

מבני נתונים ומבוא אלגוריתמים

נושא 6

חסם תחתון לבעיית המיון, מיונים ליניאריים
Lower bound to sorting, linear sorts

בתוכנית

פרק 8 בספר הלימוד

- נענה על השאלה "כמה מהר אפשר למיין?"
נלמד מהו חסם תחתון לבעיה, ונמצא חסם תחתון
לבעיית המיין
- נכיר כמה מיונים ליניאריים:
 - מיון מנייה (counting sort)
 - מיון בסיס (radix sort)
 - מיון דלי (bucket sort)

כמה מהר אפשר למיין?

תזכורת להגדרת בעיית המיין:

- קלט: סדרת איברים (a_1, a_2, \dots, a_n) שמוגדר עליהם סדר.
- פלט: פרמוטציה $(a_1', a_2', \dots, a_n')$ של איברים אלו, כך ש- $a_1' \leq a_2' \leq \dots \leq a_n'$

אלגוריתם מיין:

- פתרון לבעיית המיין, כלומר: סדרת פעולות שמפיקה מכל קלט את הפלט הממוין.
- למשל: Insertion-Sort, Bubble-Sort, Merge-Sort, Heap-Sort, Quick-Sort, ...
- נשים לב שאף אחד מהנ"ל לא רץ במקרה הגרוע ב- $O(n \log n)$. האם זה מקרה?

- במילים אחרות: האם ידוע אלגוריתם שממיין n איברים בזמן $O(n \log n)$?
- ואם לא – האם מישהו עשוי להמציא כזה אלגוריתם יום אחד?
- ואם לא, איך מוכיחים שאף אלגוריתם (ידוע או שטרם התגלה) לא יכול לעשות זאת?



חסם תחתון למיון השוואות

משפט:

כל אלגוריתם מיון השוואות על סדרה באורך n מבצע במקרה הגרוע $\Omega(n \log n)$ השוואות בין איברי הקלט.

אלגוריתם מיון השוואות: (comparison sorting algorithm)

מיון, שבו החלטות על קביעת הסדר הממוין מבוססת רק על השוואות בין איברי הקלט.

בהינתן שני איברים a ו- b אפשר לבצע כל אחת מהבדיקות הבאות:

$$a < b, \quad a \leq b, \quad a = b, \quad a \geq b, \quad a > b$$

קביעת הסדר לא מתבססת על פעולות אחרות, כמו פעולות חשבון על איברים (ואף כלל לא מניחים שהאיברים הם מספרים).

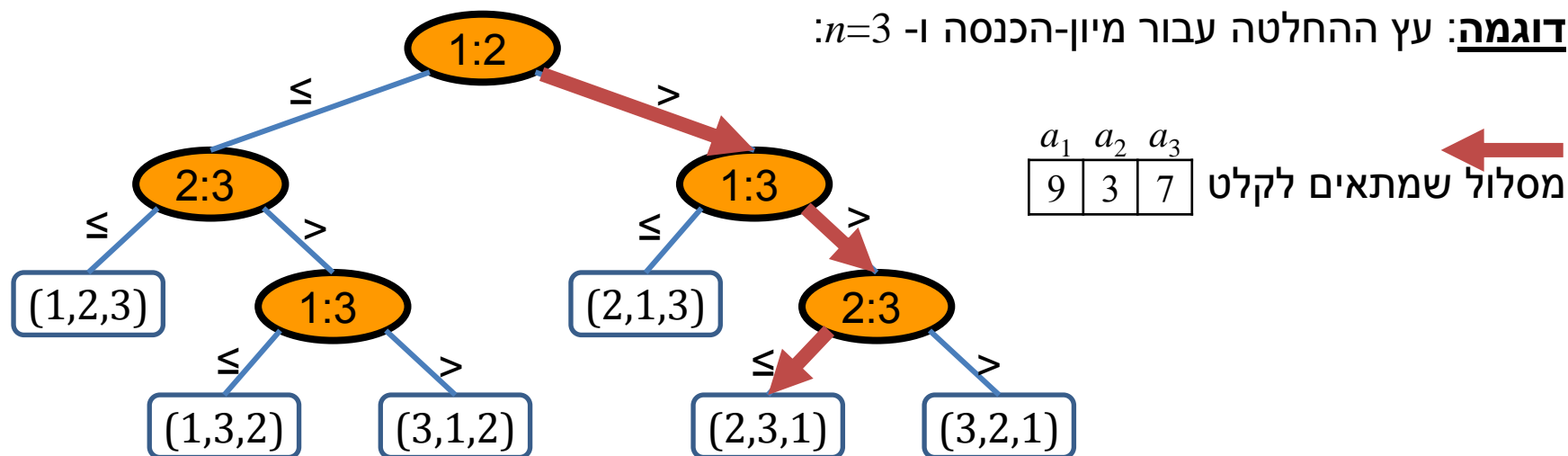
כל המיונים שראינו עד כה בקורס הם מיוני השוואות.

מסקנה: מיון השוואות לא יכול לרוץ ב- $O(n \log n)$.

לכן מיון ערימה ומיון מיזוג הם מיונים אופטימליים מבחינת סיבוכיות זמן במקרה הגרוע.

הוכחת החסם התחתון - מודל עץ ההחלטה

ניתן לתאר מיוני השוואות באופן מופשט באמצעות עצי החלטה (decision trees).
לכל אלגוריתם מיון השוואות, ולכל גודל קלט n מתאים עץ החלטה אחד.



