בתחנה $i-2$ או בתחנה $i-3$.

אם בחרנו לשלם בתחנה $i-1$ יהיה עלינו לשלם עוד $Mpay(i-1)$.

אם בחרנו לשלם בתחנה $i-2$ יהיה עלינו לשלם עוד $Mpay(i-2)$.

אם בחרנו לשלם בתחנה $i-3$ יהיה עלינו לשלם עוד $Mpay(i-3)$.

כדי להחליט מהי התחנה הבאה בה נשלם נבצע 3 קריאות רקורסיביות ונחשב את המינימום בין הערכים המוחזרים:

$$Mpay(i) \leftarrow pay[i] + \min\begin{Bmatrix} Mpay(i-1) \\ Mpay(i-2) \\ Mpay(i-3) \end{Bmatrix}$$

<u>החישוב הסופי</u>

כל מכונית שעוברת בכביש חייבת לשלם בתחנה n או בתחנה $n-1$ או בתחנה $n-2$.

מכונית שעוברת בתחנה n תשלם לפחות $Mpay(n)$.

מכונית שעוברת בתחנה $n-1$ תשלם לפחות $Mpay(n-1)$.

מכונית שעוברת בתחנה $n-2$ תשלם לפחות $Mpay(n-2)$.

התשלום המינימלי יתקבל על ידי:

$$fpay \leftarrow \min\begin{Bmatrix} Mpay(n) \\ Mpay(n-1) \\ Mpay(n-2) \end{Bmatrix}$$

<u>האלגוריתם הרקורסיבי</u>

$Mpay(i)$

  if $i = 1$ **or** $i = 2$ **or** $i = 3$

    return$(pay[i])$

$$fpay \leftarrow pay[i] + \min\begin{Bmatrix} Mpay(i+1) \\ Mpay(i+2) \\ Mpay(i+3) \end{Bmatrix}$$

return$(fpay)$

<u>איור 7.4: האלגוריתם הרקורסיבי</u>

האלגוריתם הסופי

כאמור, כדי לקבוע את התשלום הסופי עלינו לבחור לשלם באחת מבין 3 התחנות האחרונות, כפי שמתקבל מן הקוד הבא:

$$\begin{array}{l} compute(pay) \\ fpay \leftarrow \min\begin{Bmatrix} Mpay(n) \\ Mpay(n-1) \\ Mpay(n-2) \end{Bmatrix} \\ \text{return}(fpay) \end{array}$$

איור 7.5: האלגוריתם הסופי

הוכחת נכונות

מיידית.

הסיבוכיות

נתבונן בעץ הקריאות הרקורסיביות. כל קריאה עם $0 \le i \le n-3$ מניבה עץ **טרנרי** מלא (לכל צומת פנימי יש 3 בנים).

בעץ זה, אורך המסלולים מן השורש אל העלים הוא בין $n/3$ לבין n.

אם **נגזום** את העץ בעומק $n/3$, נקבל עץ טרנרי **מאוזן** ובו $3^{n/3}$ עלים.

מספר הקריאות הרקורסיביות גדול מ $3^{n/3}$ כלומר **אקספוננציאלי**. כמו בדוגמאות הקודמות, הסיבה סיבוכיות זו היא קריאה מרובה לשיטה הרקורסיבית $Mpay$.

אלגוריתם בתכנות דינמי

כדי להגיע לאלגוריתם לינארי, נשתמש בתכנות דינמי: למעשה השיטה $Mpay$ נקראת עם **בדיוק** n פרמטרים שונים. נקצה מערך עזר $tMpay$ ובו n איברים. נשתמש במערך $tMpay$ כדי לחשב את ערכי השיטה $Mpay$ כדלהלן:

$$tMpay[1] \leftarrow pay[1]$$
$$tMpay[2] \leftarrow pay[2]$$
$$tMpay[3] \leftarrow pay[3]$$

כעת נחשב את $tMpay[4]$ על ידי

$$tMpay[4] \leftarrow pay[4] + \min\begin{Bmatrix} tMpay[1] \\ tMpay[2] \\ tMpay[3] \end{Bmatrix}$$

אלגוריתם בתכנות דינמי (המשך)

באותו אופן לכל $5 \le i \le n$ בכל פעם שיהיו בידינו האיברים $tMpay[i-1]$, $tMpay[i-2]$ ו- $tMpay[i-3]$ נוכל לחשב את $tMpay[i]$ על ידי

$$pay[i] + \min\begin{Bmatrix} tMpay[i-1] \\ tMpay[i-2] \\ tMpay[i-3] \end{Bmatrix}$$

לאחר מילוי המערך, התוצאה הסופית תוחזר על ידי ביצוע

$$\min\begin{Bmatrix} tMpay[n-2] \\ tMpay[n-1] \\ tMpay[n] \end{Bmatrix}$$

אלגוריתם בתכנות דינמי

$$\begin{array}{l} DPcompute(pay) \\ tMpay[1] \leftarrow pay[1] \\ tMpay[2] \leftarrow pay[2] \\ tMpay[3] \leftarrow pay[3] \\ \textbf{for } i \leftarrow 4 \textbf{ to } n \textbf{ do} \\ tMpay[i] \leftarrow pay[i] + \min\begin{Bmatrix} tMpay[i-1] \\ tMpay[i-2] \\ tMpay[i-3] \end{Bmatrix} \\ \textbf{end for} \\ fpay \leftarrow \min\begin{Bmatrix} tMpay[n] \\ tMpay[n-1] \\ tMpay[n-2] \end{Bmatrix} \\ \textbf{return}(fpay) \end{array}$$

איור 7.6: אלגוריתם בתכנות דינמי

סיכום ההרצאה

שיטת **התכנות דינמי** משמשת להורדת סיבוכיות של אלגוריתמים רקורסיביים בהם מתבצע מספר רב של קריאות רקורסיביות עם ארגומנטים זהים.

אם נבצע כל קריאה רקורסיבית, תתקבל לסיבוכיות גבוהה ביותר. במקום זאת, נבצע את הקריאות הקרורסיביות אחת אחר השניה, באופן שיטתי.

בכל פעם שנסיים פתרון תת בעיה כלשהי, נאחסן את הפתרון כך שיהיה זמין לשימוש חוזר ככל שיידרש.

סיכום הרצאה - תיכון אלגוריתמים בשיטת התכנות הדינמי

1. מציגים אלגוריתם רקורסיבי לפתרון הבעיה.
2. מגלים כי סיבוכיות האלגוריתם הרקורסיבי היא גבוהה, על אף שמספר צרופי הפרמטרים האפשריים הוא נמוך יחסית.
3. מגלים כי הסיבה לסיבוכיות הגבוהה היא ביצוע קריאות חוזרות עם וקטור פרמטרים זהה.
4. מגדירים מערך עזר המיועד לאחסון הערכים המחושבים על ידי השיטה הרקורסיבית.

תיכון אלגוריתמים בשיטת התכנות הדינמי (המשך)

5. מחשבים את הערכים המוחזרים על ידי השיטה הרקורסיבית החל בתנאי הקצה. מאחסנים במערך העזר כל אחד מן הערכים המחושבים, ומשתמשים בערכים אלה לחישוב Bottom Up של שאר הערכים.
6. כתוצאה מקבלים אלגוריתם איטרטיבי. מוכיחים את נכונות האלגוריתם כאשר השמורה תעיד על שוויון הערכים במערך העזר לערכי השיטה הרקורסיבית המקורית.

הרצאה 8: תכנות דינמי עם כמה משתנים - חישוב תת הסדרה המשותפת הארוכה ביותר

בהרצאה זו נשתמש בתכנות דינמי כדי לפתור את בעיית

תת הסדרה המשותפת המכסימלית (Longest Common Subsequence)

או **LCS**. להלן תיאור הבעיה:

קלט: שתי סדרות תוים X ו- Y

פלט: תת סדרה משותפת שארכה מכסימלי. להלן נקרא לפלט בשם "LCS של X ושל Y", ונסמנה על ידי $LCS(X,Y)$.

שימו לב: יתכנו מספר תת סדרות מכסימליות.

דוגמא

$$X=(a,c,d,b,a,e,d,b,c,a,e,b,f)$$
$$Y=(c,a,b,c,e,a,b,f,d,a,c,d,f)$$
$$LCS1(X,Y)=(c,a,b,c,e,b,f)$$
$$LCS2(X,Y)=(c,a,b,c,a,b,f)$$

פיתוח אלגוריתם

כעת נציג מספר למות שישמשו לפיתוח אלגוריתם **רקורסיבי.**

למה 8.1 - תנאי קצה

תהיינה X ו Y שתי סדרות. אם אחת הסדרות ריקה אז תת הסדרה המשותפת המכסימלית גם היא ריקה. נסמן את הסדרה הריקה ב Λ ונקבל: לכל X ולכל Y

$$LCS(X,\Lambda)=LCS(\Lambda,Y)=\Lambda$$

הוכחה

האורך של תת הסדרה המכסימלית **קטן או שוה** לאורך כל אחת מן הסדרות.

סימונים

תהיינה X ו Y שתי סדרות.

איברי הסדרות X ו Y יקרא **תוים.**

נניח כי $X=(x_1,\dots,x_n)$ ו- $Y=(y_1,\dots,y_m)$.

הרישא ה i של X, $0\le i\le n$, תסומן על ידי $X_i=(x_1,\dots,x_i)$

הרישא ה j של Y, $0\le j\le m$, תסומן על ידי $Y_j=(y_1,\dots,y_j)$.

למה 8.2- קיצור הקלט

תהיינה $X=(x_1,\dots,x_n)$ ו- $Y=(y_1,\dots,y_m)$ סדרות.

1. אם $x_n=y_m$, אזי

$$LCS(X,Y)=LCS(X_{n-1},Y_{m-1})\circ x_n$$

כלומר: אם התו האחרון של X, שווה לתו האחרון של Y, אזי כל LCS של X ו- Y מתקבלת על ידי שרשור של אחת מתת הסדרות המשותפות הארוכות ביותר של X_{n-1} ו- Y_{m-1} עם התו המשותף x_n.

דוגמא

$$X=(a,c,b,d,f)$$
$$Y=(b,c,d,a,f)$$

במקרה זה:

$$LCS(X,Y)=$$
$$=LCS(X_4,Y_4)\circ f=(c,d,f)$$

הוכחת למה 3.2

תהי $Z=(z_1,\dots,z_k)$ תת סדרה מרבית, כלומר תת סדרה משותפת שאינה ניתנת להארכה, של X ושל Y. נניח בשלילה כי $z_k\ne x_n$. במצב זה, ברור כי $Z\circ x_n$ היא תת סדרה משותפת של X ושל Y - סתירה להנחתנו כי Z אינה ניתנת להארכה. **מש״ל**

למה 8.3

יהיו Xו- Y סדרות. אם $x_n\ne y_m$, אזי

$$LCS(X,Y)=$$
$$LCS(X,Y_{m-1})\,|\,LCS(X_n,Y)$$

כלומר: אפשר להוריד תו אחד מן הקצה הימני של אחת הסדרות, בלי לפגוע ב- LCS.

דוגמא

$$X=(c,b,d,g,f)$$
$$Y=(b,c,d,g)$$

במקרה זה אפשר להוריד את התו האחרון מן הסדרה X ולקבל

$$LCS(X,Y)=$$
$$LCS(X_3,Y_4)=(c,d,g)$$

הוכחה

אם התו האחרון באחת הסדרות משתתף ב-LCS, הוא חייב להיות במקום האחרון של הLCS. במקום זה יכול להיות תו אחד בלבד. לכן, אפשר להוריד את התו האחרון בסדרה השניה.

אם שני התווים האחרונים אינם משתתפים בLCS, בוודאי אפשר להוריד כל אחד מהם.

מש"ל

אלגוריתם רקורסיבי לחישוב אורך LCS

בהנתן שתי סדרות Xו Y, אפשר לחשב ישירות את תת הסדרה המכסימלית

אולם, מסתבר כי קל יותר לחשב תחילה את האורך של תת הסדרה הזו ורק לאחר מכן לחשב את תת הסדרה המכסימלית עצמה.

להלן נציג אלגוריתם הרקורסיבי לחישוב אורך ה-LCS תוך שימוש במשפטים שלמדנו.

סימונים

בהנתן X ו Y, נסמן ב $L(X,Y)$ את אורך תת הסדרה המכסימלית של X ו-Y.

לכל i ו-j, $0 \le i \le n$, $0 \le j \le m$, נסמן ב-$L(X_i,Y_j)$ את אורך תת הסדרה המכסימלית של X_i ו-Y_j.

נוסחת נסיגה לחישוב $L(X_i,Y_j)$

תנאי קצה

מלמה 3.1 נובע כי אם אורכה של אחת הסדרות הוא 0, אז אורך ה-LCS גם הוא 0. כלומר:

לכל X ו Y ולכל i ו j $0 \le i \le n$ ו $0 \le j \le m$;:

$$L(X_i,\Lambda) = L(\Lambda,Y_j) = 0$$

נוסחת נסיגה

מלמה 3.2 נובע כי

לכל X ו Y ולכל i ו-j $1 \le i \le n$ ו $1 \le j \le m$, אם $x_i = y_j$, אז

$$L(X_i,Y_j) = L(X_{i-1},Y_{j-1})+1$$

מלמה 3.3 נובע כי אם $x_i \ne y_j$, אז

$$L(X_i,Y_j) = \max\{L(X_{i-1},Y_j), L(X_i,Y_{j-1})\}$$

אלגוריתם רקורסיבי לחישוב $L(X_i,Y_j)$

$L(X_i,Y_j)$

if $i = 0$ **return** 0

if $j = 0$ **return** 0

if $x_i = y_j$

return $L(X_{i-1},Y_{j-1})+1$

else return

$\max\{L(X_{i-1},Y_j), L(X_i,Y_{j-1})\}$

end

נכונות וסיבוכיות

נכונות האלגוריתם נובעת מיידית מלמות 3.1-3.3.

סיבוכיות האלגוריתם תלויה במספר הקריאות הרקורסיביות. במקרה הגרוע ביותר, בו כל אותיות הקלט שונות זו מזו. במקרה זה, כל קריאה לאלגוריתם, עם קלטים שאינם ריקים ואשר סכום אורכיהם n, מבצעת שתי קריאות רקורסיביות עם קלטים שסכום אורכיהם הוא $n-1$ ולכן:

$$T(n) = 2T(n-1)+c$$

במקרה זה, הסיבוכיות של T היא אקספוננציאלית.

אלגוריתם בתכנות דינמי

נשים לב כי לכל זוג סדרות, X ו Y, מספר צרופי הפרמטרים השונים לפרוצדורה L, כולל סדרות ריקות, הוא בדיוק

$$(m+1)\cdot(n+1)$$

במצב זה, אפשר להחליף את הפונקציה הרקורסיבית $L(,)$ במערך דו-ממדי $TL[,]$ מסדר $(m+1)\times(n+1)$ כאשר בסיום האלגוריתם נקבל $TL(i,j) = L(X_i,Y_j)$, כלומר האיבר ה-(i,j) יכיל את **האורך** של תת הסדרה המכסימלית של X_i ושל Y_j.

האלגוריתם שנציע, יחשב את ערכי המערך, תוך שימוש במשפט שהוכחנו.

תאור האלגוריתם בתכנות דינמי

האלגוריתם מאפס תחילה את השורה ה 0 ואת העמודה ה 0 של המטריצה המתאימות לאורך LCS של שתי סדרות שאחת מהן ריקה.
לאחר מכן, האלגוריתם מחשב את איברי המטריצה, שורה אחר שורה, תוך שימוש בנוסחת הנסיגה מן האלגוריתם הרקורסיבי, ובערכי המטריצה שחושבו כבר.
עם סיום האלגוריתם מתקיים:

$$|LCS(X,Y)| = L(X_n, Y_m) = TL[n,m]$$

T

האלגוריתם - הקוד

```
DPL(X,Y)
for i ← 0 to n do TL[i,0] ← 0
for j ← 1 to n do TL[0,j] ← 0
for i ← 1 to m do
  for j ← 1 to n do
    if x_i = y_j then
      TL[i,j] ← TL[i-1,j-1]+1
    else
      TL[i,j] ←
        max{T[i-1,j],T[i,j-1]}
    endif
  end
end
return TL(n,m)
```


דוגמא

נתבונן בדוגמא שהצגנו בתחילת ההרצאה:

$X = (a,c,d,b,a,e,d,b,c,a,e,b,f)$

$Y = (c,a,b,c,e,a,b,f,d,a,c,d,f)$

להלן, נבנה את המערך TL עבור הדוגמא הזו.

דוגמא

	Λ	c	a	b	c	e	a	b	f	d	a	c	d	f
Λ														
a														
c														
d														
b														
b														
e														
d														
b														
c														
a														
e														
b														
f														

דוגמא

	Λ	c	a	b	c	e	a	b	f	d	a	c	d	f
Λ	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	1	1	1	1	1	1	1	1	1	1	1	1
c	0	1	1	1	2	2	2	2	2	2	2	2	2	2
d	0	1	1	1	2	2	2	2	2	3	3	3	3	3
b	0	1	1	2	2	2	2	3	3	3	3	3	3	3
b	0	1	2	2	2	2	3	3	3	3	4	4	4	4
e	0	1	2	2	2	3	3	3	3	3	4	4	4	4
d	0	1	2	2	2	3	3	3	3	4	4	4	5	5
b	0	1	2	3	3	3	3	4	4	4	4	4	5	5
c	0	1	2	3	4	4	4	4	4	4	4	5	5	5
a	0	1	2	3	4	4	5	5	5	5	5	5	5	5
e	0	1	2	3	4	5	5	5	5	5	5	5	5	5
b	0	1	2	3	4	5	5	6	6	6	6	6	6	6
f	0	1	2	3	4	5	5	6	7	7	7	7	7	7

חישוב תת הסדרה המכסימלית - תכונות המערך TL

לאחר שסיימנו את ביצוע האלגוריתם, אורך תת הסדרה המכסימלית נמצא באיבר $TL[n,m]$.
ברצוננו לפתח אלגוריתם לחישוב תת הסדרה המכסימלית עצמה.
נשים לב כי:

1. הפרש הערכים בין כל שני תאים סמוכים הוא 0 או 1.
2. הערך באיבר ה$TL[i,j]$ מחושב כפונקציה של האיברים $TL[i-1,j]$, $TL[i,j-1]$ ו-$TL[i-1,j-1]$.

תאור האלגוריתם

להלן מוצגות את חמש התצורות האפשריות לקבוצות בנות ארבעה תאים סמוכים במערך TL.
אנו מניחים כי האיבר הימני בשורה התחתונה הוא $TL[i,j]$ ומתקיים $TL[i,j]=k$, כמתואר בטבלה הבאה:

$TL[i-1,j]$	$TL[i-1,j-1]$
$TL[i,j]=k$	$TL[i,j-1]$

לטבלה זו, אנו קוראים **התצורה של** $TL[i,j]$.

חישוב תת הסדרה המכסימלית - תצורות 1-4

בכל אחת מארבעת התצורות הבאות הערך $TL[i,j]=k$, יכול היה להתקבל מאחד האיברים הסמוכים, ללא תוספת תו ל LCS.

k	k
k	k

תצורה 1

k	$k-1$
k	k

תצורה 2

k	$k-1$
k	$k-1$

תצורה 3

$k-1$	$k-1$
k	k

תצורה 4

חישוב תת הסדרה המכסימלית - תצורה 5

בתצורה הבאה הערך $TL[i,j]=k$ התקבל **אך ורק** על ידי זיהוי תו משותף ותוספת 1 לערך $TL[i-1,j-1]$.

$k-1$	$k-1$
$k-1$	k

אלגוריתם לחישוב תת הסדרה המכסימלית - איתחול

כדי לבנות את תת הסדרה המשותפת המכסימלית, מבצעים סיור מ $TL(n,m)$ אל $TL(1,1)$. במהלך הסיור בונים את תת הסדרה המשותפת המכסימלית, מן הסוף אל ההתחלה.

לפני תחילת הסיור מאתחלים:

$(j,i)\leftarrow(m,n)$

$LCS\leftarrow\Lambda$

אלגוריתם לחישוב תת הסדרה המכסימלית - צעד

בכל צעד בסיור, עוברים ממיקומנו הנוכחי $TL(i,j)$ אל אחד האיברים הסמוכים.

נניח כי מיקומנו הנוכחי הוא האיבר $TL[i,j]$. המשך הסיור מ$TL[i,j]$, תלוי בתצורה של איבר זה:

אם זוהי אחת מן התצורות 1-3, האיבר הבא הוא $l[i-1,j]$.

שימו לב: בתצורות 1-2, אפשר להתקדם הן אל $l[i-1,j]$ והן אל $l[i,j-1]$. ההחלטה להתקדם אל $l[i-1,j]$ היא שרירותית. התקדמות אל $l[i,j-1]$ תניב תת סדרה משותפת מכסימלית **אחרת**.

אלגוריתם לחישוב תת הסדרה המכסימלית - צעד (המשך)

אם $TL[i,j]$ נמצא תצורה 4, האיבר הבא הוא $TL[i,j-1]$.

בכל המקרים שתוארו עד כה, הערך של $TL[i,j]$, התקבל מאיבר שכן, ללא תוספת תו לתת הסדרה המכסימלית.

אם לעומת זאת $TL[i,j]$ נמצא בתצורה 5, האיבר הבא בסיור הוא $TL[i-1,j-1]$. במקרה זה, חייב להתקיים $x_i=y_j$ ותוך כדי המעבר מ$TL[i,j]$ אל $TL[i-1,j-1]$, אנו מוסיפים לתת הסדרה המשותפת את התו המשותף x_i.

חישוב תת הסדרה המשותפת המכסימלית - צעד

בכל צעד בסיור, עוברים ממיקומנו הנוכחי $TL[i,j]$ אל האיבר שבאמצעותו חישבנו את $TL[i,j]$ כדלהלן:

אם מתקיים השוויון

$TL(i,j)=TL(i-1,j-1)+1$

אזי, בשלב זה תת הסדרה המכסימלית הוארכה על ידי שרשור האיבר המשותף $x_i=y_j$ **לפני** האיברים שנמצאו עד כה. במקרה זה, מבצעים:

$LCS\leftarrow x_i\circ LCS$

$(i,j)\leftarrow(i-1,j-1)$

חישוב תת הסדרה המשותפת המכסימלית - צעד (המשך)

אם השוויון הקודם לא מתקיים, אזי מתקיים לפחות אחד מאי השוויונים הבאים:

$TL(i,j)=TL(i-1,j)$

$TL(i,j)=TL(i,j-1)$

תיתכן גם האפשרות או ששניהם מתקיימים.

בכל אחד ממקרים אלה, הסדרה LCS לא משתנה. כדי להמשיך בסיור, עוברים אל איבר המטריצה המקיים את השוויון. הקוד הסופי מופיע בשקף הבא:

הקוד

$Extract\_LCS(TL)$
$LCS\leftarrow\Lambda$
$(i,j)\leftarrow(n,m)$
while $(i,j)\neq(1,1)$ **do**
if $TL[i,j]=TL[i-1,j]$
    $(i,j)\leftarrow(i-1,j)$
elseif $TL[i,j]=TL[i,j-1]$
    $(i,j)\leftarrow(i,j-1)$
else
  $(TL[i,j]\neq TL[i-1,j]$ **and**
    $TL[i,j]\neq TL[i,j-1])$
    $LCS\leftarrow x_i\circ LCS$
  $(i,j)\leftarrow(i-1,j-1)$
endif
endwhile

סיכום

בהרצאה זו, למדנו פרדיגמה חדשה לתכנון אלגוריתמים הקרויה: **תכנות דינמי (Dynamic Programming)**.

עיקר ההרצאה הוקדש לפתרון בעית **תת הסדרה המכסימלית** תוך שימוש בתכנות דינמי.

הרצאה 9: תכנות דינמי - מכפלת מטריצות בשרשרת

בהרצאה זו, נציג את בעיית **מכפלת מטריצות בשרשרת** ונציג אלגוריתם לפתרונה.

האלגוריתם מהווה הדגמה נוספת של **תכנות דינמי.**

מכפלת מטריצות בשרשרת

תהי M_1 מטריצה ובה x_1 שורות ו-y_1 עמודות. במקרה זה נאמר שממדי M_1 הם (x_1, y_1).

תהי M_2 מטריצה שמימדיה הם (x_2, y_2).

אם $y_1 = x_2$ אז M_1 ו-M_2 ניתנות להכפלה. נסמן $M_{12} = M_1 \times M_2$.

