הוכן ע"י אמיר רובינשטיין

מבני נתונים ומבוא לאלגוריתמים

נושא 1

מבוא, סיבוכיות של אלגוריתמים
Introduction, complexity of algorithms

בתוכנית

פרקים 1-3 בספר הלימוד

- נכיר כמה מושגים בסיסיים בתחום האלגוריתמים
- נפגוש שוב כמה אלגוריתמים מוכרים
- נלמד לנתח סיבוכיות (זמן בעיקר) של אלגוריתמים,
  ונעזר לשם כך בסימונים: Θ, O, Ω, o, ω

מבוא ומושגים בסיסיים

<u>בעיה חישובית (computational problem)</u>

התאמה של קבוצת קלט (input) לקבוצת פלט (output).

דוגמאות:

קלט	פלט
מספרים שלמים	לכל מספר, ערכו המוחלט
מערכים של מספרים ממשיים	המערך כאשר הוא ממוין
מילים באנגלית	לכל מילה קבוצת התמורות שלה

<u>אלגוריתם (algorithm)</u>

סדרה של חישובים, אשר מייצרת לכל קלט חוקי פלט מתאים.
אלגוריתם הוא למעשה פתרון לבעיה חישובית.

<u>מהם החישובים המותרים</u>? תלוי בהקשר.

ישנן רמות שונות של רזולוציה:

- פעולות על סיביות (bits)
- חיבור, חיסור, כפל, חילוק, פנייה למשתנה (קריאה/כתיבה), השוואה,...
- מיון מערך, ...

מבחינתנו בד"כ אלו יהיו ה"פעולות היסודיות"

מבוא ומושגים בסיסיים

בעיית עוגת התפוחים

קלט: 2½ כוסות קמח, 5 תפוחים, 3 ביצי חופש, ½ כוס סוכר,...

פלט: עוגת תפוחים.

אלגוריתם שפותר את הבעיה:

רמת פירוט גבוהה מדיי :

- פתח את המגירה.
- הוצא את שקית הקמח מהמגירה.
- פתח את שקית הקמח.
- ...

רמת פירוט נמוכה מדיי :

- הכן מהמצרכים עוגת תפוחים

מבוא ומושגים בסיסיים

מימוש של אלגוריתם (implementation)
תכנית מחשב, ייצוג מדויק של אלגוריתם בשפת תכנות כלשהי.

הרצה של אלגוריתם (execution)
מתן פקודה **למחשבים ספציפיים** לבצע את ההוראות המופיעות במימוש של האלגוריתם.

אילו אלגוריתמים אתם מכירים? אילו בעיות הם פותרים?

<u>חיפוש במערך ממוין</u>

<u>דוגמה: בעיית החיפוש במערך ממויין</u>

<u>הגדרת הבעייה</u>:

<u>קלט</u>: מערך ממוין A של n איברים, וערך נוסף כלשהו key.

<u>פלט</u>: אינדקס של A שהמפתח שלו הוא key, אם קיים כזה.

אלגוריתם טריוויאלי הפותר את הבעיה (חיפוש ליניארי):

```
Linear-Search1(A, n, key)
1.   for  i ← 1 to n
2.        if  A[i] = key
3.             return  i
4.   return  Nil
```

<u>פסאודו-קוד</u> <u>(pseudo-code)</u> הוא "שפה" לתיאור אלגוריתמים המאפשרת להתרכז במבנה האלגוריתם, מבלי להתחשב במאפיינים הייחודיים של כל שפת תכנות.
פסאודו-קוד לא ניתן להריץ על מחשב, אבל הוא צריך להיות כתוב כך שניתן לתרגמו בקלות לשפת תכנות כלשהי (פירוט בעמ' 16-17 בספר).

<u>הערה</u>: מאותן סיבות, האינדקס הראשון של מערך בקורס שלנו יהיה בד"כ 1 ולא 0.

חיפוש במערך ממויין

שיפור 1:

```
Linear-Search2(A, n, key)
    i ← 1
    while  i ≤ n   and   A[i] ≤ key
        if  A[i] = key
            return  i
        i ← i+1
    return  Nil
```

שיפור 2 - חיפוש בינארי:

```
Binary-Search(A, p, q, key)        ► p and q are left and right boundaries
    while p ≤ q                    ► sub-array to search not empty
        mid ← ⌊(p + q) / 2⌋
        k ← A[mid]
        if  key = k
            return mid             ► found!
        else if key < k
                q ← mid – 1
            else  p ← mid + 1
    return  Nil                    ► not found
```

מדדים להערכת יעילותם של אלגוריתמים

הערכת אלגוריתמים והשוואה ביניהם ניתנת לביצוע במישורים שונים:

- עד כמה האלגוריתם קל להבנה?
- עד כמה האלגוריתם קל למימוש?
- עד כמה קל להוכיח נכונות האלגוריתם?
- אורך המימוש
- "אלגנטיות"
- סיבוכיות זמן ריצה – כמות פעולות שמבצע האלגוריתם
- סיבוכיות זיכרון – כמות משאבי זיכרון המחשב הדרושים
- ועוד...