- ריצה של האלגוריתם מתחילה מהשורש ומסתיימת באחד העלים.
- כל צומת פנימי מסומן $i : j$ ומייצג השוואה שהאלגוריתם עורך בין a_i ו- a_j .
- אם $a_i \leq a_j$ המשך ביצוע האלגוריתם מיוצג ע"י פניה שמאלה בעץ, אחרת ימינה.
- כל עלה מייצג תמורה של הקלט $\langle \pi(1), \pi(2), \dots, \pi(n) \rangle$ וכאשר מגיעים לעלה מתקיים

$$a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)}$$

הוכחת החסם התחתון - מודל עץ ההחלטה

כמות ההשוואות שעורך אלגוריתם במקרה הגרוע מיוצגת ע"י מסלול ארוך ביותר מהשורש לעלה כלשהו, כלומר ע"י גובה העץ.

נמצא אם כן חסם תחתון לגובה העץ (נסמנו h):

- כמה עלים לפחות יש בעץ החלטה עבור מיון של n איברים? ← לפחות $n!$
- כמה עלים לכל היותר יש בעץ בינארי בגובה h ? ← לכל היותר 2^h

$$n! \leq \text{מספר העלים} \leq 2^h \quad \text{ולכן:}$$

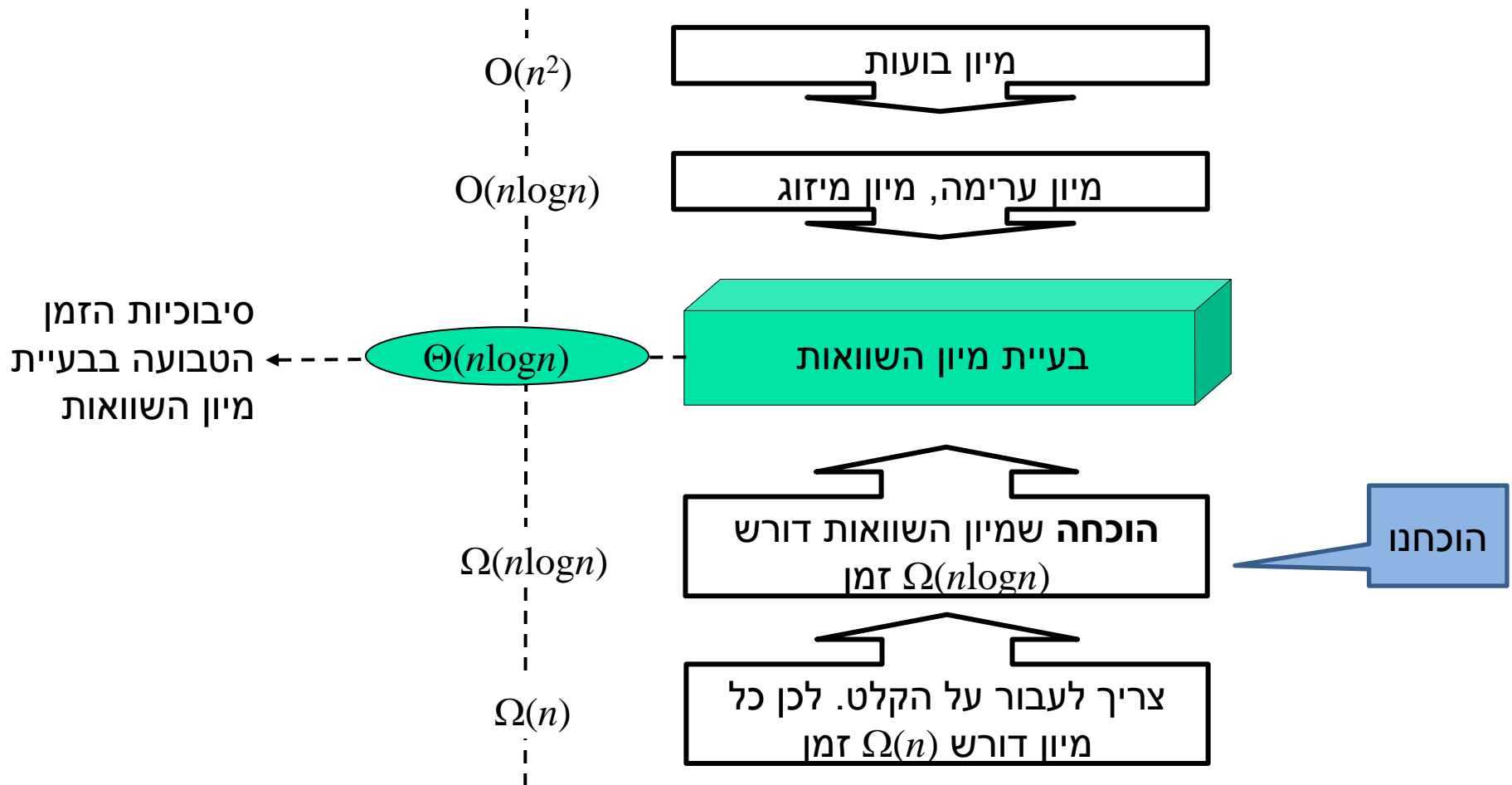
$$h \geq \log_2(n!) \quad \text{כלומר:}$$

$$h = \Omega(n \log n) \quad \text{וקיבלנו:}$$

בכך הוכחנו כי כל מיון השוואות על קלט בגודל n עורך במקרה הגרוע $\Omega(n \log n)$ השוואות, ולכן זמן הריצה שלו הוא $\Omega(n \log n)$.