יסמן ב-m_{ij} האיבר ה-(i,j) ב - M_{12}.

$$m_{ij} = \sum_{k=1}^{x_2} a_{ik} b_{kj}$$

כאשר a_{ik} ו-b_{kj}, $1 \le k \le x_2$ הם איברי השורה ה-i והעמודה ה-j ב-M_1 וב-M_2 בהתאמה.

מספר פעולות הכפל הסקאלריות הנדרשות לחישוב המכפלה

לחישוב m_{ij} נדרשות x_2 פעולות כפל סקאלריות. מאחר שמספר אברי המטריצה M_{12} הוא $x_1 y_2$ נקבל כי מספר פעולות הכפל הנדרשות לחישוב M הוא $x_1 x_2 y_2$.

מספר פעולות החיבור הסקלארי הנדרשות דומה. לכן, מעתה נזניח את פעולות החיבור ונדון אך ורק במספר פעולות הכפל הסקלארי הנדרשות לביצוע המכפלה.

מספר פעולות הכפל הסקאלריות הנדרשות לחישוב המכפלה (המשך)

נניח כעת כי ברצוננו לחשב מכפלה של שלוש מטריצות $M_1 \times M_2 \times M_3$ שמימדיהן הן (x_1, y_1), (x_2, y_2), ו-(x_3, y_3) בהתאמה.

נמספר את פעולות כפל המטריצות משמאל לימין, מספר הפעולה הראשונה הוא 1 ומספר השניה הוא 2. כפל מטריצות הוא **אסוציאטיבי** ולכן באפשרותנו לבצע את הכפל בשני **זימונים** (סדרי פעולות) **שונים**, כמפורט בשקף הבא:

מספר פעולות הכפל הסקאלריות הנדרשות לחישוב המכפלה (המשך)

1. הזימון $(1,2)$ מסמן את סדר המכפלות $(M_1 \times M_2) \times M_3$ - במקרה זה מספר הפעולות יהיה:
$x_1 x_2 y_2 + x_1 x_3 y_3 = x_1 x_3 (x_2 + y_3)$
2. הזימון $(2,1)$ מסמן את סדר המכפלות $M_1 \times (M_2 \times M_3)$ - במקרה זה מספר הפעולות יהיה:
$x_2 x_3 y_3 + x_1 x_2 y_3 = x_2 y_3 (x_3 + x_1)$

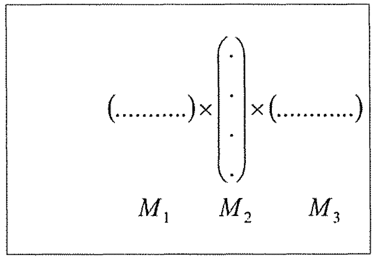
בהנתן שלישית מטריצות ניתנות להכפלה, אפשר לחשב את עלות שני הזימונים ולבחור את הזימון הדורש ביצוע פחות פעולות.

זימון הכפלות אופטימלי

מספרי פעולות הכפל הסקאלריות הנדרשות לביצוע כל אחד משני זימונים אלה שונים זה מזה ותלויים בממדי שלוש המטריצות.

לדוגמא

נניח כי רוצים להכפיל 3 מטריצות שמימדיהן הם: $(x_1, y_1) = (1, n)$, $(x_2, y_2) = (n, 1)$, ו-$(x_3, y_3) = (1, n)$ כמתואר באיור 9.1:



איור 9.1: דוגמה למכפלת מטריצות

M_1 M_2 M_3

זימון הכפלות אופטימלי (המשך)

בדוגמא זו, מספר פעולות הכפל הסקלריות הנדרשות לביצוע המכפלה לפי הזימון $(1,2)$ הוא $2n$, ואילו מספר פעולות הכפל הסקלריות הנדרשות לביצוע המכפלה לפי הזימון $(2,1)$ הוא $2n^2$.

אם נתבקש להכפיל שלוש מטריצות אפשר לחשב את עלות שני הזימונים ולהחליט ביניהם.

נעבור כעת לדון בבעיית בחירת זימון אופטימלי עבור מספר מטריצות גדול.

זימון הכפלות אופטימלי (המשך)

בתחילת הדיון נראה כי אם מספר המטריצות המוכפלות הוא n, אזי מספר הדרכים האפשריות להכפלתן הוא מעריכי (Exponential).

בהמשך ההרצאה נפתח אלגוריתם פולינומי, המשתמש בתכנות דינמי, למציאת זימון שעלותו מינימלית, כלומר, זימון הדורש מספר מינימלי של מכפלות סקאלריות.

הדגמת זימון

בטרם נגדיר את הבעיה חישובית באופן פורמלי, נדגים מהו זימון:

נניח כי $n=5$ כלומר

$M_{15}=M_1\times M_2\times M_3\times M_4\times M_5$

נתבונן בזימון $(4,1,3,2)$.

זימון זה מכתיב כי פעולת הכפל הראשונה שתתבצע היא $M_4\times M_5$.

לאחר ביצוע פעולה זו נישאר עם:

$M_{15}=M_1\times M_2\times M_3\times M_{45}$

כעת נבצע את $M_1\times M_2$ ונישאר עם

$M_{15}=M_{12}\times M_3\times M_{45}$

כעת נבצע את פעולה מסי 3 $M_3\times M_{45}$

ונשאר עם $M_{15}=M_{12}\times M_{35}$.

הדגמת זימון (המשך)

לבסוף נבצע את פעולה מסי 2 ונקבל את התוצאה הסופית, M_{15}.

סדר ביצוע הפעולות המוכתב על ידי הזימון הוא:

$$M_{15}=(M_1\times M_2)\times(M_3\times(M_4\times M_5))$$

שימו לב

בזימון זה, פעולה מסי 2 נקראת **הפעולה הראשית** או **הפעולה האחרונה**.

הגדרה פורמלית של הבעיה

הקלט

n זוגות של מספרים טבעיים $(x_1,y_1),\ldots,(x_n,y_n)$. לכל $1\le i\le n-1$, (x_i,y_i) הוא זוג מימדי המטריצה M_i ולכל $1\le i\le n-1$ מתקיים $y_i=x_{i+1}$.

שימו לב: המטריצות עצמן אינן חלק מן הקלט.

הפלט

1. מספר פעולות הכפל הסקאלריות המינימלי הנדרש לחישוב המכפלה $M=M_1\times\ldots\times M_n$.
2. זימון פעולות כפל המטריצות המניב את מספר הפעולות האופטימלי.

ניסיון 1: האלגוריתם הישיר (בדיקת כל האפשרויות)

סימונים

1. נסמן $M=M_1\times\ldots\times M_n$.
2. לכל $1\le i<j\le n$ נסמן $M_{ij}=M_i\times\ldots\times M_j$
3. לכל $1\le i\le n-1$ מספרה של הפעולה בין M_i לבין M_{i+1} יהיה i.

מאחר שיש n-1 פעולות כפל מטריצות, הזימון (פלט מסי 2) ניתן לביטוי **כתמורה** מתוך S_{n-1}.

האלגוריתם הישיר (המשך)

האלגוריתם הישיר עובר על כל אחד מן הזימונים, מחשב את עלותו (מספר המכפלות הסקאלריות) של כל זימון ובוחר את הזימון בעל העלות המינימלית.

סיבוכיות האלגוריתם הישיר

מאחר שכל זימון הוא תמורה מתוך S_{n-1}, ברור כי מסי האפשרויות הוא $(n-1)!=\Omega(2^n)$ ואפשרות זו נפסלת.

ניסיון 2: אלגוריתם הפרד ופתור

לכל זוג מספרים (i,j), $i\le j$, נסמן ב-$mc(i,j)$ את מספר פעולות הכפל המינימלי הנדרש כדי לחשב את M_{ij}.

כעת נפתח אלגוריתם רקורסיבי לחישוב $mc(i,j)$. האלגוריתם מבוסס על שתי הלמות הבאות:

למה 8.1

לכל $1\le i\le j\le n$, מתקיים:

1. $mc(i,i)=0$.
2. $mc(i,i+1)=x_ix_{i+1}y_{i+1}$.

הוכחה

נכונות סעיף 1 נובע מיידית מהגדרת $mc(i,j)$. סעיף 2 הוכח בתחילת ההרצאה. **מש"ל**

אלגוריתם הפרד ופתור (המשך)

הלמה הבאה מציבה משוואה רקורסיבית כדי לחשב את $mc(i,j)$:

למה 8.2

לכל $1\le i\le j\le n$, מתקיים:

אם $j>i+1$ אז

$$mc(i,j)=\min_{k=i}^{j-1}\left(mc(i,k)+mc(k+1,j)+x_ix_{k+1}y_j\right)$$

להלן נסמן את המשוואה הזו ב-(*).

הוכחה

יהי k מס' טבעי, $1 \le k \le j-1$. נניח כי הפעולה הראשית בחישוב M_{ij} היא פעולה מס' k, כלומר החישוב נראה כך:

$$M_{ij} = M_{ik} \times M_{k+1,j}$$

בתנאים אלה, מס' פעולות הכפל המינימלי הנדרש לחישוב הוא:

$$mc(i,k) + mc(k+1,j) + x_i x_{k+1} y_j$$

מאחר שאיננו יכולים לנחש מהי הפעולה הראשית עלינו לבדוק את כל האפשרויות.

מש"ל

תכונות הנוסחה (*)

נשים לב כי:

1. אם ערך המינימום התקבל עבור k אזי נובע כי פעולה k היא **הפעולה הראשית (האחרונה)** בזימון אופטימלי.
2. המספר המינימלי של פעולות הכפל הסקאלריות הנדרש להכפלת כל n המטריצות הוא כמובן $mc(1,n)$.

האלגוריתם הרקורסיבי – הקוד

```
mc(i, j)
    if (j = i) return(0)
    if (j = 1+1) return(x_i · x_j · y_j)
    MC = min_{k=i}^{j-1} {mc(i,k) +
            + mc(k+1, j) + x_i x_{k+1} y_j}
return(MC)
end
```

אלגוריתם 9.1 - חישוב $mc(i,j)$

נכונות האלגוריתם

האלגוריתם הוא פשוט ביותר. תנאי הקצה נובעים מלמה 8.1 והרקורסיה מבוססת ישירות על למה 8.2.

סיבוכיות האלגוריתם

כדי לחסום את מלמטה את סיבוכיות האלגוריתם, נתבונן בעץ הקריאות הרקורסיביות: בעץ זה, דרגת היציאה (מספר הקריאות הרקורסיביות) של כל צומת, נעה בין $2(n-1)$ (עבור השורש) לבין 2 (ברמה התחתונה) ואורך כל ענף גדול מ-$n-1$.

קל לראות כי מספר הצמתים בעץ גדול (בהרבה מאוד) מ-2^n ומכאן נובע כי סיבוכיות האלגוריתם היא מעריכית (אקספוננציאלית)

בנספח 1 אנו מוכיחים כי סיבוכיות האלגוריתם חסומה מלמטה על ידי 2^{n-1}.

מסקנה: גם שיטה זו בזבזנית מדי.

ניסיון 3: תכנות דינמי

כדי לפתח אלגוריתם יעיל עלינו למצוא את הגורם לבזבוז בשיטה הקודמת:

אם נפתח את המשוואות הרקורסיביות שלבים נוספים, נראה כי ישנם זוגות מספרים (k,l) כך ש $mc(k,l)$ מחושב פעמים רבות.

לדוגמה:

כדי לחשב את $mc(1,5)$ יש למצוא את המינימום בין:

$$mc(1,1) + mc(2,5) + x_1 x_2 y_5$$
$$mc(1,2) + mc(3,5) + x_1 x_3 y_5$$
$$mc(1,3) + mc(4,5) + x_1 x_4 y_5$$
$$mc(1,4) + mc(5,5) + x_1 x_5 y_5$$

דוגמה (המשך)

כדי לחשב את $mc(2,5)$ יש למצוא את המינימום בין:

$$mc(2,2) + mc(3,5) + x_2 x_3 y_5$$
$$mc(2,3) + mc(4,5) + x_2 x_4 y_5$$
$$mc(2,4) + mc(5,5) + x_2 x_5 y_5$$

כבר כאן אנו רואים כי בדרך זו אנו מחשבים את $mc(3,5)$ ואת $mc(4,5)$ פעמיים.

אם נמשיך בדוגמא, הכפילות תלך ותגדל וזמן הריצה יגיע לפונקציה מעריכית.

תכנות דינמי

נשים לה כי לכל אחד משני הארגומנטים באלגוריתם הרקורסיבי $mc(i,j)$ יש בדיוק n ערכים מותרים. מכאן נובע כי מספר צרופי הארגומנטים האפשריים הוא n^2. כדי להמנע מחזרה מיותרת על חישובי ביניים, נשמור את התוצאה של כל חישוב **בטבלת מעקב** (Lookup Table) בגודל n^2.

נסמן את טבלת המעקב באות T. בכל פעם שנחשב את $mc(k,l)$ עבור k ו-l כלשהם, נאחסן את הערך שחישבנו ב-$T[k,l]$.

כאשר נדרש לערך זה בפעם הבאה, נשלוף אותו מן הטבלה במקום לחשבו מחדש.

תכנות דינמי (המשך)

למעשה האלגוריתם יתקדם על ידי חישוב $mc(i,j)$ עבור זוגות (i,j) שמרחקם ההדדי, $j-i$ גדל והולך, כדלקמן:

<u>שלב 0:</u> חשב באופן ישיר, על ידי שימוש בתנאי קצה מס' 1 את אלכסון 0 (האלכסון הראשי):

$$T[1,1],\ldots,T[n,n]$$

<u>שלב 1:</u> חשב באופן ישיר, על ידי שימוש בתנאי קצה מס' 2 את אלכסון 1 (האלכסון מעל האלכסון הראשי):

$$T[1,2],\ldots,T[n-1,n]$$

תכנות דינמי (המשך)

<u>שלב 2:</u> השתמש בנוסחה (*) ובתוצאות שלבים 0 ו-1 כדי לחשב את אלכסון 2:

$$T[1,3],\ldots,T[n-2,n]$$

<u>שלב i:</u> השתמש בנוסחה (*) ובתוצאות שלבים 0 עד $i-1$ כדי לחשב את אלכסון i:

$$T[1,i+1],\ldots,T[n-i,n]$$

בשלב $n-2$ מחשבים את $T[1,n-1]$ ואת $T[2,n]$.

בשלב $n-1$ מחשבים את $T[1,n]$.

<u>שימו לב:</u> בשלב i, $0\le i\le n$, מחשבים $n-i$ איברים.

תכנות דינמי (המשך)

כפי שמודגם באיור 9.2, מתקבל מערך ובו n שורות ו-n עמודות. חישוב הערכים במערך מתבצע כך: האלכסון הראשי מתמלא ראשון ואחריו האלכסונים מעל הראשי בזה אחר זה.

$$\begin{pmatrix}
(1,1)(1,2)\ldots\ldots(1,n-1)(1,n)\\
(2,2)(2,3)\ldots\ldots(2,n-1)(2,n)\\
(3,3)\ldots\ldots(3,n-1)(3,n)\\
(4,4)\ldots\ldots(4,n)\\
\ldots\ldots\\
\ldots\ldots\\
\ldots\ldots(n-1,n-1)(n-1,n)\\
\ldots\ldots(n,n)
\end{pmatrix}$$

<u>איור 9.2: הטבלה T</u>

אלגוריתם Dmc לחישוב הטבלה T

```
Dmc(x_1, y_1, ..., x_n, y_n)
for i = 1 to n
                 \\ חישוב אלכסון 0
      T[i,i] <- 0
for i = 1 to n-1
                 \\ חישוב אלכסון 1
      T[i,i+1] <- x_i x_{i+1} y_{i+1}
for l = 2 to n-1
         \\ 2 <= l <= n-1 ,l חישוב אלכסון
   for i = 1 to n-l
      j <- i+l
      T[i,j] <- min_{k=i}^{j-1} {T[i+k] +
                T[k+1,j] + x_i x_{k+1} y_j
   end for
end for
Out(T)
```

<u>אלגוריתם 9.2 – חישוב טבלת המעקב</u>

סיבוכיות האלגוריתם

<u>טענה</u>

סיבוכיות האלגוריתם Dmc היא $\Theta(n^3)$.

<u>הוכחה</u>

כדי להוכיח זאת נחסום את זמן החישוב של המערך T מלמעלה ומלמטה.

שימו-לב: זמן החישוב של הזימון האופטימלי הוא לינארי ואינו משנה את סיבוכיות האלגוריתם.

חסימה הסיבוכיות מלמעלה

למעשה אנו מחשבים כאן איברי מערך ריבועי מסדר $n\times n$. חישוב איברי המערך, השונים מ-0, מתבצע בעזרת הנוסחה (*).

בנוסחה זו, אנו מוצאים את המינימום בין לכל היותר n איברים ולכן מספר הצעדים לחישוב כל איבר הוא $O(n)$.

מכאן, מספר הצעדים הכללי הוא לכל היותר:

$$O(n^2\cdot n)=O(n^3)$$

חסימה הסיבוכיות מלמטה

מספר האיברים השונים מ-0 במחצית העליונה של המערך T הוא:

$$\Omega\left(\underbrace{\frac{n}{2}}_{\text{גובה המשולש}}\cdot\underbrace{\frac{n}{2}}_{\text{אורך הבסיס}}\cdot\frac{1}{2}\right)=\Omega(n^2)$$

אפשר גם לחשב מספר זה במדויק כדלהלן:

בשורה $n/2-1$ יש $n/2$ איברים $\neq 0$

בשורה $n/2-2$ יש $n/2-1$ איברים $\neq 0$

...

בשורה 1 יש איבר יחיד $\neq 0$

חסימה הסיבוכיות מלמטה (המשך)

לפי נוסחת הסכום של טור חשבוני, מס' האיברים הכולל השונה מ-0 במחצית העליונה של המערך T הוא:

$$\left(\frac{n}{2}+1\right)\cdot\frac{n}{2}\cdot\frac{1}{2}=\frac{n^2}{8}+\frac{n}{4}$$

חישוב כל איבר כזה דורש לפחות $n/2$ פעולות כלומר $\Omega(n)$ פעולות (חישוב מינימום של מספר איברים $\le n/2$) ומכאן מספר הפעולות הכולל הוא $\Omega(n^3)$.

חישוב הזימון האופטימלי

לאחר חישוב הטבלה T, ידוע לנו כי המספר המינימלי של מכפלות סקלריות הנדרש לחישוב המטריצה $M = M_1 \times \ldots \times M_n$ מאוחסן ב-$T[1,n]$. כעת נסביר מהי התוספת הנדרשת לחישוב הזימון האופטימלי עצמו: לכל $1 \le i \le j \le n$ נסמן את הפעולה האחרונה בזימון אופטימלי לחישוב M_{ij} ב-$LastOp(i,j)$. להלן נציג בטבלה T את הערכים $mc(i,j)$ ביחד עם $LastOp(i,j)$

זימון מכפלת מטריצות – דוגמא

להלן נדגים את חישוב הטבלה T, בצירוף חישוב הפעולה האחרונה, על הקלט הבא:

קלט: $(7,3),(3,8),(8,4),(4,7),(7,5)$

חישוב אלכסון 0 (האלכסון הראשי)

ערכי האיברים באלכסון הראשי הם 0. ערכי האיברים באלכסון הראשון מעל האלכסון הראשי, מחושבים ישירות מתנאי הקצה השני.

$$T[1,1],\ldots,T[5,5]=0$$

חישוב אלכסון 1

נחשב את הערכים באלכסון 1 בחישוב ישיר, בעזרת תנאי הקצה המופיע בסעיף 1 בלמה 9.2, וכך נקבל:

$$T[1,2]=7\cdot 5\cdot 8=280$$
$$T[2,3]=3\cdot 8\cdot 4=96$$
$$T[3,4]=8\cdot 4\cdot 7=224$$
$$T[4,5]=4\cdot 7\cdot 5=140$$

מן ההגדרה ברור כי לכל i, $1 \le i \le n-1$, מתקיים:

$$LastOp(i,i+1)=i.$$

חישוב אלכסון 1 (המשך)

איור 9.3 מציג את הטבלה T לאחר חישוב אלכסונים 0 ו-1, כאשר האות Λ מסמנת איבר שטרם הגדר.

$$\begin{pmatrix} 0 & |168(1)| & \Lambda & | \Lambda & | \Lambda \\ \Lambda | & 0 & |96(2)| & \Lambda & | \Lambda \\ \Lambda | & \Lambda & | \ 0 & |224(3)| & \Lambda \\ \Lambda | & \Lambda & |\Lambda & | \ 0 & |140(4) \end{pmatrix}$$

איור 9.3: הטבלה T לאחר חישוב אלכסונים 0 ו-1

האות Λ מסמנת איבר שטרם הוגדר.

חישוב אלכסון 2

נעבור כעת לחישוב אלכסון מס׳ 2. (האלכסון השני מעל האלכסון הראשי):

$$mc(1,3)=\min\{$$
$$mc(1,1)+mc(2,3)+x_1\cdot x_2\cdot y_3,$$
$$mc(1,2)+mc(3,3)+x_1\cdot x_3\cdot y_3\}=$$
$$=\min\{0+96+84,168+0+224\}=180$$

המינימום מתקבל עבור האיבר הראשון והפעולה הראשית היא פעולה מס׳ 1. הזימון האופטימלי הוא

$$M_{13}=M_1\times(M_2\times M_3)$$

חישוב אלכסון 2 (המשך)

באופן דומה:

$$mc(2,4)=\min\{$$
$$mc(2,2)+mc(3,4)+x_2\cdot x_3\cdot y_4,$$
$$mc(2,3)+mc(4,4)+x_2\cdot x_4\cdot y_4\}=$$
$$=\min\{0+224+168,96+0+84\}=180$$

הפעולה הראשית היא פעולה מס׳ 3 והזימון האופטימלי הוא

$$M_{24}=(M_2\times M_3)\times M_4$$

חישוב אלכסון 2 $mc(3,5)$

$$mc(3,5)=\min\{$$
$$mc(3,3)+mc(4,5)+x_3\cdot x_4\cdot y_5,$$
$$mc(3,4)+mc(5,5)+x_3\cdot x_5\cdot y_5\}=$$
$$\min\{0+140+160,224+0+280\}=300$$

הפעולה הראשית היא פעולה מס׳ 3, והזימון האופטימלי הוא

$$M_{35}=M_3\times(M_4\times M_5)$$

$$\begin{pmatrix} 0 & |168(1) & |180(1)| & \Lambda & | \Lambda \\ \Lambda | & 0 & |96(2) & |180(3) & | \Lambda \\ \Lambda | & \Lambda & | \ 0 & |224(3) & |300(3) \\ \Lambda | & \Lambda & |\Lambda & | \ 0 & |140(4) \end{pmatrix}$$

איור 9.4: הטבלה T לאחר חישוב אלכסון 2

חישוב אלכסון 3

$$mc(1,4)=\min\{$$
$$mc(1,1)+mc(2,4)+x_1\cdot x_2\cdot y_4,$$
$$mc(1,2)+mc(3,4)+x_1\cdot x_3\cdot y_4,$$
$$mc(1,3)+mc(4,4)+x_1\cdot x_4\cdot y_4\}=$$
$$=\min\{0+180+147,168+224+392,$$
$$180+0+196\}=327$$

הפעולה הראשית היא פעולה מס׳ 1, הזימון האופטימלי לחישוב M_{14} הוא

$$M_{14}=M_1\times M_{24}$$

חישוב אלכסון 3 (המשך)

נמשיך בחישוב $mc(2,5)$

$$mc(2,5) = \min\{$$
$$mc(2,2) + mc(3,5) + x_2 \cdot x_3 \cdot y_5,$$
$$mc(2,3) + mc(4,5) + x_2 \cdot x_4 \cdot y_5,$$
$$mc(2,4) + mc(5,5) + x_2 \cdot x_5 \cdot y_5\} =$$
$$= \min\{0 + 300 + 120, 96 + 140 + 60,$$
$$180 + 0 + 105\} = 285$$

הפעולה הראשית היא פעולה מסי׳ 4 .