אנו נתמקד בעיקר בסיבוכיות זמן וזיכרון, אבל תמיד נשאף לשפר את האלגוריתמים שלנו בכל המדדים (למשל, תמיד נעדיף אלגוריתם "אלגנטי" יותר, אם ביתר המדדים אין הבדל...).

<u>ניתוח סיבוכיות זמן</u>

כאשר מנתחים את סיבוכיות זמן הריצה של אלגוריתם, למעשה סופרים את
<u>כמות הפעולות היסודיות</u> שהוא מבצע, כתלות <u>בגודל הקלט</u> שלו.

- מדוע סופרים פעולות ולא זמן?

כדי להתעלם ממאפיינים של מחשב ספציפי (למשל מהירות מעבד, מספר מחזורי שעון לפעולות יסוד).

נסכים כי כל פעולה יסודית (חיבור, כפל, השוואה, העתקה וכו') דורשת <u>מספר קבוע</u> של מחזורי שעון, ומעתה נתעלם מקבועים אלו (או ניקח חסם עליון על כולם).

- מדוע כתלות בגודל הקלט?

כי באופן כללי ככל שגודל הקלט עולה, גם זמן הריצה עולה.

למשל, מיון מערך באורך 100 ככל הנראה ידרוש יותר פעולות ממערך באורך 10.

המושג גודל הקלט תלוי באופי הקלט של הבעיה.
למשל:
- מערך: מספר התאים במערך
- מספר שלם: כמות הביטים שמייצגים אותו
- מטריצה: מספר השורות ומספר העמודות (או מספר התאים)
- גרף: מספר הצמתים ומספר הקשתות

הסימון Θ

```
Linear-Search1(A, n, key)
1.    for i ← 1 to n
2.        if A[i] = key
3.            return i
4.    return Nil
```

כמה פעולות יבוצעו על מערך בגודל n?

תלוי איפה (אם בכלל) נמצא האיבר שמחפשים.

- אם key לא נמצא כלל:

שורה	מספר פעמים	כמות פעולות
1	n+1	c_1
2	n	c_2
3	0	c_3
4	1	c_4

$$T(n) = c_1(n+1) + c_2 n + c_4 = (c_1 + c_2)n + (c_1 + c_4)$$

קיבלנו ביטוי מהצורה $an+b$, זאת אומרת פונקציה ליניארית ב- n.

כלומר קצב הגידול של זמן הריצה היא ליניארי ב- n. למשל, כאשר נגדיל את n פי 2, יגדל זמן הריצה בערך פי 2 (הדבר מדויק יותר ככל ש- n גדול יותר).

מסמנים זאת באמצעות האות היוונית תטא: $T(n) = \Theta(n)$.

- אם key נמצא ב- $A[1]$ יבוצעו בסה"כ מספר קבוע של פעולות (לא תלוי ב- n). סימון: $\Theta(1)$.

<u>הסימון Θ – מחלקות סיבוכיות</u>

השימוש ב- Θ נוח מאוד כאשר משווים את ביצועיהם של אלגוריתמים – כך אפשר להשוות <u>קצבי גידול</u>, או <u>סדרי גודל</u> של זמן הריצה שלהם.

כמות פעולות של אלגוריתמים שונים כתלות בגודל הקלט:

	$T_1(n) = 140$	$T_2(n) = 50n+30$	$T_3(n) = n^2$	$T_4(n) = n^2+2n$	$T_5(n) = 2^n$
n=10	140	530	100	120	1024
n=20	140	1030	400	440	1048576
n=100	140	5030	10000	10200	$1.26*10^{30}$ *
complexity	$\Theta(1)$	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(2^n)$

* קישור לערך Big Bang בויקיפדיה: http://en.wikipedia.org/wiki/Big_Bang

מסקנות (לא פורמליות כרגע):

- אפשר להזניח קבועים
- אפשר להתעלם מאיברים בסדר גודל קטן יותר

<u>תלות זמן ריצה בקלט באורך מסוים</u>

כפי שראינו, לפעמים, גם לקלטים שונים בעלי <u>אותו גודל</u> יש זמן ריצה שונה.

כלומר כמות הפעולות תלויה לפעמים גם בקלט עצמו (ולא רק בגודלו).