הערה: עץ ההחלטה מתייחס אך ורק לפעולות ההשוואה בין איברי הקלט שעורך האלגוריתם. ככל הנראה ישנן פעולות נוספות כגון החלפת איברים, קידום אינדקסים בלולאות וכו'.

חסמים עליונים ותחתונים למיון השוואות



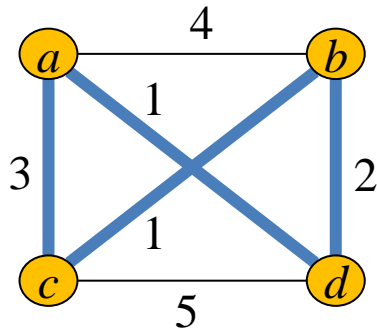
- גילוי של אלגוריתם מספק הוכחת חסם עליון לבעיה (ניתן לפתור אותה ב"לא יותר מאשר...")
- לעומת זאת, בהוכחת חסם תחתון לבעיה (לא ניתן לפתור אותה ב"פחות מאשר...") יש לקחת בחשבון את כל האלגוריתמים שפותרים אותה (קיימים ושרם נתגלו!).

פערים אלגוריתמיים

במקרים רבים ישנם "פערים אלגוריתמיים" בין החסם עליון הנמוך ביותר והחסם התחתון הגבוה ביותר שידועים לבעיה מסוימת.

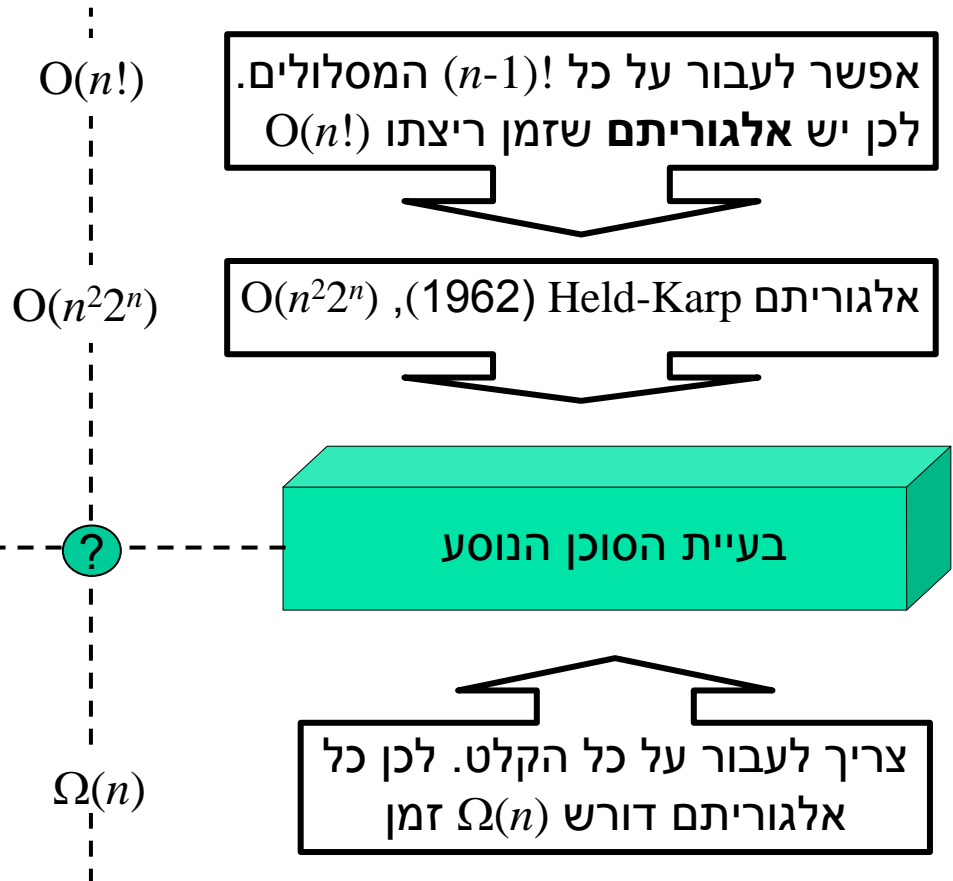
דוגמה: בעיית הסוכן הנוסע (travelling salesman problem).

נתונות n ערים ומרחקים ביניהן.
יש לחשב מסלול מעגלי קצר
ביותר שעובר בכל הערים.



שאלה פתוחה:

האם קיים אלגוריתם שרץ בזמן $O(c^n)$
עבור $c < 2$?

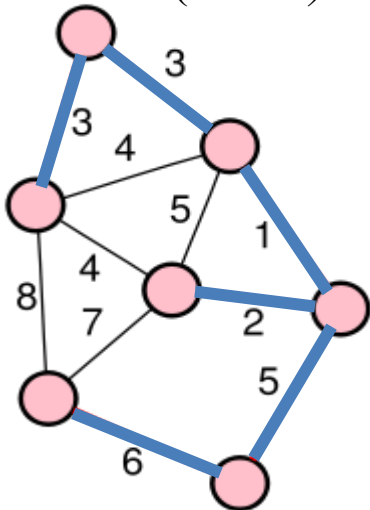


פערים אלגוריתמיים

דוגמה נוספת: בעיית עץ פורש מינימום (minimum spanning tree).