הזימון האופטימלי הוא:

$$M_{25} = M_{24} \times M_5$$

חישוב אלכסון 4

נסיים את חישוב המערך בחישוב האלכסון הרביעי ובו האיבר היחיד $mc(1,5)$:

$$mc(1,5) = \min\{$$
$$mc(1,1) + mc(2,5) + x_1 \cdot x_2 \cdot y_5,$$
$$mc(1,2) + mc(3,5) + x_1 \cdot x_3 \cdot y_5,$$
$$mc(1,3) + mc(4,5) + x_1 \cdot x_4 \cdot y_5,$$
$$mc(1,4) + mc(5,5) + x_1 \cdot x_5 \cdot y_5\} =$$
$$= \min\{0 + 285 + 105, 168 + 295 + 280,$$
$$180 + 135 + 140, 327 + 0 + 245\} = 390$$

הפעולה הראשית היא פעולה מספר 1 והזימון האופטימלי לחישוב M הוא

$$M = M_1 \times M_{25}$$

סיכום הדוגמא

המערך הסופי נראה כך:

$$\begin{pmatrix} 0 & |168(1) & |180(1) & |327(1) & |390(1) \\ \Lambda| & 0 & |96(2) & |180(3) & |285(4) \\ \Lambda| & \Lambda & | \quad 0 & |224(3) & |300(3) \\ \Lambda| & \Lambda & |\Lambda & | \quad 0 & |140(4) \end{pmatrix}$$

איור 9.5: הטבלה T בסיום החישוב

עלות הזימון האופטימלי מופיעה ב-$T[1,5]$

כעת נעבור לחישוב הזימון האופטימלי.

חישוב הזימון האופטימלי

נסמן את הזימון האופטימלי ב-$OptSched$. בתחילת החישוב מתקיים $OptSched = \Lambda$ כעת נתבונן ב-$T[1,5]$ ונקבל:

$$LastOp(1,5) = 1$$

כלומר הזימון האופטימלי לחישוב M_{15} נראה כך: $M_{15} = M_1 \times M_{25}$.

ומכאן $OptSched = (.,.,.,1)$.

כדי להמשיך בחישוב הזימון האופטימלי נתבונן כעת ב-$T[2,5]$

ונמצא כי $mc(2,5) = 285$ וכי

$$LastOp(2,5) = 4$$

חישוב הזימון האופטימלי (המשך)

מכאן נובע כי $OptSched = (.,.,4,1)$

והזימון האופטימלי יהיה:

$$M_{15} = M_1 \times (M_{24} \times M_5)$$

נותר לנו לתזמן את פעולות 2 ו 3.

כעת נתבונן ב-$T[2,4]$ ונגלה כי

$mc(2,4) = 180$ וכי

$LastOp(2,4) = 3$. מכאן נובע כי

$OptSched = (.,3,4,1)$. מכאן נובע כי

הזימון האופטימלי יהיה:

$$M_{15} = M_1 \times ((M_{23} \times M_4) \times M_5)$$

ברור כי הפעולה הראשונה תהיה 2

ולכן הזימון האופטימלי הסופי יהיה:

$OptSched = (2,3,4,1)$.

חישוב הזימון האופטימלי (המשך)

ננסה לוודא זאת על ידי חישוב עלות זימון זה באופן ישיר. הקלט הוא:

$$(7,3),(3,8),(8,4),(4,7),(7,5)$$

נחשב ונקבל:

$$x_2 \cdot x_3 \cdot y_3 + x_2 \cdot x_4 \cdot y_4 +$$
$$x_2 \cdot x_5 \cdot y_5 + x_1 \cdot x_2 \cdot y_5 =$$
$$3 \cdot 8 \cdot 4 + 3 \cdot 4 \cdot 7 +$$
$$3 \cdot 7 \cdot 5 + 7 \cdot 3 \cdot 5 = 390$$

כפי שציפינו.

זימונים שקולים

נניח כי עבור ערכי קלט מסוימים מתקבל זימון הפעולות הבא:

$$OptSched = (4,1,2,3)$$

במקרה זה, הפעולה הראשית המתבצעת היא פעולה מסי׳ 3 אשר **מפרידה** בין פעולות 1,2 לפני פעולה 4. אם נבצע את פעולות 1,2 **לפני** פעולה 4, מספר פעולות הכפל הסקאלריות **לא ישתנה**. הזימונים $Sched = (4,1,2,3)$ ו-$Sched' = (1,2,4,3)$ נקראים **שקולים**.

זימונים שקולים (המשך)

כדי לבחור בין כמה זימונים שקולים, נסכים כי תמיד נבחר בפעולות שמספרן נמוך יותר. בדוגמא הנוכחית הזימון שייבחר הוא $Sched'$.

השגרה *Extract*

השגרה *Extract* מחשבת ומחזירה זימון אופטימלי תוך שימוש בנתונים שנאגרו במערך T.

```
Extract(i, j, T)
if i = j return Λ
last ← lastOp(i, j)
LSched = Extract(i, last)
RSched = Extract(last + 1, j)
Return LSched ∘ RSched ∘ Last
```

אלגוריתם 9.3 - חישוב זימון אופטימלי

שימו לב: הסימון ∘ מסמל פעולת שרשור (Concatenation).

האלגוריתם הסופי

האלגוריתם הסופי משתמש בשגרה *Dmc* כדי לחשב את הטבלה T. לאחר מכן, האלגוריתם מפעיל את השגרה *Extract* אשר מחזירה את הזימון האופטימלי. להלן הקוד עבור האלגוריתם הסופי:

```
OptSched(x1, y1, ..., xn, yn)
  T = Dmc(x1, y1, ..., xn, yn)
  return Extract(1, n, T)
```

אלגוריתם 9.4 – האלגוריתם הסופי לחישוב זימון אופטימלי

סיכום

בהרצאה זו, השתמשנו פעם נוספת **בתכנות דינמי** כדי להציג אלגוריתם פולינומי לחישוב **זימון אופטימלי להכפלת מטריצות בשרשרת**.

נספח 1: סיבוכיות האלגוריתם הרקורסיבי

נסמן ב $T(n)$ את הזמן הנדרש כדי לחשב את $mc(1,n)$, כלומר כדי לחשב זימון אופטימלי למכפלת n מטריצות. אזי מתקיים:

$$T(2)=2$$

$$T(n)=\sum_{k=1}^{n-1}\left(T(k)+T(n-k)+2\right)$$

שימו לב: המספר 2 המופיע בתוך הסכימה שווה לזמן הנדרש לביצוע המכפלה $x_i y_k y_b$.

סיבוכיות האלגוריתם (המשך)

נראה כעת כי הסיבוכיות היא מעריכית.

טענה

$T(n)\ge 2^{n-1}$.

הוכחה

באינדוקציה על מספר המטריצות n.

בסיס

$$T(2)=2=2^1$$

צעד האינדוקציה

$$T(n)=\sum_{k=1}^{n-1}\left(T(k)+T(n-k)+2\right)$$

$$\ge\sum_{k=1}^{n-1}T(k)+T(n-k)+1$$

אם נוציא את 1 מחוץ לסכימה נקבל:

$$T(n)\ge n-1+\sum_{k=1}^{n-1}T(k)+T(n-k)$$

$$\ge n-1+2\sum_{k=1}^{n-1}T(k)$$

צעד האינדוקציה (המשך)

לפי הנחת האינדוקציה, לכל $1\le k<n$, מתקיים $T(k)\ge 2^{k-1}$. כעת, נציב את אי השוויונים הללו בנוסחה הקודמת ונקבל:

$$T(n)\ge n-1+2\sum_{k=1}^{n-1}2^{k-1}$$

$$\ge n-1+2\left(2^{n-1}-1\right)$$

$$=n+2^n-3\ge 2^{n-1}$$

מש"ל

הרצאה 10: משפט האב (Master Theorem) אלגוריתם Strassen למכפלת מטריצות ריבועיות

להרצאה זו, שני חלקים:

1. נציג ונוכיח את משפט האב המאפשר חישוב סיבוכיות של אלגוריתמים רקורסיביים רבים.
2. נציג את אלגוריתם Strassen להכפלת מטריצות ריבועיות ונחשב את הסיבוכיות שלו, בעזרת משפט האב.

משפט האב

קיימים אלגוריתמים רקורסיביים רבים שאנו מתקשים בחישוב הסיבוכיות שלהם. משפט האב מספק לנו מכשיר רב עוצמה לטיפול ברוב האלגוריתמים הרקורסיביים. אנו נלמד גרסה מסוימת של משפט האב. גרסה אחרת נמצאת בספר הקורס.

שימו לב: אתם כבר למדתם את משפט האב בקורס מבנה נתונים.

התנאים להפעלת משפט האב

משפט האב מאפשר לנו לדון באלגוריתמים רקורסיביים המקיימים את התנאים הבאים:

לכל בעיה בגודל n, אשר אינה מקרה קצה, האלגוריתם מבצע a קריאות רקורסיביות לפתרון a תת בעיות בגודל n/b.

שימו לב

1. a ו b הם קבועים התלויים בבעיה אך לא ב n.
2. a תת הבעיות אינן בהכרח חלוקה של הקלט ל a חלקים זרים.

דוגמא

מיון ע"י מיזוג:

1. **חלק** את מערך הקלט ל 2 מערכים בגודל $n/2$. ($a=b=2$).
2. **מיין** כל תת-מערך באופן על ידי קריאה רקורסיבית למיון-מיזוג.
3. **מזג** את שתי המחציות הממוינות.

לעומת דוגמא זו, **מיון מהיר** אינו עונה לדרישות אלה משום שבמיון מהיר מחלקים את מערך הקלט לשני קבצים שהיחס בין גדליהם אינו קבוע.

הצגת סיבוכיות מיון-מיזוג על ידי מערכת משוואות רקורסיביות

נסמן ב $T(n)$ את הזמן הדרוש למיון-מיזוג מערך בן n איברים.

הזמן הנדרש לכל קריאה רקורסיבית הוא כמובן $T\left(\frac{n}{2}\right)$ ומאחר שמיזוג של שני מערכים באורך $n/2$ כל אחד אורך זמן ליניארי ומיון קובץ באורך 1 אורך זמן קבוע כלשהו נקבל כי:

$$T(n)=2T\left(\frac{n}{2}\right)+dn$$

$$T(1)=c$$

עבור c ו d קבועים כלשהם.

הכללת מערכת המשוואות

נתבונן באלגוריתם רקורסיבי A לפתרון בעיה כלשהי P. נניח כי אם מריצים את A על קלט בגודל n אזי:

1. נדרשות a קריאות רקורסיביות עם קלט בגודל n/b.
2. מספר הפעולות בשגרה הקוראת (top level), למעט הקריאות הרקורסיביות, הוא $df(n)$ כאשר $f(n)$ היא פונקציה כלשהי התלויה בגודל הקלט.
3. פתרון מקרה הקצה של הרקורסיה, עבור קלט בגודל 1, דורש c פעולות.
4. הקבועים a, b, c, d והפונקציה f תלויים באלגוריתם A.

הכללת מערכת המשוואות (המשך)

נסמן ב$T(n)$ את הזמן הנדרש להרצת האלגוריתם A על קלט בגודל n.

אזי מתקיים:

$$T(n)=aT\left(\frac{n}{b}\right)+df(n)$$

$$T(1)=c$$

להלן נציג את משפט האב (Master Theorem) הפותר את מערכת המשוואות הזו באופן כללי.

משפט הMaster

תהי $T(n)$ פונקציה המוגדרת על ידי:

$$T(n)=aT\left(\frac{n}{b}\right)+df(n)$$

$$T(1)=c$$

אזי אם f היא פונקציה כפלית (תוגדר להלן) התנהגות הפונקציה $T(n)$ תלויה ביחס בין a לבין $f(b)$ כמתואר להלן:

1. אם $a>f(b)$, $T(n)=\Theta\left(n^{\log_b a}\right)$.
2. אם $f(b)=a$, $T(n)=\Theta\left(n^{\log_b a}\log n\right)$.
3. אם $a<f(b)$, $T(n)=\Theta(f(n))$.

פתרון המשוואות הרקורסיביות

כעת נשים לב כי:

$$a^{\log_b n} = \left(b^{\log_b a}\right)^{\log_b n} =$$

$$= \left(b^{\log_b n}\right)^{\log_b a} = n^{\log_b a}$$

אם נציב שני ביטויים אלה ונקבל כי

$$T(n) = \underbrace{cn^{\log_b a}}_{*} + \underbrace{\sum_{i=0}^{(\log_b n)-1} a^i f\left(b^{\log_b n - i}\right)}_{**}$$

המחובר הראשון, $(*)$, מתנהג כמו $\Theta\left(n^{\log_b a}\right)$ וזהו פתרון הבעיה כאשר $f(n)=0$, אולם אין זה ברור כלל וכלל איך אפשר לטפל במחובר השני $(**)$.

פונקציות כפליות

כדי לטפל במחובר השני נניח כי f היא **פונקציה כפלית.**

פונקציה $f(x)$ היא **פונקציה כפלית** אם לכל x ו-y f מקיימת:

$.f(xy) = f(x)f(y)$

דוגמא: הפונקציה הפולינומית $f(x) = x^d$ (d קבוע) היא כפלית שכן

$$f(xy) = (xy)^d = x^d y^d = f(x)f(y)$$

שימו לב: באופן מעשי, רוב הפונקציות המתקבלות בשימוש במשפט האב הן כפליות.

הערות למשפט הMaster

1. **במקרה 1 -** רוב צעדי A מתבצעים על ידי הקריאות הרקורסיביות.
   במקרה 2 - רוב צעדי A מתבצעים מחוץ לקריאות הרקורסיביות.
   במקרה 3 - שני הגורמים הקודמים תורמים לסיבוכיות.
2. לערכי c ו-d אין שום השפעה על $.T(n)$
3. בספר מוצגת גרסה קצת שונה של המשפט, אשר אינה מניחה כי הפונקציה f היא כפלית. במקרה זה, הטיפול המתמטי בבעיה שונה, אולם ההשלכות המעשיות דומות.

הוכחת משפט הMaster

הוכחת משפט ה-Master מתקבלת על ידי פתרון המשוואות הרקורסיביות. אנו נטפל במשוואה פשוטה יותר:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

נציב בנוסחה את $T\left(\frac{n}{b}\right)$ ונקבל:

$$T(n) = a\left[aT\left(\frac{n}{b^2}\right) + f\left(\frac{n}{b}\right)\right] + f(n)$$

נפשט ונקבל:

$$T(n) = a^2T\left(\frac{n}{b^2}\right) + af\left(\frac{n}{b}\right) + f(n)$$

תכונות פונקציות כפליות

תהי f פונקציה כפלית כלשהי. אזי $f(1)$ מקיים:

$$f(1) = f(1\cdot 1) = f(1)\cdot f(1) = 1$$

נחשב כעת את ערך $f\left(\frac{a}{b}\right)$ עבור מנה כלשהי $\frac{a}{b}$:

$$f(1) = 1 = f\left(b\cdot\frac{1}{b}\right) = f(b)\cdot f\left(\frac{1}{b}\right) \Rightarrow$$

נחלק את שני האגפים ב $f(b)$ ונקבל:

$$f\left(\frac{1}{b}\right) = \frac{f(1)}{f(b)}$$

$$f\left(\frac{a}{b}\right) = f(a)\cdot f\left(\frac{1}{b}\right) = \frac{f(a)}{f(b)}$$

פתרון המשוואות הרקורסיביות

עבור פונקציה כפלית f

נניח כעת כי הפונקציה f היא **כפלית.** במקרה כזה נפתח את $(**)$ ונקבל:

$$(**) = \sum_{i=0}^{(\log_b n)-1} a^i f\left(\frac{b^{\log_b n}}{b^i}\right) =$$

$$= \sum_{i=0}^{(\log_b n)-1} a^i \frac{f\left(b^{\log_b n}\right)}{f\left(b^i\right)} =$$

$$= f\left(b^{\log_b n}\right) \sum_{i=0}^{(\log_b n)-1} \frac{a^i}{f(b)^i} =$$

$$= \left(f(b)\right)^{\log_b n} \sum_{i=0}^{(\log_b n)-1} \left(\frac{a}{f(b)}\right)^i =$$

וזהו טור גיאומטרי עם מנה $q = \frac{a}{f(b)}$

פתרון המשוואות הרקורסיביות

נמשיך להציב את $T\left(\frac{n}{b^2}\right)$ ונקבל:

$$T(n) = a^2\left[aT\left(\frac{n}{b^3}\right) + f\left(\frac{n}{b^2}\right)\right] +$$

$$af\left(\frac{n}{b}\right) + f(n))$$

נפשט ונקבל:

$$T(n) = a^3T\left(\frac{n}{b^3}\right) + a^2 f\left(\frac{n}{b^2}\right)$$

$$+ af\left(\frac{n}{b}\right) + f(n))$$

$$= a^3T\left(\frac{n}{b^3}\right) + \sum_{i=0}^{2} a^i f\left(\frac{n}{b^i}\right)$$

פתרון המשוואות הרקורסיביות

כדי שהרקורסיה תיגמר יש להגיע עד למקרה הקצה כלומר $T(1)$. כדי להשיג זאת, על הקריאה הרקורסיבית להתבצע לעומק i, המקיים $n = b^i$ כלומר $.i = \log_b n$

המשך ההצבה $\log_b n$ פעמים מניב:

$$T(n) = a^{\log_b n} T\left(\frac{n}{b^{\log_b n}}\right) + \sum_{i=0}^{(\log_b n)-1} a^i f\left(\frac{n}{b^i}\right)$$

אם נציב $n = b^{\log_b n}$ ונזכור כי $T(1) = c$ נקבל:

$$T(n) = a^{\log_b n}\, T(1) + \sum_{i=0}^{(\log_b n)-1} a^i f\left(\frac{b^{\log_b n}}{b^i}\right)$$

<u>פתרון המשוואות הרקורסיביות</u>

<u>עבור פונקציה כפלית f (המשך)</u>

אם נניח כי $a \neq f(b)$ ונשתמש בנוסחת הטור הגיאומטרי עבור $q = \frac{a}{f(b)}$ נקבל:

$$(**) = (f(b))^{\log_b n} \frac{\left(\frac{a}{f(b)}\right)^{\log_b n} - 1}{\frac{a}{f(b)} - 1} =$$

ולאחר פישוט נוסף נגיע ל

$$(**) = \frac{a^{\log_b n} - (f(b))^{\log_b n}}{\frac{a}{f(b)} - 1}$$

<u>פתרון המשוואות הרקורסיביות</u>

<u>עבור פונקציה כפלית f (המשך):</u>

נתבונן ב $(**)$

$$(**) = \frac{a^{\log_b n} - (f(b))^{\log_b n}}{\frac{a}{f(b)} - 1}$$

בביטוי זה, המכנה הוא קבוע ולכן ערכו של הביטוי תלוי ביחס בין a לבין $f(b)$ כמתואר בניתוח הבא:

<u>מקרה 1</u>

$a > f(b)$ - במקרה זה, ערך $(**)$ המונה מתנהג כמו $\Theta(a^{\log_b n})$ והמכנה הוא קבוע ולכן במקרה זה נקבל:

$$(**) = \Theta(a^{\log_b n}) = \Theta(n^{\log_b a})$$

מאחר שערך $(*)$ גם הוא $\Theta(n^{\log_b a})$ נקבל כי במקרה זה מתקיים:

$. T(n) = \Theta(n^{\log_b a})$

<u>מקרה 2</u>

$a < f(b)$ - במקרה זה, ערך המונה הוא $\Theta(f(b^{\log_b n})) = \Theta(f(n))$ וסימנו שלילי. מאחר שגם סימן המכנה הוא שלילי נקבל:

$$(**) = \Theta(f(b^{\log_b n})) = \Theta(f(n))$$

מאחר שערך $(*)$ הוא $\Theta(a^{\log_b n})$ ו $a < f(b)$, במקרה זה, ערך $(*)$ קטן מערך $(**)$ ומתקיים כי $. T(n) = \Theta(f(n))$

<u>מקרה 3</u>

$a = f(b)$

במקרה זה, נוסחת הטור הגיאומטרי אינה שמישה, שכן ערך המכנה בנוסחה הוא 0. אולם, במקרה זה מתקיים:

$$(**) = \sum_{i=0}^{(\log_b n)-1} a^i \frac{f(b^{\log_b n})}{f(b^i)} =$$

$$= \sum_{i=0}^{\log_b n - 1} f(n) = \log_b n \cdot f(n)$$

ומאחר שמתקיים:

$$f(n) = f(b^{\log_b n}) = (f(b))^{\log_b n} =$$

$$= a^{\log_b n} = n^{\log_b a}$$

מקבלים כי במקרה זה:

$$T(n) = \Theta(n^{\log_b a} \log_b n)$$

<u>פתרון המשוואות הרקורסיביות</u>

<u>עבור פונקציה כפלית f (סיכום):</u>

לכל פונקציה כפלית f, ערך הפונקציה T המוגדרת על ידי נוסחת הנסיגה

$$T(n) = aT\left(\frac{n}{b}\right) + df(n)$$

$$T(1) = c$$

הוא

1. עבור $a > f(b)$ $\Theta(n^{\log_b a})$.
2. עבור $a < f(b)$ $\Theta(f(n))$.
3. עבור $a = f(b)$ $\Theta(n^{\log_b a} \log n)$.

תרגיל: אפשר (וצריך) לבדוק כי הוספת הקבוע d אינה משנה את הפיתוח.

<u>דרכים לשיפור אלגוריתם הפרד ופתור</u>

יהי A אלגוריתם הפרד ופתור שסיבוכיותו מקיימת את מערכת המשוואות המופיעה במשפט האב. מהן הדרכים האפשריות לשיפור הסיבוכיות של A?

1. אם ש$a > f(b)$, נסה להקטין את $\log_b a$ על ידי הקטנת a או הגדלת b.
2. אם $a \leq f(b)$ נסה להקטין את סיבוכיות הפעולות שאינן כלולות בקריאות הרקורסיביות.
3. הקבועים c וd אינם מתבטאים בפתרון ולכן הקטנתם לא תשנה את הסיבוכיות.

<u>דוגמא - סיבוכיות מיון-מיזוג</u>

כזכור, מיון מיזוג מקיים:

$$T(n) = 2T\left(\frac{n}{2}\right) + dn$$

$$T(1) = c$$

מאחר ש $a = b = 2$ ו $f(n) = n$, אנו מקבלים כי

$$a = 2 = f(2) = f(b)$$

ומכאן, סיבוכיות מיון-מיזוג היא

$$T(n) = \Theta(n^{\log_2 2} \log_2 n) = \Theta(n \log n)$$

כעת, נציג את אלגוריתם Strassen למכפלת מטריצות מרובעות. סיבוכיות האלגוריתם תקבע תוך שימוש במשפט האב.

אלגוריתם Strassen

מכפלת מטריצות ריבועיות:

קלט: שתי מטריצות **ריבועיות** A ו B מסדר $n \times n$.

פלט: מכפלת הטריצות $C = A \times B$.

שימו לב: גודל הקלט הוא $O(n^2)$.

נפתח את ההרצאה בחזרה על האלגוריתם התקני ולאחר מכן נפתח את אלגוריתם Strassen. נסיים את ההרצאה בחישוב הסיבוכיות של האלגוריתם, שהיא $\Theta(n^{2.81\ldots})$, תוך שימוש במשפט האב.

מכפלת מטריצות ריבועיות

מכפלת המטריצות A ו B היא מטריצה C, המקיימת

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

ומסמנים $C = A \times B$.

מהי סיבוכיות בעיית חישוב מכפלת שתי מטריצות?

ברור כי $\Omega(n^2)$ היא חסם תחתון לסיבוכיות הבעיה (למה?).

אלגוריתם ישיר

האלגוריתם הישיר לחישוב C הוא:

for $i = 1$ **to** n **do**
  for $j = 1$ **to** n **do**
    $c_{ij} \leftarrow 0$
    for $k = 1$ **to** n **do**
      $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$
    end
  end
end

זמן הריצה: $\Theta(n^3)$.

בהמשך ההרצאה נפתח אלגוריתם יעיל יותר המשתמש בשיטת הפרד ופתור ומשיג זמן ריצה $\Theta(n^{2.81\ldots})$.

תצוגה מטריצית של כפל מטריצות

נפתח כעת אלגוריתם אחר, רקורסיבי, למכפלת מטריצות. נפתח על ידי הצגת מטריצה בעזרת 4 תת-מטריצות:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

המטריצה A_{11} היא תת המטריצה ברביע השמאלי והעליון במטריצה A, כמתואר בציור. המטריצות A_{12}, A_{21}, ו-A_{22} ממוקמות באופן אנלוגי.

תחת תצוגה זו, המכפלה $C = A \times B$ מתקבלת כמכפלת שתי מטריצות מסדר 2×2 שאבריהן הן תת מטריצות של מטריצות הקלט.