נסמן ב- $\mathrm{K}(n)$ את קבוצת כל הקלטים בגודל n של האלגוריתם Alg.

לכל קלט $I \in \mathrm{K}(n)$, $T(I)$ יציין את זמן הריצה של האלגוריתם הרץ על הקלט הזה.

- זמן הריצה של Alg <u>במקרה הגרוע</u> (worst-case) הוא: $T_{worst}(n) = \max\{T(I) \,|\, I \in \mathrm{K}(n)\}$

  (זמן הריצה הארוך ביותר של Alg על קלט באורך n)

- זמן הריצה של Alg <u>במקרה הטוב</u> (best-case) הוא: $T_{best}(n) = \min\{T(I) \,|\, I \in \mathrm{K}(n)\}$

  (זמן הריצה הקצר ביותר של Alg על קלט באורך n)

- זמן הריצה של Alg <u>בממוצע</u> (average): $T_{average}(n) = (\sum_{I \in \mathrm{K}(n)} T(I)) / |\mathrm{K}(n)|$

  (זמן הריצה הממוצע של Alg על כל הקלטים באורך n, נקרא גם תוחלת זמן הריצה)

הערה: פה הנחנו שהקלטים מופיעים בסיכויים שווים.

בד"כ נתעניין יותר בזמן הריצה במקרה הגרוע.

ניתוח חיפוש בינארי

• <u>המקרה הגרוע</u>: כאשר key לא נמצא כלל ב- A.

בסוף האיטרציה הראשונה גודל תת-המערך בו מחפשים קטן ל- $n/2$ לכל היותר.

בסוף האיטרציה השניה: $n/4$ לכל היותר.

בסוף האיטרציה ה- k: $n/2^k$ לכל היותר.

כמה איטרציות? נחפש את ה- k בסוף האיטרציה האחרונה:

$$\frac{n}{2^k} < 1$$

$$\Rightarrow n < 2^k$$

$$\Rightarrow \log n < k$$

$$\Rightarrow k = \lfloor \log n + 1 \rfloor$$

(k הוא השלם המינימלי שגדול ממש מ- $\log n$)

בכל איטרציה מתבצע מספר קבוע של פעולות (נסמן c), ולכן: $\boldsymbol{T(n) = c\cdot \lfloor \log n+1 \rfloor = \Theta(\log n)}$.

• <u>המקרה הטוב</u>: אם key נמצא באמצע A, יבוצעו בסה"כ מספר <u>קבוע</u> של פעולות, כלומר $\Theta(1)$.

<u>הערה</u>: זהו אלגוריתם אופטימלי מבחינת סיבוכיות זמן עבור בעיית החיפוש במערך ממוין.

כמה כללים שימושיים לניתוח סיבוכיות

אלגוריתמים בעלי מבנה מהצורה הבאה (קבוע $k > 0$) הם ליניאריים:

$\Theta(n)$

$i \leftarrow 1$
while $i < n$
$\quad i \leftarrow i + 1$

$i \leftarrow 1$
while $i < n$
$\quad i \leftarrow i + k$

$i \leftarrow n$
while $i > 1$
$\quad i \leftarrow i - k$

אלגוריתמים בעלי מבנה מהצורה הבאה (קבוע $k > 1$) הם לוגריתמיים:

$\Theta(\log n)$

$i \leftarrow 1$
while $i < n$
$\quad i \leftarrow i * 2$

$i \leftarrow 1$
while $i < n$
$\quad i \leftarrow i * k$

$i \leftarrow n$
while $i > 1$
$\quad i \leftarrow i / k$

ומה לגבי:

הוכיחו:
$\Theta(\log\log n)$

$i \leftarrow 2$
while $i < n$
$\quad i \leftarrow i * i$

$i \leftarrow 2$
while $i < n$
$\quad i \leftarrow i^k$

$i \leftarrow n$
while $i > 2$
$\quad i \leftarrow \sqrt[k]{i}$

מיון הכנסה

<u>דוגמה: בעיית המיון</u>

<u>הגדרת הבעייה</u>:

<u>קלט</u>: רשימה (למשל מערך) A של איברים $(a_1, a_2, ..., a_n)$ שמוגדר עליהם סדר.