נתונים n מחשבים וביניהם m כבלים.
מהי רשת הכבלים הקצרה ביותר
שמשאירה את כולם מחוברים?

$$(m \geq n-1)$$



שאלה פתוחה:

האם קיים אלגוריתם שרץ בזמן $O(m)$?

$$O(m \log n)$$

$$O(m \log \log n)$$

$$** O(m \cdot \alpha(m, n))$$

$$\Omega(m)$$

אלגוריתם $O(m \log n)$: (1957) Prim
* אלגוריתם $O(m \log n)$: (1956) Kruskal

אלגוריתם משנת 1975 $O(m \log \log n)$

אלגוריתם משנת 1999 $O(m \cdot \alpha(m, n))$

בעיית עץ פורש מינימום

צריך לעבור על כל הקלט. לכן כל
אלגוריתם דורש $\Omega(n+m) = \Omega(m)$ זמן

* גרסה מסויימת של אלגוריתם Kruskal רצה בזמן $O(n^2)$, ואם $m = \Theta(n^2)$ הוא יעיל יותר.
** $\alpha(m, n)$ היא פונקצית אקרמן ההפוכה, פונקציה "כמעט קבועה".

מיונים ליניאריים

נכיר כעת 3 מיונים, שתחת הנחות מסוימות רצים בזמן ליניארי במקרה הגרוע:

- מיון מנייה
- מיון בסיס
- מיון דלי

כמובן, מיונים אלו אינם מיוני השוואות, והם מבצעים על איברי הקלט פעולות נוספות מלבד השוואות. לפיכך החסם התחתון שהוכחנו אינו חל עליהם.

מיון מנייה (Counting-Sort)

- מניח שהקלט הוא מערך A בגודל n שמפתחותיו הם מספרים שלמים בתחום $[1...k]$.
- משתמש במערך B , בגודל n אליו הוא כותב את הפלט, ובמערך עזר C בגודל k .

	1	2	3	4	5	6	7	8	9	10	11
A	7	1	3	1	2	4	5	3	2	4	3

מערך הקלט ($n = 11$)

	1	2	3	4	5	6	7	8	9	10
C										

מערך העזר ($k = 10$)

שלב א' – איפוס

1. **for** $i \leftarrow 1$ **to** k
2. **do** $C[i] \leftarrow 0$

	1	2	3	4	5	6	7	8	9	10
C	0	0	0	0	0	0	0	0	0	0

3. **for** $j \leftarrow 1$ **to** n
4. **do** $C[A[j]] \leftarrow C[A[j]] + 1$
5. ► $C[i]$ now contains the number of elements $= i$.

שלב ב' – מנייה

	1	2	3	4	5	6	7	8	9	10
C	2	2	3	2	1	0	1	0	0	0

6. **for** $i \leftarrow 2$ **to** k
7. **do** $C[i] \leftarrow C[i] + C[i-1]$
8. ► $C[i]$ now contains the number of elements $\leq i$.

שלב ג' – צבירה

	1	2	3	4	5	6	7	8	9	10
C	2	4	7	9	10	10	11	11	11	11

מיון מנייה (Counting-Sort)

שלב ד' – ייצור הפלט

9. **for** $j \leftarrow n$ **downto** 1
10. **do** $B[C[A[j]]] \leftarrow A[j]$
11. $C[A[j]] \leftarrow C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11
<i>A</i>	7	1	3	1	2	4	5	3	2	4	3

	1	2	3	4	5	6	7	8	9	10
<i>C</i>	2	4	7	9	10	10	11	11	11	11

	1	2	3	4	5	6	7	8	9	10	11
<i>B</i>							3				

מעריך הפלט



	1	2	3	4	5	6	7	8	9	10	11
<i>A</i>	7	1	3	1	2	4	5	3	2	4	3

	1	2	3	4	5	6	7	8	9	10
<i>C</i>	2	4	6	9	10	10	11	11	11	11

	1	2	3	4	5	6	7	8	9	10	11
<i>B</i>							3		4		

מעריך הפלט



מיון מנייה (Counting-Sort)

שלב ד' – ייצור הפלט

9. **for** $j \leftarrow n$ **downto** 1

10. **do** $B[C[A[j]]] \leftarrow A[j]$

11. $C[A[j]] \leftarrow C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11
A	7	1	3	1	2	4	5	3	2	4	3

	1	2	3	4	5	6	7	8	9	10
C	2	4	6	8	10	10	11	11	11	11

	1	2	3	4	5	6	7	8	9	10	11
B				2			3		4		

מערך הפלט



	1	2	3	4	5	6	7	8	9	10	11
A	7	1	3	1	2	4	5	3	2	4	3

	1	2	3	4	5	6	7	8	9	10
C	2	3	6	8	10	10	11	11	11	11

	1	2	3	4	5	6	7	8	9	10	11
B				2		3	3		4		

מערך הפלט

⋮



	1	2	3	4	5	6	7	8	9	10	11
B	1	1	2	2	3	3	3	4	4	5	7

מערך הפלט

מיון מנייה (Counting-Sort)

Counting-Sort (A, B, k)

1. **for** $i \leftarrow 1$ **to** k
2. $C[i] \leftarrow 0$
3. **for** $j \leftarrow 1$ **to** n
4. $C[A[j]] \leftarrow C[A[j]] + 1$
5. ► $C[i]$ now contains $|\{\text{elements} = i\}|$
6. **for** $i \leftarrow 2$ **to** k
7. $C[i] \leftarrow C[i] + C[i-1]$
8. ► $C[i]$ now contains $|\{\text{elements} \leq i\}|$
9. **for** $j \leftarrow n$ **downto** 1
10. $B[C[A[j]]] \leftarrow A[j]$
11. $C[A[j]] \leftarrow C[A[j]] - 1$

• רץ בזמן $\Theta(n + k)$.