תצוגה מטריצית של כפל מטריצות

$$C = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \times \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} =$$

$$\begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix} =$$

$$= \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

נכונות האלגוריתם הזה מוכחת באופן ישיר. לדוגמא: נניח כי A_{ij} וגם B_{ij} הן מטריצות מסדר 2×2: במקרה זה מתקיים:

$$A_{11} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad A_{12} = \begin{pmatrix} a_{13} & a_{14} \\ a_{23} & a_{24} \end{pmatrix}$$

$$B_{11} = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \qquad B_{21} = \begin{pmatrix} b_{31} & b_{32} \\ b_{41} & b_{42} \end{pmatrix}$$

אם ננסה כעת לחשב את האיבר הראשון בשורה הראשונה של $A_{11}B_{11} + A_{12}B_{21}$ נקבל

$$(a_{11}b_{11} + a_{12}b_{21}) + (a_{13}b_{31} + a_{14}b_{41})$$

השווה בדיוק לאיבר הראשון בשורה הראשונה במכפלה $A \times B$. אם נמשיך כך אפשר להוכיח את נכונות הטענה כולה.

הערות

1. כל אחת מן המטריצות A_{ij}, B_{ij} ו-C_{ij}, $i,j = 1,2$ היא מסדר $\frac{n}{2} \times \frac{n}{2}$ ובכל מטריצה כזו יש $\frac{n^2}{4}$ איברים.
2. במקרה זה, פעולות החיבור והכפל הן **פעולות של מטריצות**. פעולות הכפל מחושבות על ידי קריאות רקורסיביות לאלגוריתם הכפל. הסימון × הושמט כדי לחסוך במקום. פעולות חיבור המטריצות משתמשות באלגוריתם הבא:

סכום מטריצות

תהיינה A וB מטריצות ריבועיות מסדר n (בגודל $n \times n$).

סכום המטריצות $D = A + B$ הוא מטריצה D המקיימת

$$d_{ij} = a_{ij} + b_{ij} \quad i,j = 1,2,\ldots,n$$

נתבונן בבעיית חישוב סכום מטריצות:

קלט: שתי מטריצות מסדר n.

פלט: סכום המטריצות.

טענה: סיבוכיות בעיית חישוב סכום מטריצות היא $\Theta(n^2)$.

הוכחה: כל אלגוריתם חייב לייצר את מטריצת הסכום שגודלה הוא n^2.

אלגוריתם הפרד ופתור (רקורסיבי) לחישוב מכפלת מטריצות בגודל $n \times n$

1. "רפד" את שתי המטריצות באפסים עד שמימדן יגיע לחזקה של 2 הגדולה מ n והקרובה אליו ביותר.
2. **אם** n=1 הכפל A בB (כפל שלמים)
3. **אחרת**
   3.1 חלק כל מטריצה לארבע.
   3.2 חשב את שמונה המכפלות הנדרשות (בעזרת קריאות רקורסיביות).
   3.3 בצע ארבע פעולות חיבור.
   3.4 מזג התוצאות למטריצה הנדרשת.

אנליזה של זמן החישוב

נסמן ב $T(n)$ את מספר הפעולות האריתמטיות, חיבור וכפל, הנדרשות להכפלת שתי מטריצות שגודלן $n \times n$.
מן האלגוריתם הרקורסיבי שהצגנו נובעות **משוואות הרקורסיה** הבאות:

$$T(n) = 8T\left(\frac{n}{2}\right) + 4\left(\frac{n}{2}\right)^2 =$$
$$8T\left(\frac{n}{2}\right) + n^2$$

ידוע גם כי $T(1) = 1$ כי מטריצה מסדר 1×1 היא סקלר ומכפלת שתי מטריצות כאלה דורשת פעולת כפל יחידה.

הצבה במשוואות הרקורסיה

נשתמש במשוואות הרקורסיה כדי לחשב:

$$T(2) = 8T(1) + 4 \cdot 1^2 = 12$$
$$T(4) = 8T(2) + 4^2 = 96 + 16 = 112$$

בדרך זו נוכל לחשב את מספר הצעדים הדרושים עבור זוג מטריצות מסדר n, לכל $n > 0$ אך לא נוכל לחשב את מספר הצעדים הזה **כפונקציה של** n.
כדי לחשב את סיבוכיות האלגוריתם, נשתמש במשפט האב ונקבל:

c=1, d=1, a=8, b=2, $f(x) = x^2$

במקרה זה, $f(2) = 4 < a$ ומקבלים

$$T(n) = \Theta\left(n^{\log_2 8}\right) = \Theta(n^3)$$

דיון

ברור כי $2n^3 + n^2 = \Theta(n^3)$, כלומר סיבוכיות האלגוריתם החדש שווה לסיבוכיות האלגוריתם הישיר. מה הרווחנו?

תשובה:
הרווחנו דרך חדשה להסתכלות על הבעיה. כעת אנו יכולים לחפש אלגוריתם רקורסיבי **יעיל יותר** כדי לצמצם את זמן החישוב הנדרש כדי לחשב את תת המטריצות C_{ij}, $i, j = 1,2$.

האלגוריתם של Strassen

מדען המחשבים השויצרי, Strassen, גילה שיטה חלופית להכפלת שתי מטריצות מגודל 2×2.

בשיטת Strassen מתבצעות שבע פעולות כפל מטריצות במקום שמונה פעולות כפל מטריצות המתבצעות בשיטה המקובלת. בתמורה עולה מספר פעולות החיבור.

הפעלת שיטת Strassen מניבה אלגוריתם להכפלת מטריצות שהסיבוכיות שלו היא:

$$\Theta\left(n^{\log_2 7}\right) = \Theta\left(n^{2.81\ldots}\right).$$

האלגוריתם של Strassen (המשך)

האינטואיציה: המקור לאיבר n^3 הן המכפלות הרקורסיביות. סיבוכיות החיבור היא n^2 בלבד. כדי להקטין את הסיבוכיות נוריד את מספר המכפלות ונעלה את מספר הסכומים.

הרעיון: כמו באלגוריתם הרקורסיבי המקורי נחשב את ארבע המטריצות C_{11}, C_{12}, C_{21} ו C_{22}.
כדי לחשב מטריצות אלה נחשב תחילה שבע מטריצות ביניים:

שבע מטריצות הביניים

כדי לחשב את המטריצות C_{11}, C_{12}, C_{21} ו C_{22}, נחשב תחילה, כשלב ביניים, את שבע המטריצות הבאות:

$$M_1 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$
$$M_2 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$$
$$M_3 = (A_{11} - A_{21}) \times (B_{11} + B_{22})$$
$$M_4 = (A_{11} + A_{12}) \times B_{22}$$
$$M_5 = A_{11} \times (B_{12} - B_{22})$$
$$M_6 = A_{22} \times (B_{21} - B_{11})$$
$$M_7 = (A_{21} + A_{22}) \times B_{11}$$

חוק הפילוג עבור מטריצות

לפני שנמשיך בהצגת האלגוריתם יש לציין כי צריך (וגם קל) לוודא כי **חוק הפילוג (דיסטריבוטיביות)** של הכפל מעל החיבור מתקיים גם עבור כפל וחיבור מטריצות כלומר: לכל שלוש מטריצות מסדר $n \times n$ A, B ו C מתקיים:

$$A \times (B + C) = A \times B + A \times C$$
$$(B + C) \times A = B \times A + C \times A$$

מדוע יש צורך בשני חוקי פילוג?

תרגיל:
שכנעו את עצמכם כי חוק הפילוג עבור כפל וחיבור מטריצות אכן מתקיים.

<u>האלגוריתם של Strassen (המשך):</u>

לאחר חישוב שבע מטריצות הביניים נשתמש בהן כדי לחשב את ארבעת המטריצות הסופיות כדלהלן:

$C_{11} = M_1 + M_2 - M_4 + M_6$

$C_{12} = M_4 + M_5$

$C_{21} = M_6 + M_7$

$C_{22} = M_2 - M_3 + M_5 - M_7$

נבדוק את הנוסחה הראשונה:

$$\begin{array}{ll} A_{12}B_{21} - A_{22}B_{21} + A_{12}B_{22} - A_{22}B_{22} & (= M_1) \\ + A_{11}B_{11} + A_{11}B_{22} + A_{22}B_{11} + A_{22}B_{22} & (= M_2) \\ - A_{11}B_{22} - A_{12}B_{22} & (= M_4) \\ + A_{22}B_{21} - A_{22}B_{11} & (= M_6) \\ = C_{11} & \end{array}$$

תרגיל: בדוק נכונות שאר הנוסחאות.

<u>סיבוכיות אלגוריתם Strassen:</u>

סיבוכיות הזמן נתונה על ידי המשוואה הרקורסיבית הבאה:

$$T(n) = 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2$$

$$T(1) = c$$

נשתמש במשפט האב ונקבל:

$f(n) = n^2$, $b = 2$, $a = 7$.

פרמטרים אלה מקיימים:

$$f(2) = 4 < a$$

ולכן

$$T(n) = \Theta\left(n^{\log_2 7}\right) = O(n^{2.81\ldots})$$

<u>סיכום</u>

בהרצאה זו למדנו את **משפט האב (Master Theorem)**, המאפשר לנו לקבוע סיבוכיות של מגוון רחב של אלגוריתמים רקורסיביים.

פתחנו את ההרצאה בהצגה פרמטרית של מערכת משוואות רקורסיבית כך שתשקף את סיבוכיות הזמן של אלגוריתמים רקורסיביים המקיימים דרישה מסוימת.

לאחר מכן הצגנו את משפט האב המאפשר חישוב הסיבוכיות של אלגוריתמים כאלה.

<u>סיכום (המשך)</u>

כדי להדגים את השיטה, טיפלנו בבעית כפל המטריצות הריבועיות מסדר n.

בתחילה הצגנו את האלגוריתם הישיר שסיבוכיותו היא $\Theta(n^3)$ (יש לזכור כי גודל הקלט הוא: $\Theta(n^2)$).

המשכנו בהצגת אלגוריתם הפרד ופתור פשוט פותר את הבעיה על ידי **חלוקת מטריצות הקלט ל4**. כתוצאה, הבעיה המקורית מחולקת ל**8 תת בעיות הנפתרות באופן רקורסיבי.**

<u>סיכום (המשך)</u>

האלגוריתם הזה, לא שיפר את סיבוכיות האלגוריתם המקורי אך פתח את הדלת ל**אלגוריתם** Strassen אשר שיפר את הסיבוכיות על ידי מציאת דרך מתוחכמת ל**חישוב 8 תוצאות המכפלות הנדרשות** על ידי ביצוע בפועל של **7 מכפלות בלבד.**

מספר הפעמים בהן חישבנו **סכום של מטריצות עלה מ 4 ל 18**. ניתוח סיבוכיות הזמן של אלגוריתם Strassen, התקבל כתוצאה מיידית של משפט האב (Master Theorem).

סיבוכיות אלגוריתם Strassen היא $\Theta(n^{2.81\ldots})$.

<u>סיכום (המשך)</u>

אלגוריתם Strassen מראה את הדרך לשיפורים נוספים על ידי הורדת מספר המכפלות הנדרש והעלאת מספר הסכומים.

היום משתמשים בשיטות דומות אך מסובכות יותר, מחלקים כל בעיה למספר רב יותר של חלקים, במקום 4 בלבד, ומגיעים לאלגוריתמים למכפלת מטריצות בסיבוכיות זמן $O(n^{2.3\ldots})$.

שימו לב: אלה אלגוריתמים בעלי **ערך תיאורטי בלבד**, שכן **קבוע הפרופורציה** עולה לערכים בלתי אפשריים.

<u>סיכום (המשך)</u>

מה צריך לזכור מהרצאה זו?

1. פיתוח מערכת המשוואות הרקורסיביות.
2. משפט האב ושימושיו.
3. להבין את הוכחת המשפט.
4. הגדרת כפל מטריצות והאלגוריתם הישיר.
5. האלגוריתם הרקורסיבי הפשוט.
6. רעיון אלגוריתם Strassen (אין צורך לזכור את מטריצות הביניים ואת הפיתוח בעל פה.

<u>נספח: חישוב מדויק של סיבוכיות האלגוריתם הרקורסיבי הראשון</u>

<u>מציאת נוסחה סגורה</u>

כדי לחשב את סיבוכיות האלגוריתם, עלינו למצוא **נוסחה סגורה** לחישוב $T(n)$.

טענה: נתון

$$T(n) = 8T\left(\frac{n}{2}\right) + n^2,\ T(1) = 1$$

אזי (ניחוש): לכל $n = 2^i$

$$T(n) = 2n^3 - n^2$$

<u>הוכחה</u>

עבור $n = 2^i$ ההוכחה באינדוקציה על i. כלומר, יש להניח נכונות לגבי n ולהוכיח לגבי $2n$. נשתמש בנוסחה הרקורסיבית לחישוב $T(2n)$ בעזרת $T(n)$:

$$T(2n) = 8T(n) + (2n)^2 =$$

בתוך נוסחה זו נציב את פתרוננו עבור $T(n)$:

$$T(2n) = 8 \cdot (2n^3 - n^2) + 4n^2$$

ונפשט כדי לקבל:

$$= 16n^3 - 4n^2 =$$

$$= 2(2n)^3 - (2n)^2$$

מש"ל

הרצאה 11: אלגוריתם למציאת החציון - Order Statistic

Definition

In statistics, the k-th **order statistic** of a statistical sample is equal to its kth-smallest value.

או בעברית:

בסטטיסטיקה, האיבר ה-k בגודלו מלמטה במדגם נקרא

The k-th order statistics

בהרצאה זו נציג אלגוריתם יעיל לחישוב האיבר הזה, כפי שמוגדר בשקף הבא:

הבעיה

קלט: 1. מערך A בן n איברים **נבדלים** השייכים לתחום סדור.

2. מס׳ שלם k, $1 \le k \le n$.

פלט: האיבר ה - k בגודלו (מלמטה) ב -A.

דוגמא

קלט $k = 3$

A

3	5	6	4	2	8	7	1

פלט 3

אלגוריתם נאיבי

מיין את A והחזר את האיבר ה - k בוקטור הממוין.

נכונות

מיידית מהגדרת הבעיה.

סיבוכיות

$\Theta(n\log n)$

שאלה: האם קיים אלגוריתם יעיל יותר?

תשובה: להלן נציג אלגוריתם בזמן ריצה לינארי למציאת האיבר ה - k בגודלו ב -A.

סקירת האלגוריתם

להלן סקירת שלבי האלגוריתם:

1. בחר ציר ״חכם״.
2. בצע חלוקת ציר לתת מערך תחתון (שנסמנו ב-$A_<$) ולתת מערך עליון (שנסמנו ב-$A_\ge$).
   תזכורת: כל איבר ב-$A_\ge$ גדול ממש מכל איבר ב-$A_<$.
3. זהה את תת המערך המכיל את האיבר המבוקש.
4. המשך את החיפוש בתת המערך המכיל את האיבר המבוקש. (קריאה רקורסיבית).

תכונות חלוקת ציר

נניח שמתקיים $|A_<| = m$ ונניח גם שאנו מחפשים את האיבר ה-k בגודלו שנסמנו ב-m_k.

לאחר ביצוע חלוקת הציר עלינו לענות על השאלה הבאה:

באיזה תת מערך נמצא m_k ?

התשובה תלויה ביחס בין m ל k כמוכח בטענה הבאה:

למה 11.1

יהי A מערך מעל תחום סדור. נניח כי ב -A בוצעה חלוקת ציר שהניבה את החלקים $A_<$ בגודל m ו - $A_\ge$ בגודל $n-m$. לאחר חלוקת הציר מתקיים:

1. אם $k \le m$ אזי m_k הוא האיבר ה-k בגודלו ב-$A_<$.
2. אם $k > m$ אזי m_k הוא האיבר ה-$(k-m)$ בגודלו ב - $A_\ge$.

דוגמא

A

3	8	-2	13	11	6	9	2	14	7

$pivot = 8$, $k = 6$

$A_<$

3	-2	6	2	7

$A_\ge$

8	13	11	9	14

$m = 5$

ב $A_<$ נמצאים 5 האיברים הקטנים ביותר ב A.

האיבר ה 6 בגודלו בA, 8, הוא האיבר המינימלי ב$A_\ge$

הוכחת הלמה

ב - A יש k-1 איברים, הקטנים מ - m_k. כל שאר איברי A גדולים או שווים מ - m_k. לאחר שמתבצעת חלוקת הציר, ב - $A_<$ נמצאים m האיברים הקטנים ביותר ב-A, מסודרים בסדר שרירותי.

הוכחת סעיף 1

אם $k \le m$ אזי ברור ש-m_k הוא בין הוא בין m האיברים הקטנים ביותר במערך A, ולכן הוא מאוחסן בתת המארך התחתון.

הוכחת סעיף 2:

אם, לעומת זאת, $k > m$ אזי איברי $A_<$ הם m האיברים הקטנים ביותר במערך A, ו-m_k הוא האיבר ה $k-m$ בגודלו ב - $A_\ge$.

מש"ל

תנאי הסיום: אם $n \le 10$ נחפש את האיבר המבוקש באופן סדרתי.

תנאי הקצה

ידוע כי שיטות רקורסיביות אינן יעילות עבור קלט בגדלים קטנים. כדי להגדיל את היעילות, בגורם קבוע בלבד, נקבע כי אם מספר איברי מערך הקלט קטן או שווה מ-10. נקרא לשיטה *Find* להלן:

```
Find(A,k)
A ← Bubblesort(A)
return(A[k])
```

אלגוריתם 11.1: השיטה Find

האלגוריתם

להלן קוד האלגוריתם *Select* המחשב את האיבר ה-k בגודלו במערך הקלט A:

```
Select(A,k)
if |A| ≤ 10 then
    return Find(A,k)
p ← WiseChoosePivot(A)
Partition A into A_< and A_≥ A_<
if k ≤ m return Select(A_<,k)
if k > m return
        Select(A_≥,k−m)
```

אלגוריתם 11.2: השיטה Select

נכונות

נכונות האלגוריתם נובעת מיידית מן הטענה שהוכחנו.

סיבוכיות האלגוריתם – איכות איבר הציר

סיבוכיות השיטה *Find* אינה תלויה בגודל הקלט n, ולכן היא לא משפיעה על סיבוכיות האלגוריתם כולו.

סיבוכיות החלוקה היא כמובן לינארית.

הגורם היחיד המשפיע על סיבוכיות אלגוריתם *Select* הוא הגודל היחסי של תת המערכים $A_<$ ו-$A_\ge$. גודל זה נקבע ישירות על ידי **איכות איבר הציר**.

איכות איבר הציר (המשך)

אם לדוגמא נשתמש בשיטה *Choosepivot* שהשתמשנו בה במיון מהיר, אזי במקרה הגרוע ביותר, אחד מתת המערכים יכיל איבר יחיד.

אם תופעה זו תחזור על עצמה (כפי שקורה בדוגמאות מסויימות), סיבוכיות האלגוריתם תגיע ל $\Theta(n^2)$.

כדי להתגבר על בעיה זו, נשקיע עבודה **לינארית** בבחירת איבר הציר וכתוצאה נקבל שסיבוכיות המקרה הגרוע ביותר באלגוריתם כולו תהיה גם היא **לינארית**.

בחירת איבר ציר איכותי

אנו מחפשים איבר ציר שיקיים: מספר האיברים בכל אחד מתת המערכים $A_<$ ו-$A_\ge$ יהיה לפחות αn ולכל היותר ה $(1-\alpha)n$, כאשר α הוא קבוע שאינו תלוי ב-n ואשר מקיים $0<\alpha<1$.

האלגוריתם שנציג מבטיח בחירה של ציר שמרחקו מכל אחד משני קצות מערך הקלט הוא לפחות $3/10n$. מכאן נקבל כי גדלו של כל אחד מתת המערכים יהיה לכל היותר $7/10n$, כלומר:

$$3/10n < |A_<| < 7/10n$$

$$3/10n < |A_\ge| < 7/10n$$

תאור האלגוריתם לבחירת הציר

יהי A מערך עם n איברים. **החציון** של A הוא האיבר ה$\lfloor n/2 \rfloor+1$ בגודלו ב-A.

האלגוריתם פועל כך:

1.נחלק את המערך A ל-$\lceil n/5 \rceil$ קבוצות בנות 5 איברים כל אחת. בכל חמישיה, נמצא את האיבר השלישי בגודלו, כלומר את החציון של כל חמישייה כזו. (זמן החישוב הכולל בסעיף זה הוא $\Theta(n)$).

תאור האלגוריתם לבחירת הציר (המשך)

2.תהי M קבוצת החציונים שמצאנו ויהי m החציון של M. m **הוא הציר שנבחר**.

3.כדי לחשב את m, נבצע קריאה רקורסיבית:

$Select(M, \lfloor |M|/2 \rfloor + 1)$

(זמן החישוב $T(n/5)$).

הקוד מופיע בשקף הבא:

השיטה WiseChoosePivot

```
WiseChoosePivot(A)
for i = 1 to ⌈n/5⌉
/* בחר את החמישיה ה-i */
  for j = 1 to 5 do
    B[j] ← A[(i−1)*5 + j]
  endfor
  B ← sort(B)
  /* בחר את חציון החמישיה ה-i */
  M[i] ← B[3]
Return Select(M, ⌊|M|/10⌋+1)
```

אלגוריתם 11.2: *WiseChoosePivot*

אלגוריתמים א' הרצאה מס' 11 – פרופ' עמוס ישראלי

דוגמא - הוקטור A ממוין בחמישיות

35	46	72	77	90
12	17	19	21	23
50	69	70	71	125
10	18	22	128	256
-4	25	40	41	42
10	11	560	580	600
7	13	199	201	326
38	39	61	147	269
-100	-66	52	421	856
500	601	702	803	904
62	63	$\boxed{64}$	65	66
-120	-60	-40	200	400

אלגוריתמים א' הרצאה מס' 11 – פרופ' עמוס ישראלי

דוגמא (המשך)

הוקטור M

72	19	70	22	40	560	199	61	52	64	702

החציון של M הוא $m = 64$.

שימו לב:

1. מספר האיברים ב M הוא כמספר החמישיות כלומר $\left\lceil \frac{n}{5} \right\rceil$.
2. מספר איברי M הקטנים או שווים ל m הוא $\left\lceil \frac{n}{10} \right\rceil$.

אלגוריתמים א' הרצאה מס' 11 – פרופ' עמוס ישראלי

דוגמא (המשך)

3. כל האיברים הנמצאים משמאל לאיבר הקטן מ-m גם הם קטנים מ-m ולכן, ב-A יש $\left\lceil \frac{3n}{10} \right\rceil$ איברים הקטנים מ m.
4. מספר איברי M הגדולים או שווים מ m הוא $\left\lceil \frac{n}{10} \right\rceil$.
5. כל האיברים הנמצאים מימין לאיבר הגדול מm גם הם גדולים מ m ולכן, בA יש $\left\lceil \frac{3n}{10} \right\rceil$ איברים הגדולים מ m.

אלגוריתמים א' הרצאה מס' 11 – פרופ' עמוס ישראלי

חישוב α

טענה: יהי m החציון של הוקטור M. מספר איברי A הקטנים ממש מ-m הוא לפחות $\left\lceil \frac{3}{10} \right\rceil n$.

מספר איברי הוקטור A הגדולים ממש מ-m הוא לפחות $\left\lceil \frac{3}{10} \right\rceil n$.

שימו לב

מטענה זו נובע מיידית

$$|A_{\ge p}| \le \left\lceil \frac{7}{10} \right\rceil n \quad |A_{<p}| \le \left\lceil \frac{7}{10} \right\rceil n$$

כלומר, גודל הקלט בקריאה הרקורסיבית, הוא **לכל היותר** 7/10 מגודל הקלט המקורי.

אלגוריתמים א' הרצאה מס' 11 – פרופ' עמוס ישראלי

הוכחה

מספר האיברים ב M הוא $\lceil n/5 \rceil$.

מכאן נובע ש-m גדול מ $n/10$ מאיברי M. כל אחד מאיברי M, m_i, קטן משני האיברים בחמישיה "שלו".

מכאן נובע כי m גדול או שווה מ $3n/10$ מאיברי A.

באופן סימטרי אפשר להוכיח כי m קטן או שווה מ $3n/10$ מאיברי A.

מש"ל

אלגוריתמים א' הרצאה מס' 11 – פרופ' עמוס ישראלי

סיבוכיות האלגוריתם

משפט

סיבוכיות האלגוריתם למציאת האיבר ה k היא לינארית.