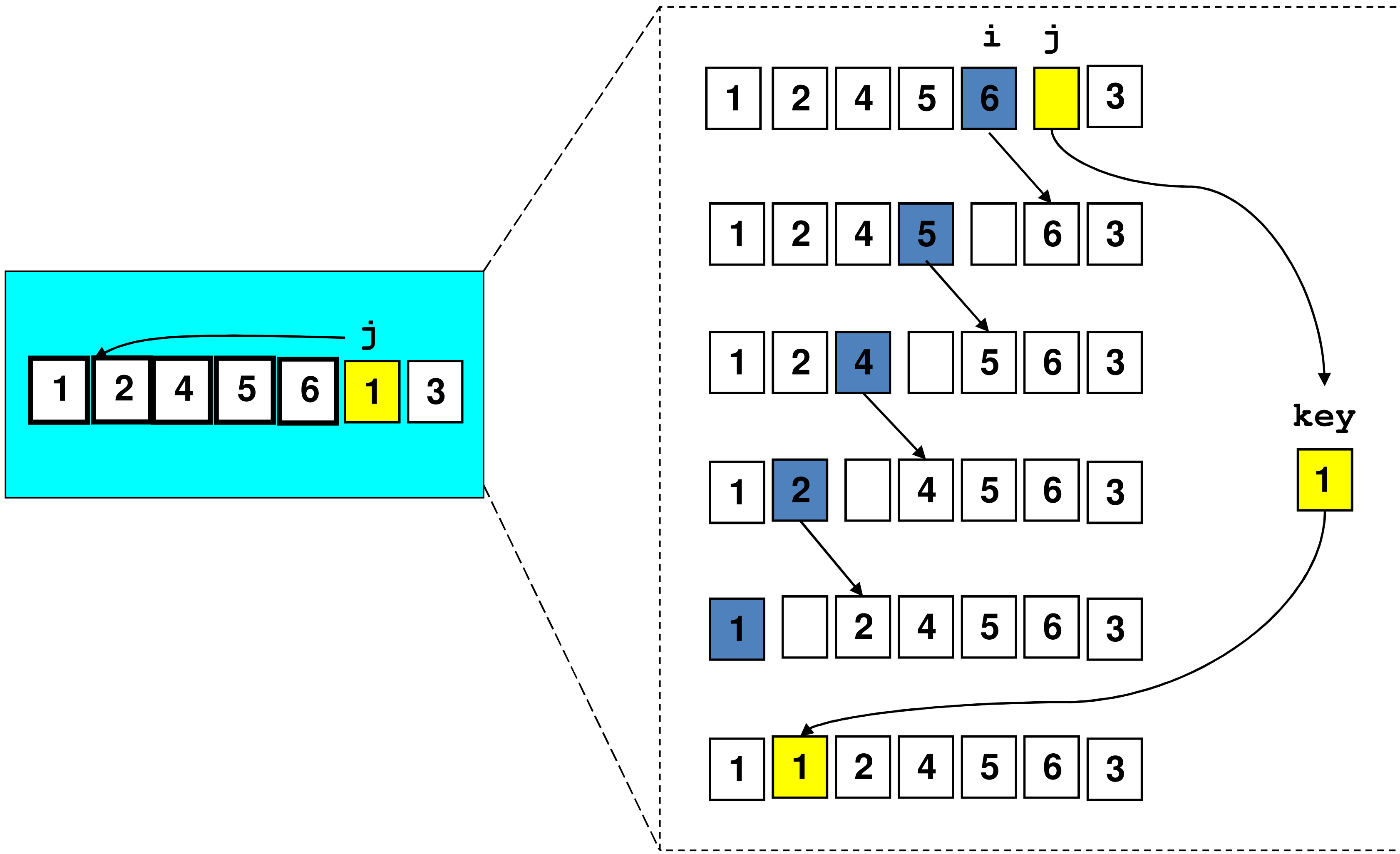
<u>פלט</u>: תמורה (permutation) של A: $(a_1', a_2', ..., a_n')$. כך שהם מסודרים מקטן לגדול:
$a_1' \le a_2' \le ... \le a_n'$

<u>אלגוריתם</u>: מיון הכנסה.

```
Insertion-Sort(A, n)
    for j ← 2 to n
        key ← A[j]
        ► insert A[j] into the sorted sequence A[1..j-1]
        i ← j -1
        while i > 0 and A[i] > key
            A[i+1] ← A[i]
            i ← i – 1
        A[i+1] ← key
```

מיון הכנסה

הדגמת איטרציה חיצונית אחת של מיון הכנסה



ניתוח מיון הכנסה

```
Insertion-Sort(A, n)
    for j ← 2 to n
        key ← A[j]
        ▶ insert A[j] into the sorted sequence A[1..j-1]
        i ← j -1
        while i > 0 and A[i] > key
            A[i+1] ← A[i]
            i ← i – 1
        A[i+1] ← key
```