• כאשר $k = O(n)$ זמן הריצה הוא $\Theta(n)$.

• סיבוכיות זיכרון נוסף $\Theta(n + k)$.

• לא מבצע כלל השוואות בין איברי הקלט!

• שאלה: בשלב ייצור הפלט, מדוע עוברים על הקלט בסדר הפוך?

• שאלה: אחרי שלב ב' (המנייה), אפשר כבר לייצר את הפלט (יודעים כמה פעמים מופיע

כל מספר). מדוע לא עושים זאת?

מיון בסיס (Radix-Sort)

- מניח שהקלט הוא מערך A בגודל n שמפתחותיו איברים הניתנים לייצוג כמספרים שלמים בני d ספרות בבסיס b כלשהו.

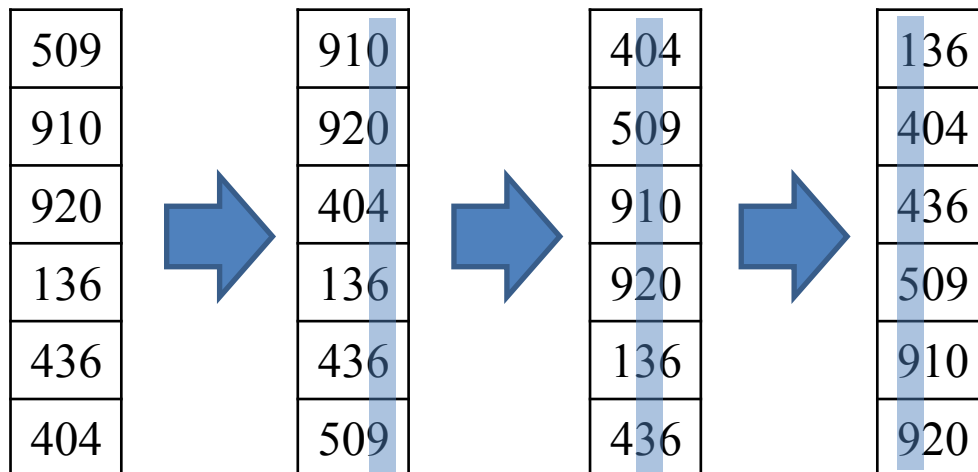
למשל: בסיס בינרי (0-1), עשרוני (0-9), הקסאדצימלי (0-9,A-F), בסיס א"ב כלשהו (A-Z)

האלגוריתם

ממיון בשלבים:

- החל בספרה הפחות משמעותית (LSD) ועד לספרה היותר משמעותית (MSD).

- בכל שלב מפעיל מיון יציב (למשל מיון מנייה).



מיון בסיס (Radix-Sort)

שאלות:

- מדוע נדרש בכל שלב מיון יציב?

- מדוע למיין מה- LSD אל ה- MSD ולא להיפך?

סיבוכיות זמן

ישנם d שלבים.

אם בכל שלב נשתמש במיון מנייה, סיבוכיות כל שלב תהיה $\Theta(n + b)$.

סה"כ: $\Theta(d(n + b))$.

כאשר d קבוע, ו- $b = O(n)$ מיון בסיס רץ בזמן ליניארי.

מיון בסיס (Radix-Sort)

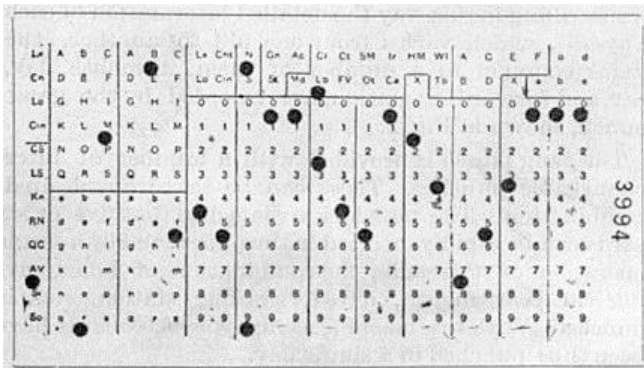
קצת היסטוריה

מיון בסיס היה האלגוריתם ששימש את ממיינות הכרטיסים מסוף המאה ה-19 ועד שנות ה-70 של המאה ה-20.