הוכחה

נסמן ב $T(n)$ את זמן הריצה של האלגוריתם על מערך בגודל n.

לאלגוריתם שני קטעים בעלי סיבוכיות לינארית:

1. בחירת המערך M.
2. ביצוע חלוקת הציר.

נסמן ב cn את מספר הצעדים הכולל, הדרוש לביצוע שני חלקים אלה.

אלגוריתמים א' הרצאה מס' 11 – פרופ' עמוס ישראלי

המשך ההוכחה

בנוסף לכך, במהלך החישוב מתבצעות שתי קריאות רקורסיביות לשגרה *Select*:

1. למציאת m החציון של M. הסיבוכיות היא, $T(|M|) = T\left(\frac{n}{5}\right)$.
2. להמשך הפתרון בתת הבעיה שנמצאה: במקרה הגרוע ביותר, הסיבוכיות היא, $T\left(\max\left|A_{<p}\right|,\left|A_{\ge p}\right|\right) = T\left(\frac{7n}{10}\right)$.

המשוואות המתקבלות

המשוואות הרקורסיביות המתקבלות הן :

$$T(10) = const$$

$$T(n) = \underbrace{T\left(\frac{n}{5}\right)}_{\text{בחירת ציר}} + \underbrace{T\left(\frac{7n}{10}\right)}_{\text{קריאה רקורסיבית}} + \underbrace{cn}_{\text{חלוקה}}$$

עלינו להוכיח : $T(n) = \Theta(n)$.

ברור כי $T(n) > n$. לכן, מספיק להראות כי $T(n) < dn$ עבור קבוע כלשהו d, $d > 0$.

הוכחה

נוכיח באינדוקציה על n כי מערכת המשוואות שהצבנו מקיימת $T(n) < 10cn$.

הוכחה: באינדוקציה על n.

בסיס: $T(10) = const$ וכן המשפט מתקיים.

צעד: נניח כי לכל $m < n$ הטענה מתקיימת, כלומר $T(m) < 10cm$.

בפרט מתקיים :

(1)

$$(2)T(|M|) = T\left(\frac{n}{5}\right) \le \frac{10cn}{5} = 2cn$$

הוכחה (המשך)

(2)

$$t(\max|A_{<p}|,|A_{\ge p}|) \le T\left(\frac{7n}{10}\right) \le \frac{7 \cdot 10cn}{10}$$

$$\le t(\max|A_{<p}|,|A_{\ge p}|) \le 7cn$$

נציב תוצאות אלה במשוואה ונקבל :

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + cn \le$$

$$\le 2cn + 7cn + cn = 10cn$$

מש"ל

חשוב לציין

אפשר להוכיח כי בחירת ציר בהסתברות שווה מבין כל איברי המערך תניב גם היא אלגוריתם שהסיבוכיות הממוצעת שלו היא $\Theta(n)$, אך במקרה זה ייתכנו ריצות בזמן גדול יותר (ובהסתברות נמוכה).

בחירת הציר בדרך המוצעת מבטיחה כי סיבוכיות האלגוריתם היא לינארית עבור כל קלט וקלט.

תוצאה זו לא תתקבל "חינם".

התשלום הוא הגדלת הסיבוכיות הממוצעת בגורם קבוע.

במלים אחרות : הסיבוכיות הממוצעת בשני האלגוריתמים היא **לינארית** אך

באלגוריתם הדטרמיניסטי, המקדם גדול יותר.

נספח

להלן נתאר סדרת קריאות אפשרית בביצוע האלגוריתם :

דוגמא

נניח כי מחפשים את האיבר ה - 70 במערך A בן 200 איברים. כדי לבצע זאת מפעילים את הקריאה $Select(A,70)$. בשקף הבא, אנו מפורטות (חלק מן הקריאות הרקורסיביות שתתבצענה.

אנו מניחים כי $Find$ היא פרוצדורה פשוטה למציאת האיבר הi בגדלו במערך שבו לכל היותר 10 איברים.

בסוגרים רשומות הנחות לגבי גודל המערכים המתקבלים בחלוקת ציר.

פרוט הקריאות

$Select(A,70)$

$ChoosePivot$

compute M_1, ($|M_1| = 40$)

$Select(M_1,21)$

* $Choosepivot$

* compute M_2, ($|M_2| = 8$)

* $m_2 \leftarrow Find$ median of M_2

* $M_{11} \leftarrow M_{1,<m_2}$, ($|M_{11}| = 12$)

* $M_{12} \leftarrow M_{1,\ge m_2}$, ($|M_{12}| = 28$)

* $Select(M_{12},9)$

$m_1 \leftarrow Find^*$ 9th elt of M_{12}

$A_1 \leftarrow A_{<m_1}$, ($|A_1| = 140$)

$A_2 \leftarrow A_{\ge m_1}$, ($|A_2| = 60$)

תיאור הקריאות בדוגמא

כדי לבצע זאת מפעילים את הקריאה

$Select(A,70)$

במהלך קריאה זו מתבצעות הפעולות הבאות :

1. בוחרים מערך M_1 בן 40 איברים.
2. מפעילים באופן רקורסיבי את הקריאה $m_1 \leftarrow Select(M_1,21)$ לאחר מציאת m_1 כמפורט בשקף הבא מתבצע :
3. מחלקים את A בחלוקת ציר לפי m_1.

תיאור הפעולות במהלך $Select(M_1,21)$

2.1 בוחרים מערך M_2 בן 8 איברים.

2.2 מפעילים שגרה פשוטה למציאת m_2, החציון (האיבר ה5) של M_2.

2.3 מחלקים את M_1 לפי m_2 כציר. יהיו M_{11} ו M_{12} החלק התחתון והחלק העליון בחלוקת M_1 לפי m_2. לפי המשפט שהוכחנו, מובטח לנו כי $12 \le |M_{11}| \le 28$ ו $12 \le |M_{12}| \le 28$. נניח כי $|M_{11}| = 12$ ו $|M_{12}| = 28$.

2.4 מפעילים את $Select(M_{12},9)$. לאחר קריאה רקורסיבית נוספת מוצאים את m_1, האיבר ה - 9 ב - M_{12} והחציון של M_1.

דוגמא - סיכום

לפי המשפט שהוכחנו מובטח לנו כי לאחר חלוקת מערך הקלט A לפי m_1, מובטח לנו כי מספר האיברים בכל אחד מחלקי A יהיה גדול מ - $60 = \frac{3}{10}200$.

במקרה הגרוע ביותר, נקבל חלוקה לחלק תחתון עם 140 איברים ולחלק עליון עם 60 איברים. במקרה זה נקרא כעת ל $Select(A_{<p},70)$.

הרצאה 12: אלגוריתמים חמדניים

אלגוריתמים חמדניים (Greedy)

החמדנות היא שיטה (Paradigm) חשובה בתורת האלגוריתמים.

באופן כללי ביותר, השיטה החמדנית מציעה כי בכל שלב, כאשר יש לבחור בין כמה צעדים חלופיים בדרך לפתרון בעיה כלשהי, נבחר בצעד המביא **רווח מיידי מכסימלי.**

בדרך זו נבנים אלגוריתמים פשוטים יחסית, אך לא תמיד אלגוריתמים אלה מניבים פתרונים אופטימליים. במצב זה, הקושי העיקרי בשיטה הוא לבדוק האם האסטרטגיה החמדנית מניבה אלגוריתם הפותר את הבעיה.

דחיסת מידע

בקבצי טקסט משתמשים כיום בקידוד ששמו Unicode. בקידוד זה, כל תו מקודד במספר קבוע של סיביות. מספר זה יכול להיות 8 (קוד UTF8), 16 (UTF16), או 32 (UTF32).

כאשר כל תו מקודד במספר סיביות שווה, אין התחשבות בשכיחות המופעים של כל תו בטקסט נתון. אם נתחשב בשכיחות זו, בטקסט נתון כלשהו, נוכל לחסוך במספר הסיביות הכולל הנדרש לקידוד הטקסט הנתון.

דוגמא לדחיסת מידע (מן הספר CLR)

נניח כי אנו רוצים לאחסן קובץ ובו 100 מלים מעל א"ב ובו 6 תווים. שכיחויות התווים מתוארות בטבלה הבאה:

התו	a	b	c	d	e	f
השכיחות	45	13	12	16	9	5
שווה אורך	000	001	010	011	100	101
משתנה	0	101	100	111	1101	1100

בשורת הבאה השלישית מופיע **קוד שווה אורך** (**Fixed-length Codeword**), ובשורה שאחריה מתואר **קוד באורך משתנה** (**Variable-lebgth codeword**).

המשך הדוגמא לדחיסת מידע

חישוב קצר מעלה כי אם בקובץ יש 100 מלים, אז הקידוד הראשון דורש 300 סיביות, אך הקידוד השני דורש 224 סיביות.

דוגמא זו מבהירה מייד את חשיבות הבחירה בקידוד יעיל לדחיסת מידע.

קידוד מחרוזות

תהי S מחרוזת תווים מעל א"ב נתון C

קידוד בינארי מעל C, הוא קידוד בו כל תו $c \in C$ מקודד על ידי סדרת סיביות. הסדרה המקודדת את התו c נקראת **הקידוד של** c ומסומנת על ידי על ידי $D(c)$.

הקידוד של S על ידי D, $D(S)$, מתקבל על ידי שרשור הקידודים של התווים ב-S.

עלות של הקידוד של S, שווה למספר הסיביות ב-$D(S)$ ומסומן עלך ידי $|D(S)|$

קידוד פריפיקס (Prefix Code)

קידוד פריפיקס - הוא קידוד בינארי בו לכל שני תווים c_1 ו-c_2, $D(c_1)$ אינו תחילית (prefix) של $D(c_2)$או במלים:

הקידוד של c_1 אינו תחילית של הקידוד של c_2.

שימו לב: כל קוד שווה אורך הוא קוד פריפיקס (הסבירו זאת).

שימוש בקידוד פריפיקס

1. **קידוד** – כדי לקודד מחרוזת, מאתחלים את המחרוזת המקודדת כמילה ריקה ומשרשרים את קידודי התוים במחרוזת המקודדת בזה אחרי זה.
2. **פיענוח** – אפשר להשתמשי בעץ המייצג את הקוד, כדי לפענח את תוי המחרוזת המקודדת בזה אחר זה. מאחר שהקוד הוא קוד תחילי, אין חשש לטעות.

עלות של קידוד

תהי S מחרוזת כלשהי מעל א"ב C, ויהי $D = D(C)$ קידוד של C. העלות של קידוד המחרוזת S באמצעות הקידוד D, מוגדרת להיות מספר הסיביות ב-$D(S)$. כמוגדר להלן:

תהי S מחרוזת תווים מעל א"ב C. לכל תו $c \in C$ נסמן $h[c]$ את מספר המופעים של התו c במחרוזת S. נגדיר את **העלות** של S, ביחס לקוד D, על ידי:

$$|D(S)| = \sum_{c \in C} h[c] D(c)$$

<u>קוד אופטימלי</u>

קוד פריפיקס D הוא **<u>אופטימלי ביחס ל-</u>** S, אם לא קיים קידוד פריפיקס D' כך ש $|D(S)|>|D'(S)|$.

בהמשך ההרצאה נציג את אלגוריתם האפמן (Huffman) לחישוב קידוד פריפיקס אופטימלי.

<u>הצגת קידוד פריפיקס כעץ בינארי</u>

אפשר להציג כל קוד פריפיקס כעץ בינארי בו כל תו מקודד על ידי מסלול בעץ כמודגם באיור הבא:

<u>עלות של עץ קידוד</u>

יהי T עץ קידוד של קוד פריפיקס עבור קבוצת תווים C. מאחר שמספר הסיביות בקידוד של תו $c \in C$, זהה לאורך המסלול מן השורש של T, אל העלה של c. נסמן את אורך המסלול משורש העץ T אל העלה c, ב-$l(c)$, ואת קידוד S על ידי העץ T ב-$T(S)$, נקבל:

$$|T(S)|=\sum_{c\in C} l(c)f[c]$$

<u>אלגוריתם הפמן</u>

אלגוריתם הפמן מקבל כקלט מערך זוגות D, כאשר כל זוג ב-D, מכיל תו ושכיחות התו במחרוזת נתונה. האלגוריתם מחשב עץ קידוד פריפיקס אופטימלי עבור הזוגות שבמערך D.

<u>אלגוריתם הפמן – הקוד</u>

```
Huffman(D)
n ← |D|
Q ← D
for i ← 1 to n − 1
   z ← new Node()
   x ← Q.extractMin()
   z.left ← x
   y ← Q.extractMin()
   z.right ← y
   f[z] ← f[x] + f[y]
   Q.inser(z)
end_for
return Q.extractMin()
```

<u>הוכחת נכונות</u>

כעת עלינו להוכיח שהקוד המיוצג על ידי העץ שהאלגוריתם מחשב הוא אופטימלי, או במלים אחרות: לא קיים קוד פריפיקס המקודד את התווים הנתונים, בשכיחויות הנתונות בפחות סיביות. טענה זו נובעת משתי הלמות הבאות:

<u>למה 1</u>

תהי S מחרוזת מעל קבוצת תווים C. לכל תו $c \in C$ תהי $f[c]$ שכיחות התו c במחרוזת S, ויהיו $x, y \in C$ שני התווים בעלי שכיחות מינימלית ב-S. בתנאים אלה קיים קוד פריפיקס אופטימלי ל-C בהם התווים x ו-y מקודדים על ידי שתי מלים שאורכן שווה הנבדלות אך ורק בסיבית האחרונה.

<u>הערה:</u> שימו לב כי בעץ הקוד האופטימלי הזה, התווים x ו-y הם "אחים".

<u>הוכחה</u>

יהי T עץ קידוד של קוד אופטימלי כלשהו עבור S, ויהיו a ו- b, שני עלים "אחים" בעץ T.

קל לראות כי אם נחליף את קידוד התווים a ו- b עם הקידוד של x ו-y, עלות הקוד לא תגדל וייתכן אף שתקטן.

<u>למה 2</u>

תהי S מחרוזת מעל קבוצת תווים C.
לכל תו $c \in C$ תהי $f[c]$ שכיחות התו c במחרוזת S. יהי T עץ קידוד של קוד פריפיקס אופטימלי ל-C, ויהיו $x, y \in C$ שני תווים אחים בעץ הקידוד T. יהי z תו נוסף ששכיחותו היא $f[z] = f[x] + f[y]$ ויהי T' עץ הקידוד המתקבל מן העץ T, על ידי החלפת האב של העלים x ו-y בעלה z.
בתנאים אלה העץ T' מהווה עץ קידוד אופטימלי עבור הא״ב $C' = C - \{x, y\} + z$.

<u>הוכחה</u>

נניח בשלילה כי T' אינו עץ קידוד אופטימלי עבור הא״ב המתואר, ויהי T'' עץ קידוד אופטימלי עבור הא״ב הזה. קל לראות כי עלות העץ המתקבל מ-T'' על ידי החלפת העלה המתאים לתו z בשני האחים x ו-y קטנה יותר מעלות העץ T, בסתירה לאופטימליות של T.

מש״ל

<u>משפט</u>

אלגוריתם הפמן מחשב עץ קידוד אופטימלי עבור מערך הזוגות D שבקלט.

<u>הוכחה</u>

יהיו $x, y \in C$, שני התווים בעלי השכיחות המינימלית ב-D.
לפי למה 1, קיים עץ קידוד אופטימלי ל-D, שבו התוים x ו-y הם אחים.
לפי למה 2, העץ T' המתקבל מ-T, על ידי צמצום הצמתים המתאימים לתוים x ו-y, מהווה עץ קידוד אופטימלי ל א-ב המתקבל מהחלפת

התוים x ו-y בתו z, בשכיחות $f[x] + f[y]$.

מש״ל

חרטול

אלטוריתקן

תרגיל -1-

פסודו:

```
for i ← 2 to N
    if A[i] > A[1] then return(i) **
end (for)
return (-1) ***
```

טענת נכונות:

מחזיר את האינדקס הראשון של המספר הגדול מ $A[1]$. אם אין מספר כזה יוחזר -1.

שמורה:

המספרים במקומות הקטנים שווים ל $A[1]$. $A[i-1] \le A[1]$

הוכחת השמורה:

אינדוקציה על i:

בדיקה: עבור $i=2$ המספרים במקומות $1..1$ קטנים שווים ל $A[1]$.

נניח שהשמורה שווה ל $i=k$. כלומר* המספרים במקומות $1...k-1$ קטנים שווים ל $A[1]$. ואם הגענו לשלב $i=k+1$ התנאי לא יתקיים, כי אז היינו יוצאים מהלולאה. כלומר $A[k] \le A[1]$, ביחד עם * נקבל שהמספרים במקומות $1...k$ קטנים שווים ל $A[1]$, לפי השמורה. מש"ל.

הוכחת נכונות:

נניח הוחזר $i=k$; $k \neq -1$ כלומר הוא הוחזר ב**. כלומר התקיים $A[k] > A[1]$

לפני כן התקיימה השמורה: כל המספרים במקומות $1...k-1 \le A[1]$.

כלומר הוחזר המספר הראשון הגדול מ $A[1]$.

נניח שהוחזר -1: כלומר הוחזר ב ***, התנאי לא יתקיים $A[N] \le A[1]$

לפני כן התקיימה השמורה, כלומר: כל המספרים במקומות $1...N-1 \le A[1]$.

כלומר, כל מספרי המערך $\le A[1]$. כלומר אין מספר הגדול מ $A[1]$.

$$1+2+3+4+\dots+n=\frac{n(n+1)}{2} \Rightarrow \Theta(n^2)$$

$$1^2+2^2+3^2+\dots+n^2=\frac{n(n+1)(2n+1)}{6} \Rightarrow \Theta(n^3)$$

$$1^3+2^3+3^3+\dots+n^3=\frac{n^2(n+1)^2}{4} \Rightarrow \Theta(n^4)$$

⇐ נוסחה כללית:

$$\boxed{f(n)=1^k+2^k+3^k+\dots+n^k \Rightarrow \Theta(n^{k+1})}$$

הוכחה:

$$1^k+2^k+3^k+\dots+n^k \le n^k+n^k+\dots+n^k \le n\cdot n^k = n^{k+1}$$

כלומר $f(n)=O(g(n))$

$$1^k+2^k+3^k+\dots+n^k \ge \left(\frac{n}{2}\right)^k+\left(\frac{n+1}{2}\right)^k+\dots+n^k \ge \underbrace{\left(\frac{n}{2}\right)^k+\left(\frac{n}{2}\right)^k+\dots+\left(\frac{n}{2}\right)^k}_{\frac{n}{2}\text{ פעמים}}$$

$$\ge \frac{n}{2}\left(\frac{n}{2}\right)^k \Rightarrow \left(\frac{n}{2}\right)^{k+1} = \frac{n^{k+1}}{2^{k+1}}$$

$$C=\frac{1}{2^{k+1}}$$

$$f(n)\ge c\cdot g(n)$$

לכן $f(n)=\Omega(n^{k+1})$

$$\Rightarrow f(n)=\Theta(n^{k+1})$$

תרגיל
-2-

שאלה - מה מבצע האלגוריתם הבא?

```
quit (A,n)
for i←1 to n
    X← A[n]
    XInd← n
    for j← i to n
        if A[j] > X
            X← A[j]
            Xind← j
 *  ───→ endif
    end for
** ───→ SWAP(A, i, Xind)
end for
```

האלגוריתם quit מקבל כקלט מערך A, ומספר טבעי n, וממיין את A מהגדול לקטן.

כתבו <u>שמורה עבור מיקום *</u>:

$X = Max\{A[i]...A[j], A[n]\}$

וב Xind המיקום שלו במערך.

<u>הוכחת השמורה:</u>

<u>בסיס</u> $j=i$.

השמורה $X=Max\{A[i], A[i]\}$, אכן זה מה שמושם ע"י ה-if.

<u>נניח שהשמורה נכונה עבור $j=k$</u>

$X=Max\{A[i], ... A[k], A[n]\}$

לאחר איטרציה נוספת $j=k+1$

נניח שמתקיים ש $A[k+1]>x$ אז אחרי זה נכנס ל-x ב *

לפי סוף השמורה נקבל ש $X=Max\{A[i]... A[k+1], A[N]\}$ נניח שמתקיים $A[k+1] \leq X$, אז לפי סוף השמורה שוב מתקיים ש $X=Max\{A[i]...A[k+1], A[N]\}$.

מש"ל

<u>שמורה ב **</u>:

במקומות $1...i$ נמצאים i האיברים הגדולים במערך במספר יורד.

<u>הוכחת השמורה:</u>

<u>בסיס</u> $i=1$ כשהיינו ב* עם $n=j$ התקיים $X=Max\{A[1]... A[n]\}$

לאחר ביצוע ה swap הגדול במערך נמצא ב $A[1]$, זה מתקיים

שמורה ** עם $i=1$. <u>נניח שנכון ב** עם $i=k$ ומתקיים:</u>

במקומות $k...1$ נמצאים k האיברים הגדולים במערך, במספר יורד.

<u>אנו מבצעים איטרציה $i=k+1$</u> כשהגענו ל* עם $n=j$ מתקיים שמורה *

$X=Max\{A[k+1]... A[N]\}$. במקומות $1...k$, נמצאים k הגדולים במערך

כולם במספר יורד, לכן הם $\geq$ מ $A[k+1]$, לכן האיברים במקומות $1...k+1$

הם עדיין במספר יורד, כיוון ש $A[k+1] \geq$ מכל האיברים

במקומות n ... $l=k+2$, סדר האיברים במקומות

$k+1$ ... 1, הם הגדולים במערך ובסדר יורד.

מש"ל.

בסיום הלולאה: האלגוריתם ממיין את המערך בסדר יורד.

הוכחה: ע"פ השמורה ** $n=i$: במקומות $n...1$ נמצאים

(n האיברים הגדולים במערך), ובסדר יורד.

מש"ל.

שאלה:

מה הסיבוכיות של האלגוריתם הבא:

* ידוע שסיבוכיות של $func(n)=\Theta(n^2)$.

```
prog(n)
while n>1
    x<-func(x)
    print(x)
    n<-n/3
end (while)
```

$$T(n)=c\cdot n^2+m+c\cdot\left(\left(\frac{n}{3}\right)^2\right)+m+c\left(\left(\frac{n}{3^2}\right)^2\right)+m+\dots+c\cdot\left(\frac{n}{3^k}\right)^2+m$$

$$n=3^k \qquad k=\log_3 n$$

$$T(n)=m(\log_3 n+1)+c\cdot n^2\left(1+\left(\frac{1}{3}\right)^2+\left(\frac{1}{3^2}\right)+\dots+\frac{1}{3^{\log_3 n}}\right.$$

$$T(n)\le m(\log_3 n+1)+c\cdot n^2$$

$$T(n)=\Theta(n^2)$$

תרגול
-3-

שאלה: מה הסיבוכיות של הפונקציה הבאה:

```
last P ← N
for i ← 1 to N
    J ← i + 1
    While J ≤ last P
        if A[i] = A[j]
            then Delete(J)
        else J ++;
end for
```

Delete(J):

```
for l = j+1 to last_P
    A[l-1] ← A[l]
end for
last P --;
```

נוכיח שהסיבוכיות הינה $O(n^2)$:

1. נוכיח שמספר הפעמים שה- Delete מופעל לכל היותר n:
ע"י מעקב אחר המשתנה last-P, הוא מאותחל ב n. הוא לא גדל אף פעם ובכל קריאה של Delete הוא קטן ב-1.
לאחר n פעמים ש Delete מופעל ערכו 0, ולכן לא ניתן להפעיל יותר, כי $j \ge 2$ והתנאי של ה while לא יתקיים.
מש"ל

2. הסיבוכיות: נחלק לסיבוכיות מתוך ה Delete + סיבוכיות מחוץ ל Delete (המעבר עובר באופן ספרתי וסך כל זמני הלולאה שלו מכל רחבי הקוד שווה לסיבוכיות). מתוך ה Delete לכל היותר n פעמים נכנס, ולכל היותר n אלמנטים מפנים. סה"כ $O(n^2)$.
מחוץ ל Delete 2 לולאות מגודל $O(n)$ מקוננות $O(n^2)$
סה"כ $O(n^2)$.

מש"ל.