- <u>המקרה הגרוע</u> הוא כאשר המערך ממוין הפוך (מגדול לקטן).
  במקרה זה מבצעת הלולאה הפנימית j-1 איטרציות, לכל ערך של j בין 2 ל- n.
  הלולאה החיצונית מבצעת n-1 איטרציות.
  נניח שכל איטרציה פנימית דורשת c_1 פעולות, וכל איטרציה חיצונית c_2 פעולות (ללא האיטרציות הפנימיות).

$$T(n) = c_1 \sum_{j=2}^{n} (j-1) + c_2(n-1) = c_1 \frac{n(n-1)}{2} + c_2(n-1) = \Theta(n^2)$$

- <u>המקרה הטוב</u> הוא כאשר המערך ממוין (כרגיל, בסדר עולה).
  הפעם הלולאה הפנימית מבצעת 0 איטרציות כל פעם.

$$T(n) = c_2(n-1) = \Theta(n)$$

חסמים עליונים ותחתונים

לפעמים אנו מעוניינים לתת חסם עליון / תחתון לזמן ריצה של אלגוריתם.

- כי קשה לחשב את סדר הגודל (Θ) של זמן הריצה, אבל קל לחסום אותו מלמעלה / מלמטה
- כי מעניין אותנו רק חסם עליון / תחתון לזמן הריצה (דוגמאות?)

סימון	שם הסימון	משמעות	סוג החסם
$t(n) = O(g(n))$	או גדולה	t לא גדולה אסימפטוטית מ- g	עליון
$t(n) = \Omega(g(n))$	אומגה גדולה	t לא קטנה אסימפטוטית מ- g	תחתון
$t(n) = o(g(n))$	או קטנה	t קטנה אסימפטוטית מ- g	עליון לא הדוק
$t(n) = \omega(g(n))$	אומגה קטנה	t גדולה אסימפטוטית מ- g	תחתון לא הדוק
$t(n) = \Theta(g(n))$	תטא	אותו קצב גידול אסימפטוטי	הדוק

נעיר, שניתן לחשב חסמים עליונים / תחתונים הן למקרה הגרוע, הן למקרה הטוב, והן לזמן הריצה הממוצע.

חסמים עליונים ותחתונים

כאמור, לפעמים קשה לחשב את סדר הגודל של זמן הריצה, ואז ננסה לחשב חסם עליון ותחתון בנפרד. ישנם אלגוריתמים (מסובכים) שהחסם העליון והתחתון שלהם לא מתלכדים, ואז סדר הגודל של זמן הריצה שלהם לא ידוע.

חסמים עליונים

$t(n)=O(n^3)$

$t(n)=O(n^2)$

חסם הדוק???

אלגוריתם שזמן ריצתו $t(n)$

חסמים תחתונים

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

$t(n)=\Omega(n)$

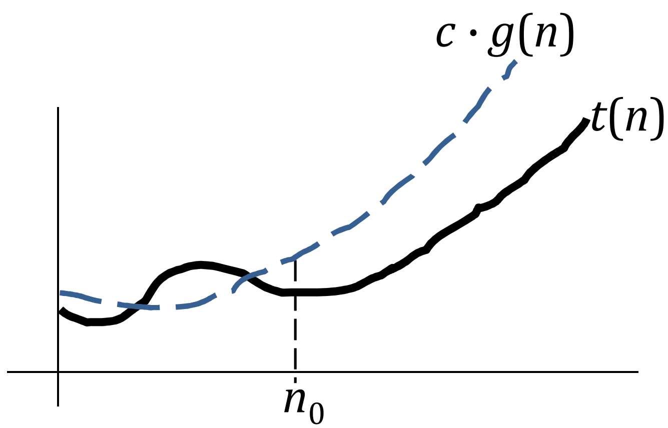
בקורס שלנו ברוב המקרים:

- או שנוכל לחשב חסם הדוק (Θ) באופן ישיר
- או שנוכל לחשב חסם עליון וחסם תחתון שמתלכדים

חסם עליון – O גדולה

תהיינה $t(n)$ ו- $g(n)$ שתי פונקציות חיוביות אסימפטוטית.

$t(n) = O(g(n))$ אם קיימים קבועים חיוביים c, n_0 כך שלכל $n>n_0$: $t(n) \leq c \cdot g(n)$



$t(n) = O(g(n))$

לדוגמה:

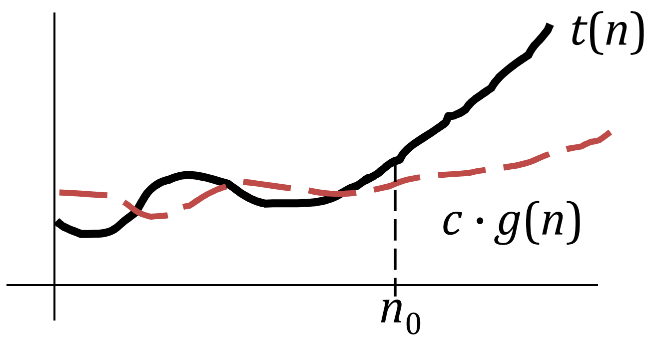
$$t(n) = 10n^2 + 30 = O(n^2)$$

$$t(n) = 10n^2 + 30 = O(n^3)$$

חסם תחתון – Ω גדולה

תהיינה $t(n)$ ו- $g(n)$ שתי פונקציות חיוביות אסימפטוטית.