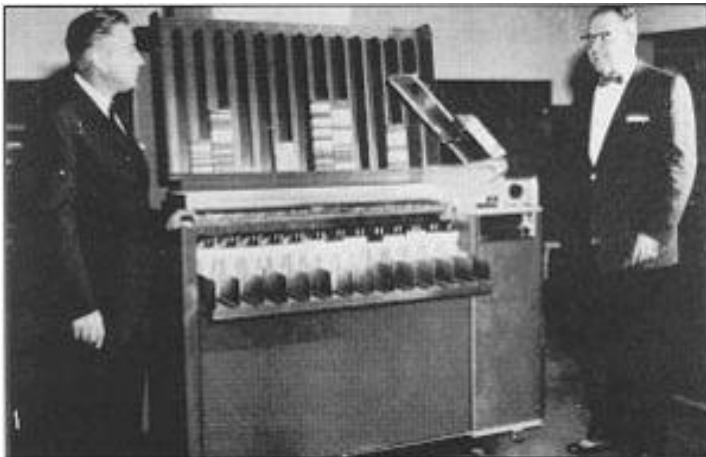
הקלט לממיינת כרטיסים היה כרטיס ניקוב, אותו המציא **Herman Hollerith** ב-1890, לטובת מפקד האוכלוסין

האמריקני:

- מספר עמודות, כשבכל אחת מספר ערכים עבור ספרות ואותיות
- כל "תא" יכול להיות מנוקב או לא מנוקב



ממיינת הכרטיסים היתה בודקת כל פעם עמודה אחת, ומפזרת את הכרטיסים לתאים לפי התא המנוקב.



מיון לפי מספר עמודות נעשה בשיטת מיון בסיס:

מיון חוזר לפי עמודות, מהעמודה הפחות משמעותית,

אל העמודה היותר משמעותית.

Hollerith הקים עסק פרטי, שבשנת 1924 הפך ל-IBM.



מיון דלי (Bucket-Sort)

- מניח שהקלט הוא מערך A בגודל n שמפתחותיו הם מספרים שלמים בין 1 ל- k .

הרעיון

שלב 1: נפזר את n המפתחות ל- k "דליים" (כל דלי הוא רשימה מקושרת)

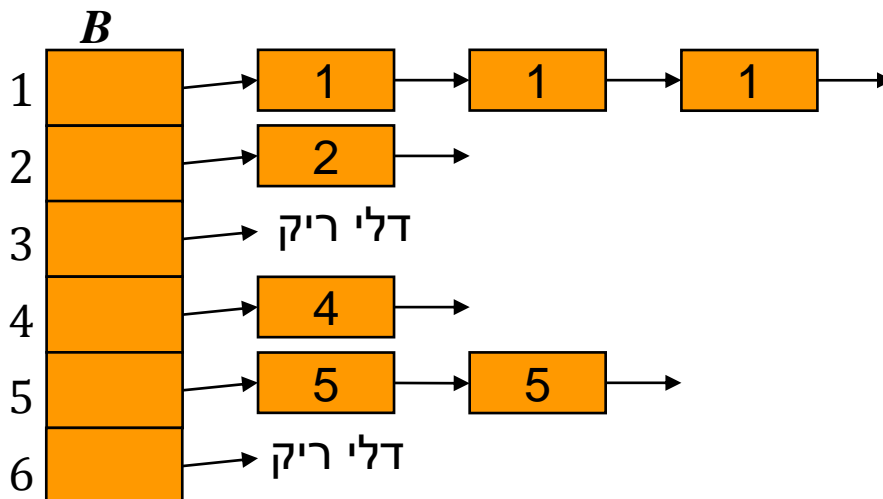
מערך הקלט

A

1
5
2
1
5
1
4

$n=7$

$k=6$



שלב 2: נשרשר את הדליים.



- איך אפשר לדאוג שהמיון יהיה יציב?

להכניס איברים ל**סוף** הרשימות (צריך בשביל זה גם מצביע לסוף כל רשימה)

מיון דלי (Bucket-Sort)

Bucket-Sort(A, n)

1. **for** $i \leftarrow 1$ **to** n
2. insert $A[i]$ into list $B[A[i]]$
3. concatenate the lists $B[1], B[2], \dots, B[n]$ together in order

סיבוכיות זמן

- פיזור האיברים לדליים לוקח סה"כ $\Theta(n)$ (הכנסה לרשימה ב- $\Theta(1)$).
- שרשור הרשימות לוקח $\Theta(n+k)$ בלי מצביעים לסוף הרשימות, או $\Theta(k)$ עם מצביעים לסוף הרשימות

סה"כ: $\Theta(n+k)$.

מיון דלי המורחב

- מניח שהקלט הוא מערך A בגודל n המורכב ממספרים ממשיים בתחום $[0,1)$ שהם בעלי התפלגות אחידה בתחום זה.
- התפלגות אחידה: הסיכוי להופעת מספר בתוך תת-קטע נתון פרופורציונית לאורכו למשל, הסיכוי להופעת קלט בתת-קטע $(\frac{1}{4}, \frac{1}{2}]$ הוא $\frac{1}{4}$.
- אם המספרים מתפלגים בצורה אחידה בקטע $[a, b)$ עבור $a < b$ כלשהם, אפשר "לנרמל" אותם. עבור $x \in [a, b)$ נשתמש בנוסחה: $(x-a)/(b-a)$.

הרעיון

- נפזר את n המפתחות ל- n "דליים", שכל אחד

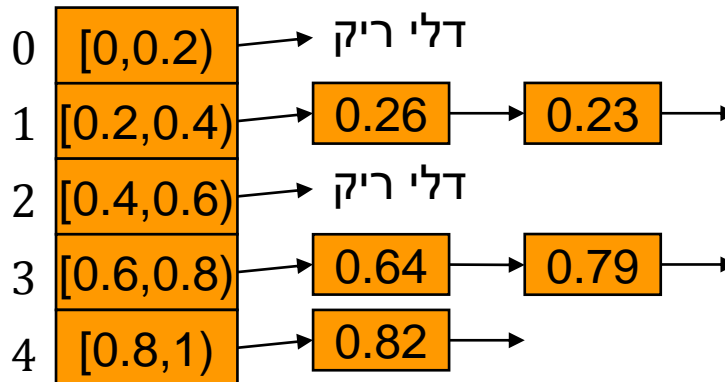
מהם "מכסה" תת-קטע שגודלו $1/n$.