משפט המאסטר :

נניח שפונקציה, הסיבוכיות שלה $T(n)$ מקיימת את התנאי הבא:

$$T(n)=a\cdot T\left(\frac{n}{b}\right)+d f(n)$$

$$T(n)=2T\left(\frac{n}{2}\right)+c\cdot n$$

$$T(n)=\Theta(n\log n)$$

$f(n)$ פונקציה כפלית

$$f(a\cdot b)=f(a)\cdot f(b)$$

$$f(n)=n^3$$

$$f(n\cdot m)\overset{?}{=}f(n)\cdot f(m)$$

$$(n\cdot m)^3=n^3\cdot m^3$$

אם $a>f(b)$ $\quad T(n)=\Theta\left(n^{\log_b a}\right)$

אם $a=f(b)$ $\quad T(n)=\Theta\left(n^{\log_b a}\cdot\log n\right)$

אם $a<f(b)$ $\quad T(n)=O(f(n))$

$b>1$

* פתור את הנוסחה הרקורסיבית: $\quad T(n)=2\cdot T\left(\frac{n}{2}\right)+c\cdot n^3$

נפתור במשפט המאסטר $\quad f(n)=n^3 \quad a=2,\ b=2$

פונקציה כפלית $\quad a\ ?\ f(b)$

$$2<2^3$$

$$\Rightarrow \quad T(n)=O(n^3)$$

* פתור את : $\quad T(n)=4\cdot T\left(\frac{n}{2}\right)+7\cdot 1$

משפט המאסטר

$a=4 \quad f(n)=1$

$b=2 \quad d=7$

$$4>f(2)=1$$

$$T(n)=\Theta\left(n^{\log_2 4}\right)=\Theta(n^2)$$

שאלה:

נתון הצעה לאלגוריתם למיון מערך A מגודל n של איברים שונים.

א. אם $|A| \le 3$ מיין ע"י השוואות

ב. קרא ברקורסיה לאיברים $n \ldots \frac{1}{3}n$ ($\frac{2}{3}$ אינדקסים העליונים)

ג. קרא ברקורסיה לאיברים $\frac{2}{3}n \ldots 1$ ($\frac{2}{3}$ האינדקסים התחתונים).

1. מה הסיבוכיות?

2. האם ממיין?

$$T(n) = 2 \cdot T\left(\frac{2}{3}n\right) + d \cdot 1$$

$$\downarrow$$

$$\frac{n}{\frac{3}{2}}$$

$a = 2$

$b = 3/2$

$f(n) = 1$

$$2 > 1$$

$$T(n) = O\left(n^{\log_{3/2} 2}\right) = O\left(n^{1.709}\right)$$

פחות טוב מ- $n\log(n)$.

תרגול
-4-

שאלה:

נתון מערך A של מספרים שלמים שונים, ממוין. מספר טבעי i נקראת נקודת שֶבֶת במערך, אם מתקיים $A[i]=i$

```
Find (A,i,j)
   if i>j return(false)
 * if i=j return (A[i] = i)
   // j>i
     m <- ⌊(i+j)/2⌋
 (1) if A[m] > m return (Find (A, i, m-1))
 (2) if A[m] = m return (true)
     Return (Find (A, m+1, j))
   end
```

נוכיח שהפונקציה עוצרת תמיד:

נוכיח באינדוקציה על $i-j$.

בסיס: אם $j-i\le 0$ אנחנו עוצרים באחת מ2 ה if הראשונים.

נניח שבכל $j-i\le k$ הפונקציה עוצרת.

נוכיח ל $j-i=k+1$, אם מתקיים התנאי בשורה ה(1) $m-1-\cancel{i} \overset{?}{<} j-\cancel{i}$

$m-1 \overset{?}{<} j$

$m-1<m<j$ [illegible]

אם התקיים התנאי של שורה (2), אז ברור שעצרנו, אחרת:

$\cancel{j}-(m+1)<\cancel{j}-i$

$-m-1<-i$ $\quad i-m-1<0$

$m+1>i$ $\quad$ [illegible] $i-m\le 0$

$i-m-1\le 0-1\le 0$

לכן הקריאה זו עוצרת

מש"ל.

נוכיח שהפונקציה מוצאת האם יש נקודת שבת בטווח $i..j$ ב-A:

נוכיח זאת באינדוקציה $j-i$.

אם $0 \ge j-i$ אז: אם $0 > j-i$, הטווח ריק ולכן אין נקודת שבת.
אכן זה מה שמוחזר. אם $j-i=0 \Rightarrow j=i$, אכן מוחזרת התשובה הנכונה ב-*.

אחרת $j-i>0 \Rightarrow i<j$, נניח שהפונקציה מחזירה נכון עבור $j-i \le k$. נוכיח עבור $j-i=k+1$:
הקריאות הרקורסיביות, כפי שראינו בהוכחה הקודמת עם הסדר קטן יותר לכן עוצרות ומחזירות ערך נכון. אם מתקיים $A[m]>m$ כיוון שאיברי המערך שונים שלמים וממוינים הרי ש-$A[l]-l$ גדל מימין לאינדקס m, ולכן יתקיים $A[l]>l$ לכל $m \le l$, ולכן אם יש נקודת שבת היא מצויה באינדקסים $i..m-1$, ומהנחת האינדוקציה $find(A,i,m-1)$ תחזיר ערך נכון לשאלה כולה.
אם $A[m]=m$ ברור שיש נקודת שבת, ולכן מוחזרת תשובה נכונה.
אחרת $A[m]<m$, ולכן לפי אותו שיקול לא תיתכן נקודת שבת משמאל ל-m, ולכן אם יש נקודת שבת היא רק מימין ל-m כלומר בטווח $m+1..j$, ומהנחת האינדוקציה אנו מניחים ש $find(A,m+1,j)$ תחזיר את התשובה הנכונה.

תמורה:

מערך A מגודל n נקרא תמורה (פרמוטציה) אם ורק אם:
$1 \le A[i] \le n$ וכולם שונים
שלמים

3	7	1	5	6	2	4

נאמר שמערך B של שלמים שונים שקול לפרמוטציה A אם ורק אם
לכל $1 \le i < j \le n$ מתקיים $A[i]<A[j] \iff B[i]<B[j]$

דוגמא:

B	-3	4	17	5	9	3	8
A	1	3	7	4	6	2	5

נתון מערך B של n שלמים שונים, רוצים לחשב את A הפרמוטציה השקולה ל B.

פתרון: נשתמש במערך C[1...n] של רשומות

```
for i ← 1 to n
    C[i].val ← B[i]
    C[i].ind ← i
end for
sort(C)    מיון לפי ה val
for i ← 1 to n
    A[C[i].ind] ← i
```

מסקנה: ניתן לחשב תמורה שקולה של וקטור מגודל n בסיבוכיות $O(n\log n)$

תמורה שקולה לוקטור

ראינו כיצד ניתן לחשב תמורה שקולה לוקטור ב $O(n\log n)$
נוכיח שלא ניתן לעשות זאת ע"י פעולת השוואה והשמה בלבד
בפחות מ $O(n\log n)$ פעולות.

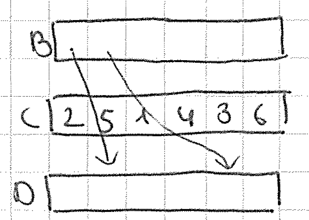
<u>משפט</u>: לא ניתן למיין וקטור A מאורך n, ע"י פעולות השוואה והשמה בלבד בפחות $O(n\log n)$ פעולות.

<u>הוכחה</u>: נניח בשלילה קיימת פונקציה EqParm(A) המשתמשת בפעולות השוואה והשמה בלבד, מבצעת פחות מ $O(n\log n)$ פעולות ומוצאת תמורה שקולה לוקטור A ($|A|=n$).

נתבונן בפונקציה הבאה: Sort(B)

```
C ← EqParm(B)
for i ← 1 to n
D[C[i]] ← B[i]
end for
Out(D)
```

לדוגמא:



אנו מיינו את B, כשביצענו אך ורק פעולות השוואה והשמה (בתוך EqParm - כפי שהייתה הנחת השלילה)

<u>הסיבוכיות</u>: $T(n) + O(n) < O(n\log n)$ הסיבוכיות של EqParm

$\uparrow$ $o(n\log n)$

<u>מסקנה</u>: לא ניתן למצוא תמורה שקולה, ע"י פעולות ההשוואה והשמה בלבד בפחות מ $O(n\log n)$ פעולות.

שאלה:

וקטור נקרא בתצורת גבעה אם קיים אינדקס $1 \le j_0 \le n$

ומתקיים: לכל $1 \le l < k \le j_0$ $A[l] \le A[k]$

וכן לכל $j_0 \le l < k \le n$ $A[l] \ge A[k]$

<u>טענה</u>: לא ניתן להפוך וקטור B בתצורת גבעה, ע"י פעולת השוואה והשמה בלבד, בפחות מ $O(n\log n)$ פעולות.

<u>הוכחה</u>: נניח בשלילה קיימת פונקציה Hill(x) המקבלת וקטור מאורך n, ומחזירה וקטור עם אותם ערכים בתצורת גבעה, מבצעת פעולת השוואה והשמה בלבד ופועלת בסיבוכיות קטנה מ $O(n\log n)$.

Sort (x)

$A \leftarrow Hill(x)$

מצאנו את האינדקס j_0 של האיבר המקסימלי ב A

* $D_1 \leftarrow A[1 \ldots j_0]$

$D_2 \leftarrow A[j_0+1 \ldots n]$

$D_3 \leftarrow$ D_2 הפוך

$D \leftarrow Merge(D_3, D_1)$

Out (D)

אכן מיינו את X סיבוכיות * $O(n)$

סיבוכיות Hill(x) < $O(n\log n)$ סה"כ < $O(n\log n)$

כמו כן Hill(x) מבצע פעולת השוואה והשמה בלבד

ב * מבצע רק השוואות והשמה

Sort ממיינת בפחות מ $O(n\log n)$ פעולות.

ע"י פעולות השוואה והשמה בלבד בסתירה למשפט.

לכן לא ניתן להפוך וקטור לתצורת גבעה בפחות מ $O(n\log n)$ פעולות, ע"י פעולות השוואה והשמה בלבד.

שאלה:

הפונקציה Third(A) מחזירה את המערך A כאשר $1/3$ מהאיברים השמאליים שלו ממויינים בסדר עולה.

טענה: לא קיימת פונקציה כזאת, המשתמשת אך ורק בפעולות השוואה והשמה ופועלת בסיבוכיות $< O(n\log n)$.

הוכחה: נניח בשלילה קיימת פונקציה Third(x) אשר מחזירה את המערך x כאשר $1/3$ מאיברי השמאליים ממויינים בסדר עולה, משתמשת אך ורק בהשוואה והשמה, ופועלת בסיבוכיות קטנה מ $O(n\log n)$. נתבונן בפונקציה הבאה:

Sort(A)

1. נקח מערך B מגודל $3n$
2. נכניס את A ל $1/3$ האיברים הראשונים של B
3. $T \leftarrow$ Third(B)
4. Out(T[1..n])

מהנחת השלילה בתוך Third מופעל אך ורק השוואות והשמות.
בשאר הפקודות כנ"ל מופעל רק השוואות והשמות.

סיבוכיות:
1. $O(1)$
2. $O(n)$
3. $O(3n\log 3n) = O(n\log n)$
4. $O(n)$

סה"כ $< O(n\log n)$

ואכן מופעל מיון

מסקנה: הנחת השלילה שגויה כי ניתן להפעיל Third(x) בפחות מ $O(n\log n)$ פעולות השוואה והשמה.

תרגיל 6

תכנות דינמי:

שימוש במבנה נתונים. כדי לשמור את הפתרון הבעיה עם קלטים קטנים יותר (כמו רקורסיה ללא קריאות רקורסיביות)

בעיה:

נתון מערך A של שלמים $n=|A|$. מחפשים $n \geq j \geq i \geq 1$

כך ש $\sum_{k=i}^{j} A[k]$ מקסימלי

פתרון בסיבוכיות $O(n^2)$:

$O(n^2)$: בלולאה $i=1..n$, בתוכה $j=i...n$. נמצא את המקטע המקסימלי שמתחיל ב-i, נשמור את התוצאה במערך B. אח"כ נמצא MaxB.

$O(n)$: נשתמש במבנה נתונים $B[1...n]$

$B[i]$ - הסכום הטוב ביותר של מקטע המסתיים ב i

```
B[1] <- A[1]
for i <- 2 to n
      if B[i-1] < 0 then B[i] <- A[i]
                   else  B[i] <- B[i-1] + A[i]
end for
out( Max(B))
```

סה"כ $\Theta(n)$.

בעיה:

נתון מערך A של n מס שלמים, נגדיר $diff(i,j)=A[j]-A[i]$ עבור $i \leq j$

מחפשים את הערך Maxdiff

פתרון: נעזרת מערך עזר B $B[1...n]$

$B[i]$ - הערך המינימלי ב A בתחום $1...i-1$

$C[i]$ - ההפרש הטוב ביותר עבור אינדקס הימיני i.

-המשך-

```
B[2] ← B[1]
for i←3 to n
    B[i] ← min{B[i-1], A[i-1]}
end for
for i←2 to n
    C[i] ← A[i] - B[i]
end for
out(Max(C))
```

כל פעם $\Theta(n)$

שאלה:

נתון מערך A של מספרים חיוביים, מהו הסכום המקסימלי של תת מערך של A שאין בו שני איברים רצופים.

פתרון: נשתמש בשני מערכי עזר $w[1..n]$, $wo[1..n]$

$w[i]$ הסכום הטוב ביותר במקטע $i...1$, כשחייבים לקחת את $A[i]$.

$wo[i]$ הסכום הטוב ביותר במקטע $i...1$, כשחייבים לא לקחת את $A[i]$.

```
w[1] ← A[1]    wo[1] ← 0
for i←2 to n
    w[i] ← A[i] + wo[i-1]
    wo[i] ← max{wo[i-1], w[i-1]}
end for
out(Max{wo[n], w[n]})
```

תרגיל 7

שאלה:

נתון מערך A באורך n המכיל מס' טבעיים (מחירים של מוצרים)

מכל מוצר ניתן לקחת 0,1,2 פריטים.

תחת המגבלה הבאה: אם לקחת ממוצר 0 יחידות אז משני המוצרים הסמוכים ניתן לקחת 0/1/2.

1 יחידות => אז ניתן לקחת 0/1.

2 יחידות => אז ניתן לקחת 0.

מעוניינים לבחור מוצרים שסכום מחירם מקסימלי.

<u>פתרון</u> => נשתמש ב 3 מערכי עזר B,C,D.

B[i] - המחיר הטוב ביותר שניתן לקבל מבחירת מוצרים 1...i

כשהאיבר ה i חייבים לקחת 0 פעמים.

C[i] " - חייבים לקחת 1.

D[i] " - חייבים לקחת 2.

```
B[1]← 0   C[1]← A[1]   D[1]← 2A[1]
for i←2 to n
        B[i]← Max{ B[i-1], C[i-1], D[i-1] }
        C[i] = A[i] + Max{ B[i-1], C[i-1] }
        D[i] = 2A[i] + B[i-1]
end for
Return(Max{B[n], C[n], D[n]}).
```

שאלה: נתון לוח משובץ $n \times n$ ואומן משחק.

עלינו להעביר את האומן מהשורה התחתונה, $i=1$ אל השורה העליונה.
בכל מעבר משורה לשורה שמעליה מותר להשאיר באותה עמודה או לצעוד
אחת ימינה או שמאלה.
נתונה פונקציה אשר נותנת לנו, לכל תזוזה של האומן כמה
כסף אנו מקבלים או משלמים.

$$Price\left((i,j),\ \left(i+1,\ \begin{matrix} j \\ j-1 \\ j+1 \end{matrix}\right)\right)$$

מעוניינים להעביר את האומן מנקודה כלשהי בשורה ה1 עד לנקודה
כלשהי בשורה הn ולקבל סכום מקסימלי.

<u>פתרון:</u>

נשתמש במערך עזר $B[n \times n]$ המקיים:
$B[i,j]$: הסכום המקסימלי שניתן לקבל מהעברת האומן מנקודה כלשהי
בשורה ה1 ועד המיקום (i,j).

```
for j←1 to n
    B[1,j]←0
for i←2 to n
    for j←1 to n
                              ⎧ B[i-1][j-1] + price((i-1,j-1),(i,j))
        B[i,j] ← Max*         ⎨ B[i-1][j] + price((i-1,j),(i,j))
                              ⎩ B[i-1][j+1] + price((i-1,j+1),(i,j))
    end for
end for
```

* יש לשים לב עבור $j=1,n$ יש לבחור max בין שני איברים.

Out(Max, B[n, 1...n])

סיבוכיות $O(n^2)$

שאלה: מה עושה הקוד הבא:

```
for i←1 to Max(A)          // איפוס המלים
        B[i]←0
end for
for i←2 to N               // מניית מספרי A //
        B[A[i]]++
end for
j←1
for i←1 to Max(A)          // מיין מחדש את איברי A
        While B[i] > 0         הערה: j דורס את A.
                A[j]← i
                j++
                B[i]--
        end While
end for
```

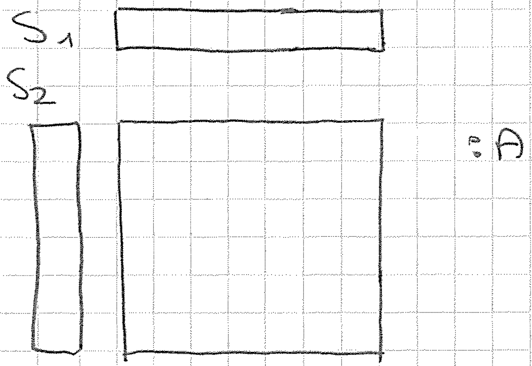
סיבוכיות

נגדיר $m=Max(A)$

$T(n,m)=\Theta(m+n)$

תזכורת LCS

Longest Common Substring.



$A[i,j]$ - אורך תת המחרוזת המקסימאלית המשותפת ל $S_2[1\ldots i]$ ו $S_1[1\ldots j]$

אם $S_1[j] = S_2[i]$

$A[i,j] \leftarrow A[i-1][j-1] + 1$

אחרת

$A[i,j] \leftarrow \max\{A[i,j-1], A[i-1,j]\}$

אחרי מילוי A

out $(A(n,m))$.

שאלה: אותה שאלה, רק שמחפשים מחרוזת חוקית רציפה מקסימאלית

פתרון -ז (האורך המקסימאלי של מחרוזת שהיא גם מחרוזת חוקית גם S_1 וגם S_2 שמחרוזת מסתיימת ב $S_2[i]$ ו $S_1[j]$.

אם $S_1[j] = S_2[i]$

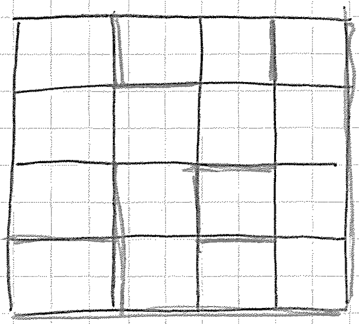
$A[i,j] = A[i-1,j-1] + 1;$

אחרת: $A[i,j] = 0$

אם $S_1[j] \neq S_2[i]$ אז השורה והעמודה (הראשונה) יאופסו:

$A[1,i] \leftarrow 0$

שאלה:



נתון מבוך nxn

כמה דרכים ניתן לעבור

במבוך זה מהפינה השמאלית

העליונה עד הימנית התחתונה כך ש:

א) אסור לעבור דרך הקירות ( )

ב) בכל שלב מותר ללכת ימינה או למטה.

הצעה לייצוג הקירות: ע"י מערך A[nxn]

A[i,j] = 4 3 2 1

דוגמא:

A:

2	1	2	2
4	4	1	2
3	2	1	2
3	1	1	3

פתרון ← נשתמש במערך עזר B[nxn]

B[i,j] = מספר הדרכים להגיע מ (1,1) עד (i,j)

(2 קירות) if(A[i-1,j] = 3 || 1) && A[i,j-1] = 2 || 3)

B[i,j]=0

(קיר רק מלמעלה) if (A[i-1,j] == 3 || 1) && A[i,j-1] = 4 || 1)

B[i,j] ← B[i,j-1]

(קיר רק משמאל) if (A[i-1,j] = 4 || 2) && A[i,j-1] = 3 || 2)

B[i,j] ← B[i-1,j]

if ((A[i-1,j]) = 4 || 2) && (A[i,j-1] = 4 || 1))

B[i,j] ← B[i-1,j] + B[i,j-1]

out(B[n,n])

תרגיל -9-

שימוש חשוב בשיטת הפעלים.

(הכפלת מיפוי מתוך "זכב")

הבעיה עלופי פעמאו :

נתון מערך A מגודל n של שלמים, חיוביים שונים, האם יש

בו 2 מספרים שסכומם 10.

נפתר :

נקצה מערך B בגודל max(A)

*אני מניח שמותר (ע״ב מערכים) לעשות דווינקטים שליליים.

```
for i←1 to n
    B[A[i]] ← i
```

בדיקה ב-$\Theta(1)$, זאת השאלה האם $a \in A$?

```
if (A[B[a]] == a)
```

```
for i←1 to n
    if A[B-A[i]] = 10 - A[i]
        return true;
return false;
```

תרגיל 2 :

נגדיר תהוו פונקציה A מערך מגודל n של שלמים.

מחזירה בסיבוכיות $\Theta(n)$ Select(A,k) , k מספר טבעי.

את האיבר ה-k שגודלו ב-A.

$$Select(A,1) = Min(A)$$

$$Select(A,n) = Max(A)$$

נתון מערך A המכיל n משכורות.

רוצים לייצר מערך B בגודל n בוליאני

המשכורת של העובד ה-i היא מעשירון העליון $\iff$ $B[i] = T$

בסיבוכיות $\Theta(n)$

```
C = Select(A, ⌊9/10 n⌋)
for i ← 1 to n
    if A[i] ≥ c then B[i] ← true
                else B[i] ← false
```

שאלה:

קלט: מערך A בגודל n של מספרים חיוביים שונים, וכן מס' חיובי k.

פלט: האיבר $A[j]$ המינימלי במערך שסכומו מתקיים

$\sum (A[i]) \leq k$

$A[i] > A[j]$

נניח ש A ממויינת

פתרון ← נסכום את איברי A מהסוף

לכן יש לנו פתרון בסיבוכיות $O(n\log n)$ (בעזרת מיון).

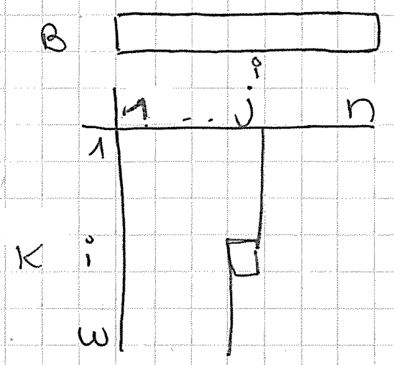
פתרון ב-סיבוכיות n:

```
Find(A, k)
    c ← Select(A, n/2)      חציון.          T(n) = c·n + T(n/2)
    A1 ← הכנס את הגדולים (או שווים) מ-c שב-A
    A2 ← הכנס את הקטנים מ-c שב-A
    Sum ← סכום איברי A1
    if sum = k return (הקטן שב-A1)
    if sum > k return (find(A1, k))
    if sum < k return (find(A2, k-Sum))
```

תרגיל
-10-

שאלה:

נתון מערך B מאורך n של טבעיים, ונתון מספר טבעי w
רוצים לבדוק האם קיים ב B תת מערך שסכום איבריו בדיוק.
מצאו פתרון בסיבוכיות w·n.



פתרון ->

נשתמש במערך $A[n \times w]$

$T = A[i,j]$ אמ"מ קיים תת מערך של $B[1..j]$ שסכומו בדיוק i.

$A[w,n]$ הפתרון.

אם ב B יש איבר 1 באינדקס הראשון J

אתחול:
- $A[1,l]=1$ לכל $l \geq j$
- $A[1,l]=0$ לכל $l<j$
- for($i \leftarrow 1$ to w) $A[i,1] \leftarrow 0$
- $A[B[1],1] \leftarrow 1$

```
for i←2 to n
    for k-1 to w
        if B[i] = k then A[k,i] ← 1
        if B[i] > k then A[k,i] ← A[k,i-1]
        else A[k,i] ← A[k-B[i], i-1]
                  or A[k, i-1]
    end for
end for
Out (A[w,n])
```

דוגמא:

B

5	2	3	1

A

	1	2	3	4
1	0	0	0	1
2	0	1	1	1
3	0	0	1	1
4	0	0	0	1
5	1	1	1	1
6	0	0	0	1

98

אלגוריתם א' - מעודכן הקריה חובבים.