$t(n) = \Omega(g(n))$ אם קיימים קבועים חיוביים c, n_0 כך שלכל $n>n_0$: $t(n) \geq c \cdot g(n)$



$t(n) = \Omega(g(n))$

לדוגמה:

$$t(n) = 10n^2 + 30 = \Omega(n^2)$$

$$t(n) = 10n^2 + 30 = \Omega(n)$$

"או" קטנה ו"אומגה" קטנה

תהיינה $t(n)$ ו- $g(n)$ שתי פונקציות חיוביות אסימפטוטית.

$t(n) = \mathrm{o}(g(n))$ אם $\lim_{n\to\infty} t(n)/g(n) = 0$

לדוגמה:

$$t(n) = 10n^2 + 30 = \mathrm{o}(n^3)$$

$$t(n) = 10n^2 + 30 \neq \mathrm{o}(n^2)$$

$t(n) = \omega(g(n))$ אם $\lim_{n\to\infty} t(n)/g(n) = \infty$

לדוגמה:

$$t(n) = 10n^2 + 30 = \omega(n)$$

$$t(n) = 10n^2 + 30 \neq \omega(n^2)$$

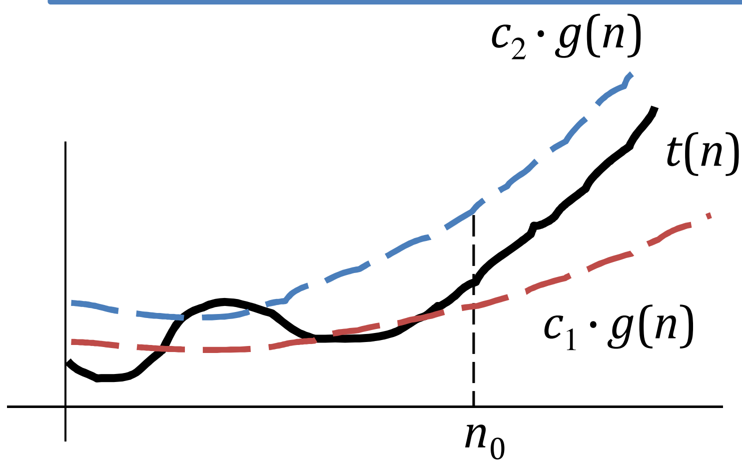
חסם הדוק – Θ

כעת (סוף סוף) נגדיר פורמלית את Θ:

תהיינה $t(n)$ ו- $g(n)$ שתי פונקציות חיוביות אסימפטוטית.

$t(n) = \Theta(g(n))$ אם קיימים קבועים חיוביים c_1, c_2 ו- n_0 כך שמתקיים לכל $n \geq n_0$

$$c_1 g(n) \leq t(n) \leq c_2 g(n)$$



- $t(n) = \Theta(g(n))$ אם"ם $t(n) = O(g(n))$ וגם $t(n) = \Omega(g(n))$
- $t(n) = \Theta(g(n))$ אם"ם $t(n) = O(g(n))$ וגם $t(n) \neq o(g(n))$
- $t(n) = \Theta(g(n))$ אם"ם $t(n) = \Omega(g(n))$ וגם $t(n) \neq \omega(g(n))$

חסמים – כללים שימושיים (1)

הוכיחו: אם $f_1(n) = O(g_1(n))$ וגם $f_2(n) = O(g_2(n))$ אז:

א. $f_1(n) + f_2(n) = O(\max\{g_1(n), g_2(n)\})$

ב. $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$

חסמים – כללים שימושיים (2)

טענה: עבור k קבוע כלשהו, תהיינה $f_1(n), \dots f_k(n)$ פונקציות חיוביות אסימפטוטית, אז:

$$f_1(n) + f_2(n) + \dots + f_k(n) = \Theta(\max\{f_i(n)\})$$

לדוגמה: $74\log n + 2n^3 + 15n = \Theta(n^3)$

הוכחה: עבור n מספיק גדול:

$$\max\{f_i(n)\} \quad \le \quad f_1(n) + f_2(n) + \dots + f_k(n) \quad \le \quad k(\max\{f_i(n)\})$$

שאלה: האם הטענה נכונה גם כאשר מספר המחוברים אינו בהכרח קבוע?