(כל דלי הוא רשימה מקושרת)

A[1..5]

B[0..4]

0.79
0.23
0.26
0.64
0.89



- במיון כל דלי, ונשרשר את הדליים הממוינים.

מיון דלי המורחב

Bucket-Sort(A, n)

1. **for** $i \leftarrow 1$ **to** n
2. insert $A[i]$ into list $B[\lfloor nA[i] \rfloor]$
3. **for** $i \leftarrow 0$ **to** $n-1$
4. sort list $B[i]$ using (*e.g.*) insertion sort
5. concatenate the lists $B[0], B[1], \dots, B[n-1]$ together in order

מכיוון שהתפלגות המפתחות אחידה, אנו מצפים למצוא מספר קבוע של מפתחות בכל דלי,
ולכן מיון כל דלי ידרוש $\Theta(1)$ זמן.

סיבוכיות הזמן הממוצעת של מיון דלי המורחב היא $\Theta(n)$.
(הוכחה בספר הלימוד, עמודים 146-147)

שאלות חזרה

1. החסם התחתון למיין השוואות שהוכחנו מתייחס למקרה הגרוע. במקרה הטוב אפשר כמובן למיין בזמן $O(n \log n)$. הראו אלגוריתם מיין השוואות שרץ בזמן ליניארי במקרה הטוב.
2. בהוכחת החסם התחתון למיין השוואות, צוין כי בעץ ההחלטה שמתאים לקלט בגודל n יש לפחות $n!$ עלים. נסו לחשוב מדוע עלולים להיות יותר מ- $n!$ עלים בעץ החלטה ומתי זה קורה.
3. במונחים של Θ , מהו גובה עץ ההחלטה של מיין בועות עבור קלט בגודל n ? ושל מיין ערימה?
4. מהי סיבוכיות הזמן במקרה הגרוע של מיין דלי המורחב (ומהו המקרה הגרוע)?
5. איך ממיינים בזמן ליניארי מערך של אותיות באנגלית?
6. במיין מנייה, אחרי שלב ב' (המנייה), אפשר כבר לייצר את הפלט (יודעים כמה פעמים מופיע כל מספר). מדוע לא עושים זאת? (התשובה בשקפים)

תשובות לשאלות חזרה

1. האלגוריתם יבדוק אם המערך ממוין, ואם כן, פשוט יחזיר אותו, אחרת יפעיל את מיון ערימה (למשל):

Linear-Best-Case-Sort(A, n)

```
1. for  $i \leftarrow 1$  to  $n-1$ 
2.   if  $A[i] > A[i+1]$ 
3.     return Heap-Sort( $A, n$ )
```

גם מיון הכנסה רץ בזמן ליניארי במקרה הטוב.

2. אם יש יותר מ- $n!$ עלים, אז ישנה השוואה מיותרת שמבצע האלגוריתם.

3. מיון בועות מבצע במקרה הגרוע $\Theta(n^2)$ השוואות, וזהו גובה עץ ההחלטה המתאים. מיון ערימה מבצע במקרה הגרוע $\Theta(n \log n)$ השוואות, וזהו גובה עץ ההחלטה המתאים.

4. במקרה הגרוע, כל האיברים נכנסים לאותו דלי, כלומר נוצר דלי אחד עם n איברים. מיונו של דלי זה באמצעות מיון הכנסה לוקח $\Theta(n^2)$ (אפשר כמובן לשפר ל- $\Theta(n \log n)$, אם נשתמש במיון ערימה).

5. מיון מנייה למשל יעבוד בזמן ליניארי, כאשר כל אות מיוצגת ע"י מספר בין 1 ל- 26.

תרגילים

1. א. הוכיחו שכל אלגוריתם מיון השוואות הממין מערך באורך 5 חייב לבצע (במקרה הגרוע) לפחות 7 השוואות.
ב. כמה השוואות מבצעים אלגוריתמי ההשוואות שהכרנו בקורס על מערך באורך 5, במקרה הגרוע ובמקרה הטוב?

ג. אתגר: כיתבו אלגוריתם מיון השוואות המבצע 7 השוואות במקרה הגרוע על מערך באורך 5.

2. פרופסור כלשהו במחלקה כלשהי במכללה כלשהי בצפון הארץ טוען שלאחר שנות מחקר רבות, מצא אלגוריתם ליניארי שמקבל ערימה בת n איברים, ומדפיס את איבריה ממוינים.
האם הייתם נותנים לפרופסור ללמד מבני נתונים? אם כן – הראו אלגוריתם כמו זה שמציע הפרופסור. אם לא – הוכיחו כי טענתו לא יכולה להיות נכונה.

3. **פלינדרום** (Palindrome) באורך n הוא סדרת תווים a_1, a_2, \dots, a_n המקיימת $a_i = a_j$ לכל $i = n-j+1$, עבור $1 \leq i, j \leq n$.