שאלה 1:

נתון מערך A של המספרים שלמים מגודל N, ונתון אלגוריתם הבא:

```
DD(A)
down ← true
i ← 1
A[N] ← max int
While down do
    if A[i] ≥ A[i+1] then
        i ← i+1
    else
        down ← false
    end(if)
End(While)
return (i)
```

א. תארו את מטרת האלגוריתם DD:

האלגוריתם מקבל מערך, ובודק אם הוא ממוין, כלומר $A[i] > A[i+1]$ (מספר יורד)

זה אומר שהאיבר הבא יותר קטן מהקודם לכן המערך לא ממוין.

אם מחזיר return $i = N-1$ סימן שהמערך ממוין, ואחרת $N-1 \neq i$ מערך לא ממוין.

ב. נסחו משפט נכונות:

יהי A מערך. האלגוריתם DD(A) בודק האם המערך ממוין מספר יורד.

ומחזיר N-1 אם מערך ממוין, ואינדקס i האיבר הראשון שלא ממוין.

ג. שמורה:

$A[1] \geq A[2] \ldots \geq A[i]$

הוכחת השמורה: נוכיח באינדוקציה סגור i.

בסיס: $i = 1$ $\quad$ $i = 1$; $A[1] \geq A[1]$

נניח כי $i = k$ לכן $A[1] \geq \ldots \geq A[k]$

נוכיח עבור $i = k+1$, לפני כן התקיימה השמורה עבור $i = k$

לכן $A[1] \geq \ldots \geq A[k]$ $\quad$ $A[1] \geq \ldots \geq A[k+1]$

-2

חסרה התייחסות לתנאי

הוכחת הטענה:

נניח שהוחזר t $t \neq N-1$ = ~~שם~~

הטענה נכונה עבור ה-t לכן נכונה גם עכשיו.

נניח ש $t=N-1$ סימן שהמערך ממוין בסדר יורד

לכן תחזיר האינדקס 1 לפני אחרון.

נכונה למעלה.

נניח הוחזר $k=i$ כאשר התקיימה ~~עבור~~ ?

השמורה עבור $i=k$, אזי כל האיברים במקומות

$A[1...k]$ מקיימים את תנאי השמורה

כאשר הוחזר האינדקס של האיברים $A[k]$

שבה אלו המערך ממוין בסדר יורד.

```
WTF2(A, m, pm)
if A[1] > A[2] then
  {m ← 1 : pm ← 2}
else {m ← 2 : pm ← 1}
for i ← 3 to N
  if A[i] > A[m] then
    {pm ← m
     m ← i}
  else if A[i] > A[pm] then
    {pm ← i}
end for
out (m, pm)
```

שאלה 3:

א. מה מבצע אלגוריתם הבא?

האלגוריתם WTF2 מקבל (הקלט) מערך A ו-2 פרמטרים של מספרים שלמים, מבצע חיפוש של שני איברים המקסימליים ומחזיר כפלט: m את האינדקס של האיבר הגדול ביותר וב pm את האינדקס של האיבר השני בגודלו.

ב. הוכיחו את נכונות האלגוריתם:

שמורה:

$A[m] \geq A[pm] \geq A[1] \dots A[i-1]$

הוכחת השמורה:

אינדוקציה עד $i-1$.

בסיס $i=3$:

כש $i=3$ אם $A[1]>A[2]$ אז $m=1$ ו $pm=2$

אם $A[2]>A[1]$ אז $m=2$ ו $pm=1$ $\Leftarrow$ $A[m]>A[pm]$

נניח עבור $i=k$, לכן $A[m]>A[pm]>A[1] .. A[k-1]$

צעדי רק התקיימה השמורה, כל האיברים במקומם

נניח עבור $i=k+1$, לכן $A[m]>A[pm]>A[1] \dots A[k+1-1]$

$\Leftarrow$ $A[m]>A[pm]>A[1] \dots A[k]$

לכן הטענה נכונה עבור $i-1$.

הוכחת העצירה:

נניח שהותפר $t_1=m$ ו $t_2=pm$ ו $m \neq N$ וגם $pm \neq N$

אז השמורה התקיימה לפני, לכן תתקיים גם עכשיו.

נניח ש $A[N]>A[m]$ אז $pm=m$, $m=i$ ולכן השמורה התקיימה ל-$n-1$ ותתקיים גם עכשיו. נניח ש $A[N]>pm$ אז $A[m]>A[N]>A[pm]$

$\Leftarrow$ $pm=N$ $\Leftarrow$ לכן השמורה התקיימה ב-1 ותתקיים גם עכשיו.

```
WTF3 (A, m, nm)
  m ← A[1]
  nm ← 1
  for i ← 2 to N
    if (A[i] > m) then
      {m ← A[i]
       nm ← 1}
    else if A[i] = m then
       nm++
  end for
```

שאלה 4:

א. הסבירו את אק"ים הערכים המוחזרים:

האלגוריתם WTF3 מקבל כקלט מערך A

ומחזיר את האיבר המקסימלי ואת מספר

הפעמים שהוא מופיע, (ומחזיר כפל)

את האיבר המקסימלי, ואת כמות הפעמים שמופיע.

שמורה:

$m \geq A[1] \ldots A[i-1]$

מס' פעמים של $m \geq nm$.

הוכחת השמורה:

אינדוקציה על $i-1$.

בסיס: עבור $i=2$, $m=A[1]$ ו $nm=1$, לכן לין $m=A[1]$ $nm=1$

והתנאי מתקיים. נניח שהשמורה נכונה עבור $k=i$: $m \geq A[1] \ldots A[k-1]$ (*)

$\wedge$ מס' הפעמים של $m \geq nm$. נוכיח עבור $k+1=i$: לפני כן התקיימה השמורה עבור $k=i$ (*) לכן,

$m \geq A[1] \ldots A[k+1-1] = m \geq A[1] \ldots A[k]$

לכן הטענה נכונה.

הוכחת הטענה:

$t_1 \geq A[N]$

נניח שהוחזר $t_1=m$ ו $t_2=nm$,

אז השמורה התקיימה לפני לכן תתקיים עכשיו.

אם $A[N]>m$ אז $m=A[i]$ ו $nm=1$, לפני זה השמורה

התקיימה עבור $i-1$, לכן הטענה מתקיימת.

נניח ש $A[i]=m$ אז נכנסים ל if השני ומקדמים את nm

לפני זה השמורה התקיימה, לכן מתקיימת גם עכשיו.

98 קרינה חופשית. אלגוריתמים - מטלה 2

שאלה 1:

```
What (B, q, n)
  j ← n
  While j ≥ 1 and B[j-1] > q
    B[j] ← B[j-1]
(*) → j ← j-1
  end While
  Return (j)
```

נתון מערך B ממוין מספר שלם מגודל n.
האלגוריתם what
מקבל כקלט מערך B
ממוין מספר שלם, הוא q, וגודל n.
בודק האם המספר בין המספרים
במערך ומחזיר את האינדקס של המיקום
המיועד שלו.

ב. הוכח נכונות ע"י הגדרת שמורה במקום שתבחר ותוכיח.

שמורה: $B[n] \ldots B[j-1] \geq q$

הוכחת השמורה:

בסיס: מערך B. מגודל 2, $B[n]..B[1] > q$ אם נכנס ל while מתקיים.

הנחת האינדוקציה: $j=k$, $B[n] \ldots B[k-1] > q$

לאחר איטרציה נוספת אם נכנס ללולאה סימן ש $j=k+1$

$B[n] .. B[k-1] .. B[k] > q$

אם זה לא מתקיים סימן שיצאנו מה while.

הוכחת הנכונות:

לפי השמורה * $B[j-1] \leq q$, ויחזיר אינדקס j

מש"ל

שאלה ב:

<u>מה מבצע האלגוריתם הבא:</u> שאיקרין דרי השוואה

האלגוריתם Q מקבל כקלט מערך A

והפלט - מערך, ממיין את המערך

בסדר יורד.

```
Q (A,n)
for i<-1 to n-1
   elm <- -∞
   for j<-n downto i
   if (A[j] > elm) then
       elm <- A[j]
       elmind <- j
*  ---> end if
   end for
** ---> Swap (A,elmind,i)
end for
```

<u>שמורה ב*:</u>

$elm = Max\{A[n] \dots A[j]\}$

ב elmind המיקום שלו במערך.

<u>הוכחת השמורה:</u>

<u>בסיס</u> $j=i=1$

השמורה $elm = Max\{A[1], A[1]\}$,

אכן זה מה שמבוצע ע"י ה if.

<u>נניח שהשמורה נכונה עבור $j=k$</u>: $elm = Max\{A[n] \dots A[k]\}$

לאחר <u>איטרציה נוספת $j=k+1$</u>, נניח שהתקיים $A[k+1] > elm$, אז אומר

זה נכנס elm ב* על-סמך, השמורה נקבל ש $elm = Max\{A[n] \dots A[k+1]\}$

נניח שהתקיים $A[k+1] \leq elm$, אז על-סמך השמורה שוב מתקיים ש

$elm = Max\{A[n] \dots A[k+1]\}$

מש"ל

<u>שמורה ב**:</u> במקומות $1 \dots i$, נמצאים i האיברים הגדולים

במערך, ובסדר יורד.

<u>הוכחת השמורה:</u>

<u>בסיס:</u> $i=1$, כשהיינו ב* עם $j=1$ התקיים $elm = Max\{A[n] \dots A[1]\}$

לאחר ביצוע swap הגדול במערך נמצא ב $A[1]$, די מספיק שמורה

** עם $i=1$. <u>נניח שנכון ב** עם $i=k$, ומתקיים:</u> במקומות $1 \dots k$

נמצאים k האיברים הגדולים במערך בסדר יורד. אנו מבצעים <u>איטרציה</u>

<u>$i=k+1$</u>, השתמשנו ב* עם $j=i$ מתקיים שמורה * $elm = Max\{A[n] \dots A[k+1]\}$

במקומות $1 \dots k$, נמצאים k הגדולים במערך כולם בסדר יורד, לכן המס' $A[k+1] \leq A[k]$,

לכן האיברים במקומות $k+1 \dots 1$ בסדר יורד, כיוון ש $A[k+1] \leq$ מכל

האיברים במקומות $n \dots k+2 = l$, לכן האיברים במקומות $1 \dots k+1$

הם הגדולים במערך בסדר יורד.
מש"ל

<u>טענת נכונות</u>: האלגוריתם What ממיין את המערך בסדר יורד.
<u>הוכחה</u>: לפי השמורה ** $i=n-1$: במקומות $n...1$ נמצאים
n האיברים הגדולים במערך ובסדר יורד.
מש"ל //

<u>שאלה 3:</u>

```
Q(A)
m ← 1
mm ← 1
for i ← 1 to N-1
    if A[i] ≤ A[i+1] then
        m ← m+1
    else
        mm ← max{m,mm}
        m ← 1
* → end for
mm ← max{m,mm}
out(mm)
```

<u>מה מבצע האלגוריתם הנ"ל:</u>

האלגוריתם Q מקבל כקלט מערך A ומבצע חיפוש של מערך ממויין בסדר עולה, ומחזיר כפלט את הרצף של המיון בסדר עולה הארוך ביותר.

<u>שמורה ב*:</u>

לאחר סיום if ולפני סיום ה-for באיטרציה i במשתנה m או mm יש את הרצף הארוך ביותר של איברים מ- i+1 ... 1

הוכחת השמורה:

אינדוקציה על i

<u>בסיס i=1</u> מתקיים: אם $A[1] \le A[2]$ $m=2$ אם $A[2] > A[1]$ $m=1$ בשני המקרים ב m יש את הרצף המקסימלי בתחום 2..1.

<u>הנחת אינדוקציה:</u> נניח שהשמורה מתקיימת עבור $i=K$ כלומר לאחר סיום הלולאה מתקיים במשתנה mm או ב m יש את הרצף הארוך ביותר, בתחום $K+1$...1.

נניח כי מבצעים <u>איטרציה $i=K+1$</u>, אם התנאי יתקיים $A[K+1] \le A[K+2]$ אז הרצף ממשיך ו m עולה ב-1, לפי כן התקיימה השמורה והוא יוצא אם הרצף הארוך ביותר ב m או ב mm שמצאנו מההתחלה, לכן במשתנה m או mm יש את הרצף הארוך ביותר בתחום $K+2$ ...1.

אם התנאי לא מתקיים $A[K+1] > A[K+2]$ אז ב mm מוחזק המקסימום בין m ו mm, לפי כן ב- א התקיימה השמורה והיה שמור ב m או ב mm את המקסימום, לכן ב mm נשאר המקסימום, ויש בו את הרצף הארוך של איברים ממויינים בסדר עולה בתחום $K+2$...1.

מש"ל

__הוכחת נכונות:__

נוכיח בשמורה 1-n ,ושכן בסוף הלולאה במשתנה m או max
יש את הרצף הארוך ביותר של איברים ממויינים
מתחום 1,...,n , כי ממנו הוסק המקסימום מבין m או max,
ולכן חוזר האורך של הרצף המקסימלי של איברים ממויינים
כמספר טבעי.

```
ImpBubble(A)
 Lim <- n
 While lim > 1 do
   newlim <- 1
   for j=1 to lim-1 do
*  --->  if A[j] > A[j+1] then
         Swap (A[j], A[j+1])
         new lim = j
       endif
   end for
   lim <- new lim
** ---> end While
```

שאלה 4:

<u>טענת נכונות:</u>

האלגוריתם ImpBubble ממיין כהלכה מערך A, ומניח אין מספר אלה.

<u>שמורה מ*:</u>

$A[j] = \max\{A[1] \dots A[j]\}$

<u>הוכחת השמורה:</u>

נוכיח באינדוקציה על j

<u>בסיס:</u> $n=j=1$

$A[1]=\max\{A[1]\}$ $1 \dots 1$ ✓

נניח עבור $j=k$ שהשמורה מתקיימת

$A[K]=\max\{A[1] \dots A[K]\}$

לאחר איטרציה נוספת $j=k+1$ לפני כן התקיימה השמורה *

$A[K]>A[K+1]$ אז Swap $A[K]$ $A[K+1]$ אחרת $A[K+1] \geq A[K]$

בשני המקרים השמורה $A[K+1]=\max\{A[1] \dots A[K+1]\}$ מתקיימת.

מש"ל

<u>שמורה מ**:</u>

$A[lim] \leq \dots \leq A[N]$

<u>הוכחת שמורה **:</u>

<u>בסיס:</u> $lim=N$ לאחר איטרציה של While השמורה * מתקיימת

לכן $A[N]=\max\{A[1] \dots A[N]\}$ <= $A[N]$ הכי גדול

נניח ב $lim=K$ השמורה ** מתקיימת $A[N] \dots A[K]$

לאחר איטרציה נוספת $lim=K+1$ הלולאה for נאמרת

השמורה * התקיימה לכן $A[K+1]=\max\{A[1] \dots A[K+1]\}$ לכן השמורה **

התקיימה ב $j=K$ ו $A[K+1]$ הוא הכי גדול מ $A[K-1] \dots A[1]$

לכן $A[K+1] \leq A[K] \leq \dots A[N]$ ממויין כמספר עולה.

מש"ל

<u>הוכחת הנאה:</u>

לפי השמורה * 4 $line = 2$ במקומות n ... 1 נמצאים
n המספרים הממוינים במערך ומספר עוצר.

מש"ל.

ב- הוכחה שהאלגוריתם סופי.

מטלה 3 - אלגוריתמים   קרינה חיבבוב.

100

א. כתבו משפט נכונות לאלגוריתם:

האלגוריתם merge מקבל כקלט מערך A, B ממויין ומערך C, וממזג את A, B ממויינים למערך C.

```
merge (A, B, C)

A[m+1] ← ∞
B[n+1] ← ∞
ap ← 1
bp ← 1
cp ← 1

While cp < m+n+1 do

  if A[ap] ≤ B[bp] then

     C[cp] ← A[ap]
     ap ← ap+1

  else
     C[cp] ← B[bp]
     bp ← bp+1

(*) → end if
  cp ← cp+1

end While
```

שמורה:

m_1 האיבר המינימלי מבין שני המערכים A, B

m_i: האיבר המקסימלי מבין שני המערכים עד לנקודה i

$C[1] \le \dots \le C[cp] = m_1 \le \dots m_{cp}$

ap - מצביע לאיבר הבא במערך A שטרם הועבר ל-C.

bp - מצביע לאיבר הראשון במערך B שטרם הועבר ל-C

cp - מצביע למערך C.

אחרי i מעברים בלולאה, i האיברים הראשונים במערך ממויינים בסדר עולה. ומכילים את i האיברים המינימליים של A ו B.

הוכחת השמורה:

אינדוקציה על cp.

בסיס: cp = 1, נכנס ללולאה אם $A[ap] \le B[bp]$ אז ל $C[cp]$ נכנס $A[ap]$ האיבר המינימלי מ A ו B, אחרת: $A[ap] > B[bp]$ אז ל $C[cp]$ נכנס $B[bp]$, האיבר המינימלי מ A ו B.. ו cp וגם ap או bp גדלים בהתאמה.

נניח כי $cp = i \Leftarrow C[1] \le \dots C[i] = m_1 \dots m_i$

לאחר איטרציה נוספת, $cp+1 = i+1$, לפני שה התקיימה השמורה

$C[1] \le \dots \le C[i] = m_1 \dots m_i$, (ישנו שאלה השאלה אם $A[ap] \le B[bp]$

$C[i+1] = A[ap]$, ap+1 לכן עדיין מתקיימת השמורה $m_{i+1} = A[ap]$

לכן $C[1] \le \dots \le C[i+1] = m_1 \dots m_{i+1}$

אם התנאי לא מתקיים סימן ש $A[ap] > B[bp]$ $m_{i+1} = B[bp]$

$C[i+1] = B[bp]$ ו $bp{+}{+}$, לכן השמורה עדיין מתקיימת,

$$C[1] \dots \le C[i+1] = m_1 \le \dots m_{i+1}$$

מש"ל

<u>הוכחת סיום:</u>

לאחר האיטרציה $cp = m+n+1$ לא נכנס יותר ללולאה

וכי $cp[m+n]$ נמצא האיבר המקסימלי ביותר.

לפי השמורה * $C[1] \le \dots \le Cp[m+n]$.

נמצאים $m+n$ האיברים המינימליים של A ו B

בסדר עולה במערך C.

לכן קיבלנו מערך C ממוין שמכיל מיזוג של

מערכים A ו B.

מש"ל.

```
2. WAI (A,N)
   i ← 1
   While i ≤ N
      j ← i
      While A[j] = A[j+1]
         j++
      j++
      i ← j
   end While
```

<u>מה הסיבוכיות של הפונקציה?</u>

נוכיח שהסיבוכיות של הפונקציה היא $\Theta(n)$:

בלולאה החיצונית רץ מ i ל N. ה j מאותחל ב i כל עוד יש איברים שווים מוסיפים ה ++j. לפני סיום נוסף בלולאה החיצונית מאתחלים את i ל j.

בכל מקרה לכל היותר הלולאות ירוצו בסה"כ n פעמים. עדיין ש i מקבל את j וj מקבל את i

```
3. puzzle(A,N)
  for i←1 to N
      j← i+1
      While (A[j] != A[i] and j<N)
           A[j] ← A[i]
           j++
      end While
  end for.
```

<u>מהי הסיבוכיות של הפונקציה</u>:

נוכיח שהסיבוכיות של הפונקציה היא $\Theta(n)$:

1. נוכיח שהמספר הפעמים של הלולאה הפנימית מתבצעת הוא n-1 לכל היותר, ה j מאותחל ל $i+1$, בכל כל איטרציה אם מתקיים התנאי $A[j]!=A[i]$ וגם $j<N$ נכנס ללולאה, לכל היותר הוא ייכנס n-1 פעמים עד ש $N=j$ ויפסיק.

2. נוכיח שמספר הפעמים שהפונקציה נכנסת ללולאה החיצונית הוא לכל היותר N ; מתקדם כל פעם, ואין ירידה ב i, לכן לכל היותר N פעמים.

משל

$T(n) =$ זמן לולאה חיצוני + זמן לולאה פנימי

$n + n-1 = 2n+1 = O(n)$

זמן לולאה חיצוני

$n \Rightarrow \Omega(n)$

$\Leftarrow \Theta(n)$.

100

מטלה 4 אלגוריתמים
הקריה חופשית.

שאלה 1:

א.

```
MaxMin (A,n)
  if (n==2) {
      if(A[1]>A[2]) {
          y.min <- A[2]
          y.max <- A[1]
      }
      else {
          y.min <- A[1]
          y.max <- A[2]
      }
      return y
  }
  if (n==1) {
      y.max = y.min = A[1]
      return y;
  }
  y1 <- MaxMin (A[1... n/2], n/2)
  y2 <- MaxMin (A[n/2+1,.... n], n/2)
  y.max = max{y1.max, y2.max}
  y.min = min{y1.min, y2.min}
  return y;
```

ב. הוכחת נכונות:

האלגוריתם MaxMin מקבל כקלט מערך A ואורך n ומחזיר כפלט את המינימום והמקסימום.

נוכיח שהפונקציה MaxMin עוצרת: (באינדוקציה על n).

בסיס: $n=2$, נכנס ל if הראשון ומחזיר y. => הפונקציה עוצרת.

הנחת האינדוקציה: נניח שכאשר $n<k$ הפונקציה עוצרת.

נוכיח: עבור $n=k$ שהפונקציה עוצרת.

כש נכנסנו לקריאה הרקורסיבית ה 1 => הפונקציה עוצרת לפי הנחת האינדוקציה עבור $\frac{n}{2}<k$. כשנכנסנו לקריאה הרקורסיבית ה 2 => הפונקציה עוצרת עבור $\frac{n}{2}<k$ לפי הנחת האינד'. אין יותר קריאות רקורסיביות, לכן הפונקציה תמיד עוצרת.

מש"ל

נוכיח ש MaxMin עובדת נכון:

באינדוקציה על n.

בסיס: $n=2$. נכנס ל if ושואל מי יותר גדול, ומכניס אותו ל y.max, ואת השני (הקטן יותר) נכניס ל y.min.

הנחת האינדוקציה: עבור $k<n$, הפונקציה MinMax מחזירה ב y.max את האיבר המקסימלי וב y.min את האיבר המינימלי.

נוכיח: עבור $k=n$, בקריאה ה-1 תחזיר מינימום ב y1.min ומקסימום ב y1.max לפי הנחת האינדוקציה $\frac{n}{2}<n<k$,

בקריאה הרקורסיבית השנייה ב y2.max תחזור המקסימום וב y2.min יחזור המינימום, לפי הנחת האינדוקציה $\frac{n}{2}<n<k$.

לאחר הקריאות y.max מקבל את ה max מבין y1.max ל y2.max ו y.min מקבל את המינימום בין y1.min ל y2.min.

וה y מוחזר כש בתוכו y.max המקסימלי במערך, וב y.min המינימלי שבמערך.

מש"ל

ב. סיבוכיות:

$$T(n)=C_1+T\left(\frac{n}{2}\right)+T\left(\frac{n}{2}\right)$$

$$T(n)=C_1+2T\left(\frac{n}{2}\right)$$

$$a=2$$
$$b=2$$

$$T(n)=\Theta(n^{\log_2 2})=\Theta(n)$$

שאלה 2:

א. Find E(T):

```
node=T;
if(node==null)
    return
if((node.left==null) && (node.right==null)) {
    y.sum= node.info
    y.n=1
    y.E=0
}
A=FindE(node.left)
B=FindE(node.right)
y.sum= A.sum+B.sum+node.info
y.n= A.n+B.n +1
y.E= y.sum/y.n
return y
out
```

משפט נכונות:

הפונקציה findE מקבלת כקלט עץ בינארי T, ומחזירה כפלט את הממוצע של תת עץ השמאלי, ותת עץ ימני של T, ואת הסכום ומס' הצמתים שלו.

ב. סיבוכיות: נוסחה רקורסיבית היא n על היותר. O(n).

נניח שבקריאה הראשונה יש m צמתים => nodeleft יש m צמתים => שם node.right יש $n-m-1$ צמתים

=> סך הכל יש $\underbrace{n-m-1}_{\text{הקריאה 2}}+\underbrace{m}_{\text{הקריאה 1}}+\underbrace{1}_{\text{צומת השורש}} = n$"

סה"כ O(n).