חסמים – כללים שימושיים (3)

- $\log^k n = o(n^\varepsilon)$ לכל $\varepsilon, k>0$ (הוכחה בעזרת כלל לופיטל)

  למשל: $\log^2 n = o(n^{0.001})$

- $n^k = o(a^n)$ עבור $a>1$

- $a^n = o(b^n)$ עבור $0<a<b$

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

- טור גיאומטרי: $\sum_{i=0}^{n} x^i = {x^{n+1}-1}/{x-1} = \Theta(x^n) \quad (x \neq 1)$

- טור גיאומטרי מתכנס: $\sum_{i=0}^{\infty} x^i = 1/(1-x) = \Theta(1) \quad (0 < x < 1)$

- הטור ההרמוני: $H_n = \sum_{i=1}^{n} 1/i = \Theta(\log n)$

- פולינום מדרגה d: $\sum_{i=0}^{d} a_i n^i = \Theta(n^d) \quad (a_d \neq 0)$

היררכיה של סיבוכיות

קבוע $\Theta(1)$

$\Theta(\log\log n)$

לוגריתמי $\Theta(\log n)$

פולי-לוגריתמי $\Theta(\log^2 n)$

ליניארי $\Theta(n)$

$\Theta(n\log n)$

ריבועי $\Theta(n^2)$

$\Theta(n^2\log n)$

אקספוננציאלי $\Theta(2^n)$

(מעריכי) $\Theta(3^n)$

היררכיה של סיבוכיות

טענה חשובה: $\log(n!) = \Theta(n\log n)$

- $\log(n!) \le \log(n^n) = n\log n$ $\Rightarrow$ $\log(n!) = O(n\log n)$

- $$\log(n!) = \log(n\cdot(n-1)\cdot\ldots\cdot\lceil \tfrac{n}{2}\rceil \cdot \ldots 2\cdot 1) \ge$$
$$\log(n\cdot(n-1)\cdot\ldots\cdot\lceil \tfrac{n}{2}\rceil) \ge \log(\lceil \tfrac{n}{2}\rceil \cdot \ldots \cdot \lceil \tfrac{n}{2}\rceil) =$$
$$\log(\lceil \tfrac{n}{2}\rceil^{\lfloor \frac{n}{2}\rfloor+1}) \ge \log(\tfrac{n}{2}^{\frac{n}{2}}) = \tfrac{n}{2}\log(\tfrac{n}{2}) =$$
$$\tfrac{n}{2}\log n - \tfrac{n}{2}\log 2 \ge \tfrac{n}{2}\log n - \tfrac{n}{4}\log n = \tfrac{1}{4}n\log n$$
$\Rightarrow$ $\log(n!) = \Omega(n\log n)$

קביעת סיבוכיות זמן של אלגוריתמים

```
Alg(A, n)
    s ← 0
    for i ← 1 to n
        j ← 1
        while j ≤ n
            j ← j + i
            s ← s + A[i]
```

ננתח את סיבוכיות הזמן של האלגוריתם הבא כפונ' של n:

איזה מקרה ננתח? גרוע? טוב?
הפעם אין הבדל!
בכל מקרה מתבצעת אותה כמות של פעולות.

ישנן 2 לולאות מקוננות (אחת בתוך השנייה).

__ניתוח גס__: לולאה חיצונית שמבצעת n איטרציות
לולאה פנימית שמבצעת __לכל היותר__ n איטרציות כל פעם:

$$T(n) \le c_1 + n\,(c_2 + n \cdot c_3) = O(n^2)$$

__ניתוח זהיר יותר__: מספר האיטרציות הפנימיות תלוי בערכו של i והוא שווה ל- $\lceil n/i \rceil$.

$$T(n) = \Theta\left(\sum_{i=1}^{n} \lceil n/i \rceil\right) = \Theta\left(n \cdot \left(1 + \tfrac{1}{2} + \tfrac{1}{3} + \ldots + \tfrac{1}{n}\right)\right) = \Theta(n \cdot H_n) = \Theta(n \log n)$$

קביעת סיבוכיות זמן של אלגוריתמים

ננתח את סיבוכיות הזמן של האלגוריתם הבא, המקבל מערך ממוין A, גודלו n, ומפתח לחיפוש key:

```
Alg-Search(A, n, key)
1.    for i ← 1 to n
2.        found ← Binary-Search(A, 1, i, key)
3.        if found ≠ Nil
4.            print('i' is located in index 'found')
```