נתונה סדרה באורך n של מספרים שלמים בין 1 ל- $3n$, וידוע שכל מספר מופיע כמות זוגית של פעמים. האם ניתן להפוך את הסדרה הנתונה לפלינדרום באורך n , שמכיל את אותם מספרים בדיוק, בזמן ריצה $O(n \log n)$? הראו כיצד או הוכיחו כי לא ניתן.

4. נתונים n מספרים שלמים בתחום $[0, n^2-1]$. הציעו דרך יעילה למינם.

רמז: מהו המספר המקסימלי בן שתי ספרות בבסיס n ?

5. בעיית הוקטור הפרבולי

בבעיית הוקטור הפרבולי נתון מערך בגודל n של שלמים שונים (אין חסם על גודלם) וצריך לסדר מחדש את איברי המערך כך שיתקיים:

• לכל $1 < i \leq \lceil n/2 \rceil$ מתקיים $A[i] \geq A[i-1]$

• לכל $\lceil n/2 \rceil + 1 \leq i < n$ מתקיים $A[i] \geq A[i+1]$

מערך שמסודר באופן הנ"ל נקרא וקטור פרבולי.

לדוגמה, המערך הבא הינו וקטור פרבולי: $\langle 3, 4, 12, 9, 8, 6 \rangle$

א. תנו חסם תחתון (במונחים של Ω) לסיבוכיות הזמן הדרושה לפתרון בעיית הוקטור הפרבולי.

ב. הציגו אלגוריתם אופטימלי (הן מבחינת סיבוכיות זמן והן מבחינת סיבוכיות מקום) לפתרון בעיית הוקטור הפרבולי.

פתרון 1

א. הוכחנו כי גובה עץ החלטה של מיון השוואות על מערך באורך n הוא לפחות $\log_2(n!)$.

$$h \geq \log_2(5!) \approx 6.9 \quad \text{נציב } n=5 \text{ ונקבל:}$$

גובה הוא מספר שלם, לכן גובה של עץ החלטה כנ"ל הוא לפחות 7.

מכיוון שגובה עץ ההחלטה מייצג את המסלול הארוך ביותר מהשורש לעלה, זהו גם מספר ההשוואות במקרה הגרוע שיבצע האלגוריתם.

ב.

מיון	מקרה גרוע	מקרה טוב
מיון הכנסה	10	4
מיון מיזוג	8	5
מיון מהיר	10	6
מיון ערימה	12	9

פתרון 2

טענתו של הפרופסור אינה נכונה.

נניח בשלילה שקיים אלגוריתם, נקרא לו Alg, שמקבל ערימה ומדפיס את איבריה ממוינים בזמן ליניארי בגודל הערימה.

נבדוק האם בעזרת אלגוריתם הנ"ל ניתן למיין כל מערך. ניקח מערך כלשהו, נבנה ממנו ערימה בזמן ליניארי באמצעות Build-Heap, ואז נקרא לאלגוריתם Alg הנ"ל עם הערימה שבנינו כקלט. קיבלנו שניתן למיין מערך בזמן ליניארי. זו כמובן סתירה לחסם התחתון שלמדנו לבעיית המיין.

פתרון 3

מיון מנייה (או מיון דלי) על הסדרה הנתונה ירוץ בזמן $\Theta(n+k) = \Theta(n+3n) = \Theta(n)$.

אחרי המיון, נבצע מעבר אחד על הסדרה, ונמקם איבר בעל אינדקס זוגי במקום הפנוי הבא מההתחלה, ואיבר בעל אינדקס אי-זוגי במקום הפנוי הבא מהסוף.

נבחן כמה אסטרטגיות:

- (1) מיון מבוסס השוואות ייקח $\Omega(n \log n)$.
- (2) Counting Sort ייקח $O(n+k) = O(n+n^2) = O(n^2)$.
- (3) - נעביר כל מספר לבסיס n . כך כל מספר יהיה בעל 2 ספרות לכל היותר.
 - סיבוכיות $O(n)$ (בהנחה שחלוקה ושארית רצות בזמן קבוע)
 - כעת נפעיל Radix-Sort על המספרים.
 - סיבוכיות $O(d \cdot (n+b)) = O(2(n+n)) = O(n)$.

הפתרון השלישי הוא העדיף (והוא אופטימלי מבחינת סדר גודל – מדוע?).

א. נוכיח כי החסם התחתון לבעיה הוא $\Omega(n \log n)$.

נניח בשלילה, כי קיים אלגוריתם שהופך מערך נתון בגודל n לוקטור פרבולי, ורץ בזמן $o(n \log n)$ (o קטנה). נקרא לאלגוריתם זה PV.

נראה כעת שמהנחת השלילה נובע שקיים אלגוריתם מיון, שרץ בזמן $o(n \log n)$:
אלגוריתם המיון (מקבל מערך A בגודל n):

1. הרץ את PV על המערך A // נוצר וקטור פרבולי
2. הפוך את סדר האיברים בחצי השני של A // כעת כל מחצית ממוינת
3. מזג את שני החצאים של A . // כעת A ממוין

אלגוריתם מיון זה רץ בזמן $o(n \log n) + \Theta(n) + \Theta(n) = o(n \log n)$
 וזו כמובן סתירה לחסם התחתון למיון השוואות שלמדנו. לכן ההנחה בשלילה שגויה.

ב. מיון ערמה, ואח"כ הפיכת סדר האיברים בחצי המערך השמאלי.

סיבוכיות הזמן היא $\Theta(n \log n)$, ואילו סיבוכיות המקום הנוסף היא $\Theta(1)$.