נוכיח שהפונקציה findE עוצרת:

נוכיח באינדוקציה על n (גובה של T)

בסיס: n=0 , נכנס ל if הראשון ועושה return לכן עוצרת.
n=1 , נכנס ל if השני ויש return.

הנחה: הפונקציה עוצרת לכל א>n.

נוכיח: עבור n=א שהפונקציה עוצרת.

בקריאה הרקורסיבית הראשונה ששולחים את התת עץ השמאלי

גובה של תת עץ השמאלי של T הוא n-1 , לכן מתקיימת

הנחת האינדוקציה => הפונקציה עוצרת.

לאחר הקריאה הרקורסיבית השנייה, ששולחים את תת עץ הימני:

גובה תת עץ ימני < n , לכן מתקיימת הנחת האינדוקציה =>

הפונקציה עוצרת. לאחר 2 הקריאות האלה לא מתבצעות עוד

קריאות, לאחר מכן יש return , לכן הפונקציה תמיד עוצרת.

מש"ל.

נוכיח שהפונקציה findE מחזירה נכון:

נוכיח באינדוקציה על n (גובה של T)

בסיס: n=1 , נכנס ל if השני ממוצע של תתי עץ 0

מכיוון ש אין תתי עץ.

הנחה: עבור א>n , הפונקציה מחזירה נכון y.E (ממוצע)

של תתי העץ של T , ו ב mysum סכום של תתי עץ וב n.y מס' צמתים.

נוכיח: עבור n=א , בקריאה הרקורסיבית הראשונה אנחנו מקבלים

ב A1.sum את סכום כל הצמתים של תת עץ השמאלי

ואת מס' הצמתים בו ב A.n , ואת הממוצע של A.E

לפי הנחת האינדוקציה. בקריאה הרקורסיבית השנייה, נקבל

ב B.sum את סכום הצמתים של תת עץ ימני, ואת מס' הצמתים

בו ב B.n , ואת הממוצע ב B.E.

לאחר הקריאות מחברים את A.sum ו B.sum ו node.info ומכניסים ל mysum , ומחברים ב

A.n + B.n + 1 , ואז מחלקים את הממוצע של עץ T

ומחזירים את זה כתשובה.

מש"ל.

שאלה 3:

א) IsEven(A,n,m)

```
if (n==m)
    return A[n]%2
y1=IsEven (A,n, (n+m)/2)
y2=IsEven (A, (n+m)/2 +1 ,m)
return ((y1+y2)%2)
end IsEven.
```

משפט נכונות:

הפונקציה IsEven מקבלת כקלט מערך A אינדקס n,m ומחזירה כפלט אם המערך N $m..n$ זוגי=0, אי-זוגי=1.

נוכיח שהפונקציה IsEven עובדת נכון:

אינדוקציה על $m-n$

בסיס: $m-n=0 \Rightarrow m=n$, נכנס ל if הראשון ומחזיר את $A[n]\%2$ אם שארית 0 מספר זוגי, אם שארית 1 מספר אי זוגי.

הנחה: נניח כי עבור $k>m-n$ הפונקציה מחזירה 0 אם זוגי ו 1 אם אי-זוגי.

הוכחה: $m-n=k$, בקריאה הרקורסיבית הראשונה $m>\frac{n+m}{2}$ לכן $\frac{n+m}{2}-n < m-n$ לכן מתקיימת הנחת האינדוקציה וה y_1 חוזר 0 אם זוגי ו 1 אם אי-זוגי.

בקריאה הרקורסיבית השנייה $m \geq \frac{n+m}{2}+1$ לכן $m-\frac{n+m}{2}+1<k$ לכן מתקיימת הנחת האינדוקציה וה y_2 חוזר 0 אם זוגי ו 1 אם אי-זוגי

לאחר הקריאות, מחברים y_1+y_2 ועושים מודולו 2, אם חוזר זוגי + זוגי או זוגי, זוגי + אי-זוגי = אי-זוגי ואי-זוגי + אי-זוגי חוזר זוגי לכן הטענה מתקיימת. מש"ל

נוכיח שהפונקציה עוצרת נכון:

אינדוקציה על $n-m$:

בסיס: $n-m=0 \Rightarrow m=n$ נכנס ל if הראשון ישר return

בהצלחה הפונקציה עוצרת.

הנחת אינדוקציה: נניח כי ל $k<n-m$ הפונקציה עוצרת.

נוכיח: כי $k=n-m$ הפונקציה עוצרת.

הפונקציה נכנסת ל הקריאה הרקורסיבית הראשונה

$k > n - \frac{n+m}{2}$ לכן הנחת האינדוקציה מתקיימת, לכן הפונקציה

עוצרת. כשהפונקציה נכנסת לקריאה הרקורסיבית השנייה

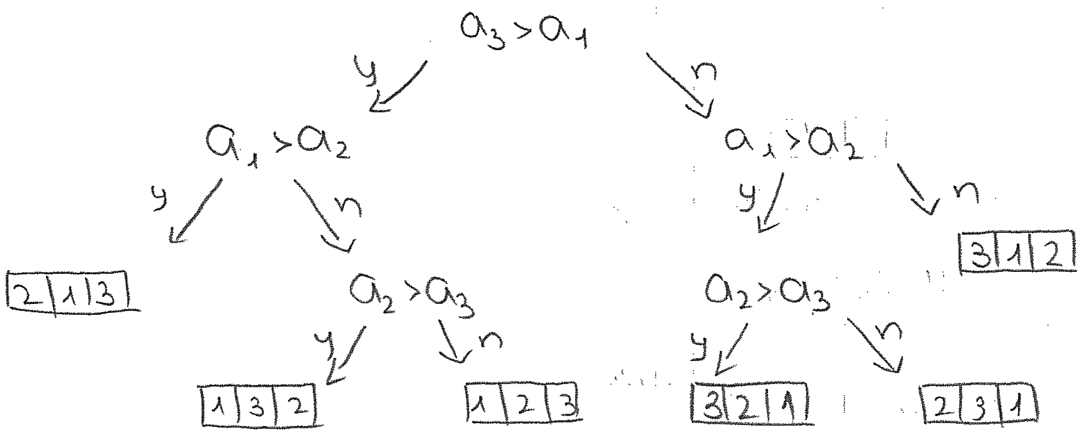
$k > \frac{n+m}{2} - m$, לכן הנחת האינדוקציה מתקיימת

ולכן הפונקציה עוצרת. מאחר שזה אין יותר קריאות רקורסיביות

ומוחזר מחרוזת לכן הפונקציה עוצרת.

מ״ש

שאלה 4:



$a_1 > a_3$

$a_2 > a_1$

$a_2 > a_1 > a_3$

23/02/16

אלגוריתמים א' תשע"ו - בחינת סיכום מועד א'

- בבחינה חמש שאלות, יש לפתור את כל השאלות.
- בחינה בחומר **סגור**.
- אין להשתמש במחשבונים.
- משך הבחינה 3.5 שעות – ללא הארכת זמן!
- **מרצה:** פרופ' עמוס ישראלי,
- **מתרגל:** מר נועם לוי.
- **שימו לב:** בשאלות 3-5 מותר להשתמש בכל האלגוריתמים שלמדנו בהרצאות ובתרגול ואין צורך לכתוב את הקוד שלהם, להוכיח את נכונותם או לנתח את הסיבוכיות שלהם. אם אחד או יותר ממבני הנתונים שאתם משתמשים בהם גדול במיוחד, ציינו זאת בפתרון השאלה.

בהצלחה!!!

<u>שאלה מס' 1 (20 נקודות)</u>

להלן הקוד של אלגוריתם **מיון מיזוג (Merge Sort)** :

$mergesort(A)$

  if $|A|=1$ **return** (A)

  $B_1 = mergesort(A[1,\lfloor n/2 \rfloor])$

  $B_2 = mergesort(A[\lfloor n/2 \rfloor+1,n])$

  $C \leftarrow merge(B_1,B_2)$

return (C)

פונקציית העזר $merge(B_1,B_2)$ מקבלת שני מערכים **ממוינים** ומבצעת מיזוג שלהם. עליכם לענות על השאלות הבאות :

1.1 נסחו משפט נכונות לאלגוריתם.

1.2 הוכיחו את המשפט שניסחתם. במהלך ההוכחה, אתם יכולים להשתמש בתכונות הפונקציה $merge$.

1.3 מהי סיבוכיות הזמן של האלגוריתם שהצגתם כפונקציה של גודל הקלט? נמקו תשובתכם.

<u>שאלה מס' 2 (20 נקודות)</u>

נתון קוד של אלגוריתם $quiz13$ המקבל כקלט מערך שאיבריו הם n מספרים ממשיים. האלגוריתם משתמש בשיטה בשם $indMax(A[i...n])$ אשר מקבלת כארגומנט תת מערך של מערך הקלט A. נתון שהשיטה $indMax(A[i...n])$ מחזירה את האינדקס של האיבר המכסימלי בתת המערך שבארגומנט, כאשר האינדקס המוחזר מתייחס למערך המקורי. להלן נתון קוד האלגוריתם :

$quiz13(A)$

 for $i=1$ **to** $n-1$

  $j \leftarrow indMax(A[i...n])$

  $swap(A[i],A[j])$

 end (for)

 $out(A)$

ענו על השאלות הבאות :

2.1 מה מחשב האלגוריתם $quiz13$?

2.2 נסחו משפט נכונות לאלגוריתם $quiz13$.

2.3 הוכיחו במדויק את נכונות המשפט שניסחתם.

2.4 נתון שסיבוכיות השיטה $indMax(A[i...n])$ היא לינארית ב- $n-i+1$. מהי סיבוכיות האלגוריתם $quiz13(A)$?

<u>שאלה מס' 3 (20 נקודות)</u>

נתון מערך A שאיבריו מספרים ממשיים. בשאלה זו עלייכם להציע אלגוריתם אשר מחשב את אורך תת המערך המקסימאלי (**<u>לא בהכרח רציף</u>**) שאיבריו מהווים סדרה מונוטונית עולה ממש (אין לחזור על איברים שווים).

<u>לדוגמה:</u>

במערך $[1,0,6,2,15,4,6,1,8]$ (קוראים משמאל לימין). הסדרה העולה הארוכה ביותר היא $(1,2,4,6,8)$ והאלגוריתם צריך להחזיר 5.

<u>הדרכה:</u>

1. אם אתם משתמשים במערכי עזר, הסבירו מהי הגדרת האיברים של כל מערך עזר.
2. הפתרון הנדרש הוא בסיבוכיות $\theta(n^2)$.

<u>שאלה מס' 4 (20 נקודות)</u>

מערך A שאיבריו הם מספרים ממשיים ושמספרם מתחלק במדויק ב-3, **<u>נקרא מערך שליש-ממוין</u>** אם שליש האיברים הקטנים ביותר מאוחסנים בסדר שרירותי בשליש התחתון של המערך (אינדקסים $1...n/3$), שליש האיברים הגדולים ביותר במערך מאוחסנים בסדר שרירותי בשליש העליון של המערך (אינדקסים $2n/3+1...n$) ושאר האיברים ממוינים בסדר עולה בשליש האמצעי של המערך (אינדקסים $n/3+1...2n/3$).

הוכיחו שלא ניתן לארגן את איבריו של המערך A למערך **שליש-ממוין**, תוך ביצוע פעולות השוואה והשמה בלבד על איבריו, בסיבוכיות נמוכה מ- $\theta(n\log n)$ פעולות.

<u>שאלה מס' 5 (20 נקודות)</u>

בשאלה זו עליכם להציע אלגוריתם, יעיל ככל האפשר הפותר את הבעיה הבאה:

<u>קלט</u>

מערך A שאיבריו הם מספרים טבעיים.

<u>פלט</u>

תשובה לשאלה האם מערך הקלט מכיל איבר השווה למכפלה של שני איברים מן המערך.

בהקשר זה עליכם לענות על השאלות הבאות:

5.1 תארו את האלגוריתם באופן מילולי

5.2 תארו את האלגוריתם בעזרת קוד דמה (Pseudo code).

5.3 נמקו בפרוט את נכונות האלגוריתם.

5.4 מהי סיבוכיות האלגוריתם? נמקו תשובתכם.

אלגוריתמים א' תשע"ד - בחינת סיכום מועד א'

30/01/14

- בבחינה ארבע שאלות, יש לפתור את כל השאלות.
- בחינה בחומר **סגור**.
- משך הבחינה 3.5 שעות – ללא הארכת זמן!
- מרצה: פרופ' עמוס ישראלי, מתרגל: מר נועם לוי.
- **שימו לב:** בשאלות 2-4 מותר להשתמש בכל האלגוריתמים שלמדנו בהרצאות ובתרגול ואין צורך לכתוב את הקוד שלהם, להוכיח את נכונותם או לנתח את הסיבוכיות שלהם.

בהצלחה!!!

שאלה מס' 1 (25 נקודות)

להלן נתון הקוד של אלגוריתם Order- Statistics (Select):

1.1 הוכיחו כי אם נתון מערך A ובו n מספרים, ומספר טבעי k המקיים $1 \le k \le n$, אזי האלגוריתם $Select(A,k)$ מחזיר את האיבר ה - k בגדלו במערך A. בהוכחה אתם יכולים להניח כי השיטה $Find(A,k)$ מחזירה את המספר ה - k בגדלו במערך A.

```
Select(A,k)
    if |A| ≤ 10 then
        return Find(A,k)
    p ← ChoosePivot(A)
    Partition A into A_{<p} and A_{≥p}
    m ← |A_{<p}|
    if k ≤ m return (Select(A_{<p}, k))
    if k > m return (Select(A_{≥p}, k − m))
```

```
ChoosePivot(A)
  for i = 1 to ⌈n/5⌉ do
    for j = 1 to 5 do
      B[j] ← A[(i−1)*5] + j
    endfor
    B ← sort(B)
    M[i] ← B[3]
  endfor
  return (Select(M, ⌊|M|/2⌋ + 1))
```

1.2 הוכיחו כי הערך המחזר על ידי השיטה $ChoosePivot(A)$ גדול או שווה מ- $3n/10$ מאיברי מערך הקלט A וקטן או שווה מ - $3n/10$ מאיברי מערך הקלט A.

<u>שאלה מס' 2 (25 נקודות)</u>

נתון מערך A בגודל n שאיבריו מספרים ממשיים שונים זה מזה. האיבר $A[i]$ נקרא **<u>איבר חריג</u>** אם מתקיים $A[i-1]>A[i]$ וגם $A[i+1]>A[i-1]$. המערך A נקרא **<u>כמעט ממוין</u>** אם הוא ממוין בסדר עולה מלבד כמה איברים חריגים שמספרם אינו עולה על $\log n$.

הוכיחו כי לא קיים אלגוריתם מבוסס השוואות בסיבוכיות קטנה מ $\Theta(n\log n)$ אשר מסדר את איבריו של המערך A כך שיהיה כמעט ממוין.

<u>שאלה מס' 3 (25 נקודות)</u>

נתון מערך דו ממדי A מסדר $n\times n$ של מספרים שלמים. נתון איבר $A[i,j]$. **<u>מעבר ישר</u>** הוא מעבר מן האיבר $A[i,j]$ אל האיבר $A[i+1,j]$. **<u>מעבר אלכסוני</u>** הוא מעבר מן האיבר $A[i,j]$ אל האיבר $A[i+1,j-1]$ או אל האיבר $A[i+1,j+1]$. **<u>מסלול תקין</u>** במערך A, הוא סדרה בת n איברים מן המערך A, כך שהאיבר הראשון, מן השורה הראשונה, השני מן השורה השנייה וכן הלאה. כמו כן נדרש כי המעבר מאיבר לאיבר במסלול יהיה מעבר ישר או מעבר אלכסוני וכי אסור כי יהיו שני מעברים אלכסוניים רצופים.

לדוגמא:

$$A[1,4],\underline{A[2,4],A[3,5]},A[4,5],\underline{A[5,5],A[6,6]},A[7,6]$$

הוא מסלול תקין בו המעברים האלכסוניים מודגשים בקו תחתי.

לעומת זאת:

$$A[1,4],\underline{A[2,4],A[3,5],A[4,4]},A[5,4],A[6,6],A[7,6]$$

אינו מסלול תקין, כי יש בו שני מעברים אלכסוניים רצופים המסומנים בקו תחתי והמעבר $A[5,4],A[6,6]$ אינו תקין לחלוטין.

בשאלה זו עליכם לכתוב אלגוריתם, **יעיל ככל האפשר**, לפתרון הבעיה הבאה:

<u>קלט</u>

מערך דו ממדי A מסדר $n\times n$ שאיבריו הם מספרים שלמים.

<u>פלט</u>

סכום המספרים המרבי המתקבל במסלול תקין במערך.

עליכם לענות על השאלות הבאות:

3.1 הציגו קוד דמה של אלגוריתם, **יעיל ככל האפשר**, הפותר את הבעיה.

שימו לב: הציון בסעיף זה תלוי בסיבוכיות האלגוריתם.

3.2 מהי סיבוכיות הזמן של האלגוריתם שהצגתם כפונקציה של גודל הקלט? נמקו תשובתכם.

<u>שאלה מס' 4 (25 נקודות)</u>

במשרד הבריאות מצויה מערך של רשומות המתארות את כל האזרחים בישראל. לכל אזרח מופיעה במערך רשומה ובה מספר זהות, משקל בזמן הלידה, משכורת נוכחית ועוד פרטים. לצורך מחקר על הקשר בין משקל האוכלוסייה למשכורת המשרד מעוניין לחשב לכל שני מספרים **שלמים** n ו m, המקיימים $1 \le n \le m \le 10$, את מספר האזרחים שמשקלם בזמן הלידה היה בין העשירון ה-n לעשירון ה-m (כולל העשירונים n ו-m) ומשכורתם הנוכחית היא בעשירון העליון במדינה.

בשאלה זו עליכם להציע אלגוריתם, $FatSalary(n,m)$, יעיל ככל האפשר, לפתרון הבעיה הבאה:

<u>הקלט:</u>

מערך של רשומות האזרחים כמתואר בתחילת השאלה.

<u>הפלט:</u>

מספר האזרחים שמשקלם בזמן הלידה היה בין העשירון ה-n לעשירון ה-m (כולל העשירונים nו-m) ומשכורתם הנוכחית היא בעשירון העליון במדינה.

נא ענו על השאלות הבאות:

4.1 תארו את האלגוריתם המוצע באופן מילולי.

4.2 כתבו קוד- דמה לאלגוריתם המוצע.

שימו לב: הציון בסעיפים 4.1 ו 4.2, תלוי בסיבוכיות האלגוריתם.

4.3 נמקו את נכונות האלגוריתם המוצע – אין צורך בהוכחה פורמלית.

4.4 מהי סיבוכיות הזמן של האלגוריתם? נמקו תשובתכם.

Netanya Academic College
16 Kibutz-Galuyot St.
Kiryat Yitzhak Rabin
Netanya 42365, ISRAEL

המכללה האקדמית נתניה
רח' קיבוץ גלויות 16
קרית יצחק רבין
נתניה 42365

אלגוריתמים א' תשע"ג - בחינת סיכום מועד א'

1/02/13

- **שימו לב:** את התשובות יש לכתוב **אך ורק במחברת הבחינה!!!**
- בבחינה ארבע שאלות, יש לפתור את כל השאלות.
- בחינה בחומר **סגור**.
- משך הבחינה 3.5 שעות – ללא הארכת זמן!
- מרצה: פרופ' עמוס ישראלי, מתרגל: מר נועם לוי.
- **שימו לב:** מותר להשתמש בכל האלגוריתמים שלמדנו בהרצאות ובתרגול ואין צורך לכתוב את הקוד שלהם, להוכיח את נכונותם או לנתח את הסיבוכיות שלהם.

בהצלחה!!!

Fax: (972-9) 8844439 :פקס   Tel: (972-9) 8607777 :טל

שאלה מס' 1 (25 נקודות)

להלן הקוד של אלגוריתם **מיון מיזוג (Merge Sort)** :

$mergesort(A)$

$\quad$ **if** $|A|=1$ **return** (A)

$\quad B_1 = mergesort(A[1,\lfloor n/2 \rfloor])$

$\quad B_2 = mergesort(A[\lfloor n/2 \rfloor+1,n])$

$\quad C \leftarrow merge(B_1,B_2)$

return (C)

פונקציית העזר $merge(B_1,B_2)$ מקבלת שני מערכים **ממוינים** ומבצעת מיזוג שלהם.

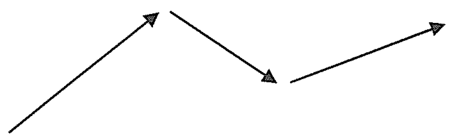
עליכם לענות על השאלות הבאות:

1.1 הוכיחו את נכונות האלגוריתם. במהלך ההוכחה, אתם יכולים להשתמש בתכונות הפונקציה Merge.

1.2 מהי סיבוכיות הזמן של האלגוריתם שהצגתם כפונקציה של גודל הקלט? נמקו תשובתכם.

שאלה מס' 2 (25 נקודות)

מערך A שאיבריו מספרים ממשיים שונים זה מזה נקרא **בתצורת ברק** אם גודל איבריו מתואר באופן סכמטי כך:



כלומר, אם קיימים אינדקסים m,k $1<m<k<n$ (ראש החץ הראשון וראש החץ האמצעי) אשר מקיימים:

1. לכל $1 \le i < j \le m$ $A[i]<A[j]$, כלומר איברי המערך A **עולים** מ-$A[1]$ ועד $A[m]$ (ראש החץ הראשון).
2. לכל $m \le i < j \le k$ $A[i]>A[j]$ כלומר איברי המערך A **יורדים** מ-$A[m]$ ועד $A[k]$ (ראש החץ השני).
3. לכל $k \le i < j \le n$ $A[i]<A[j]$ כלומר, איברי **שוב עולים** מ $A[k]$ ועד $A[n]$.

בשאלה זו **עליכם להוכיח** כי לא ניתן לארגן את איבריו של מערך נתון ובו n איברים שונים למערך בתצורת ברק ע"י פעולות השוואה והשמה בלבד, תוך ביצוע פחות מ- $\theta(n\log n)$ פעולות.

<u>שאלה מס' 3 (25 נקודות)</u>

בשאלה זו עליכם לכתוב אלגוריתם **יעיל ככל האפשר** לפתרון הבעיה הבאה:

<u>קלט</u>

מערך A ובו n מספרים טבעיים שונים.

<u>פלט</u>

תשובה לשאלה: "האם במערך הקלט A קיימים שני איברים שמכפלתם שווה לאיבר שלישי במערך, כלומר האם קיימים 3 אינדקסים שונים $1 \le i, j, k \le n$ עבורם מתקיים: $A[i] \cdot A[j] = A[k]$.

עליכם לענות על השאלות הבאות:

3.1 תארו את האלגוריתם המוצע באופן מילולי.

3.2 הציגו קוד דמה של אלגוריתם, יעיל ככל האפשר, הפותר את הבעיה.

שימו לב: הציון בסעיפים 3.1 ו 3.2, תלוי בסיבוכיות האלגוריתם.

3.3 נמקו את נכונות האלגוריתם שהצגתם.

3.4 מהי סיבוכיות הזמן של האלגוריתם שהצגתם כפונקציה של גודל הקלט? נמקו תשובתכם.

<u>שימו לב:</u> ניקוד מינימלי יינתן לאלגוריתם שהסיבוכיות שלו היא $\theta(n^3)$ (יש אלגוריתם בסיבוכיות $\theta(n^2)$).

<u>שאלה מס' 4 (25 נקודות)</u>

בשאלה זו עליכם להציע אלגוריתם, יעיל ככל האפשר לפתרון הבעיה הבאה:

<u>הקלט:</u>

מערך A ובו n מספרים ממשיים אשר **לפחות אחד מהם חיובי**.

<u>הפלט:</u>

סכום האיברים המרבי בתת מערך של A אשר אינו מכיל **שלושה איברים רצופים**.

לדוגמה: עבור הקלט $A=(10,9,8,11,12,3,17,4,13)$ אחד מתת המערכים המבוקשים הוא $B=(10,9,11,12,17,13)$.

4.1 תארו את האלגוריתם המוצע באופן מילולי.

4.2 כתבו קוד- דמה לאלגוריתם המוצע.

שימו לב: הציון בסעיפים 4.1 ו 4.2, תלוי בסיבוכיות האלגוריתם.

4.3 נמקו את נכונות האלגוריתם המוצע – אין צורך בהוכחה פורמלית.

4.4 מהי סיבוכיות הזמן של האלגוריתם? נמקו תשובתכם.