<u>המקרה הגרוע – אף חיפוש לא מצליח</u>

<u>*ניתוח גס*</u>: n קריאות לחיפוש בינארי על תת מערך בגודל≤n. לכן כל חיפוש מתבצע ב- $O(\log n)$:

$T(n) = n \cdot O(\log n) = O(n\log n)$

<u>*ניתוח זהיר יותר*</u>: סיבוכיות הזמן של חיפוש על תת-מערך באורך i היא $\Theta(\log i)$:

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

<u>המקרה הטוב – כל החיפושים מצליחים מייד</u>

n קריאות לחיפוש בינארי, שמבצע $\Theta(1)$ פעולות. סה"כ $\Theta(n)$.

איך נראה הקלט שנותן את המקרה הטוב?

שאלות חזרה

1. האם אתם מבינים מה ההבדל בין זמן ריצה במקרה גרוע/טוב/בממוצע לבין חסם עליון/תחתון/הדוק?

2. מהי סיבוכיות הזמן כתלות ב- n של שני הקטעים הבאים? שימו לב ששני הקטעים מבצעים בדיוק אותו דבר.

```
1.    n ← 100
2.    for  i ← 1 to n
3.         …
```

```
1.    n ← 100
2.    for  i ← 1 to 100
3.         …
```

3. מה המשמעות של השוויון n=o(m)? תנו דוגמה ל- n ול- m המקיימים אותו.

4. הסתכלו שוב בשקף שכותרתו "<u>הסימון Θ – מחלקות סיבוכיות</u>". נניח שבידנו מחשב המבצע מיליארד פעולות בסיסיות בשנייה. כמה זמן יידרש להרצת האלגוריתמים שזמני הריצה שלהם מתוארים ע"י ה- T_i השונים עבור n=10, 20, 100?

תשובות לשאלות חזרה

1. למשל, O הוא חסם עליון, אותו אפשר לחשב הן עבור המקרה הגרוע, הן עבור המקרה הטוב, והן עבור זמן הריצה הממוצע.

2. סיבוכיות הזמן של קטע הקוד השמאלי היא $\Theta(n)$ ואילו של הימני היא $\Theta(1)$. למעשה, השאלה שנשאלת כאן היא מהו קצב הגידול של זמן הריצה כתלות ב- n. כלומר, אם למשל מגדילים את n פי 2, פי כמה יגדל זמן הריצה? בקטע הימני זמן הריצה ליניארי ב- n ויגדל בערך פי 2, ואילו בקטע השמאלי זמן הריצה כלל לא ישתנה, כלומר הוא אינו תלוי כלל ב- n.

3. n קטן אסימפטוטית בסדר גודל מ- m. לדוגמה, $n=\log m$ או $n=m^{0.99}$ אבל לא $n=½m$ או $n=m-10$.

4. יש לחלק כל מספר ב- 10^9. תשובה עבור $n=100$:

	$T_1(n)=140$	$T_2(n) = 50n+30$	$T_3(n) = n^2$	$T_4(n) = n^2+2n$	$T_5(n) = 2^n$
$n=100$	$1.4*10^{-7}$ sec	$\sim5*10^{-6}$ sec	10^{-4} sec	$\sim10^{-4}$ sec	$1.26*10^{21}$ כ- 40 טריליון שנים. המפץ הגדול ארע לפני כ- 15 מיליארד שנים.

<u>טורים וכללים שימושיים</u>

טור אריתמטי: $\sum_{i=1}^{n} i = \frac{1}{2} n(n+1) = \Theta(n^2)$

טור גאומטרי: $(x \neq 1) \quad \sum_{i=0}^{n} x^i = {x^{n+1}-1}/{x-1} = \Theta(x^n)$

טור גאומטרי אינסופי יורד: $(0 < x < 1) \quad \sum_{i=0}^{\infty} x^i = 1/(1-x) = \Theta(1)$

טור ריבועים: $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

טור מעוקבים: $\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}$

ליניאריות של סכום: $\sum_{i=1}^{n} (ca_i + b_i) = c\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i$

ליניאריות הסימונים האסימפטוטיים: $\sum_{i=1}^{n} \Theta(f(i)) = \Theta\left(\sum_{i=1}^{n} f(i)\right)$

חוקי חזקות ולוגריתמים שימושיים

$$a^{-k} = 1/a^{k}$$

$$(a^{m})^{n} = a^{mn}$$

$$a^{m}a^{n} = a^{m+n}$$

$$\log_c(ab) = \log_c a + \log_c b$$

$$\log_c a^{n} = n\log_c a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b(1/a) = -\log_b a$$

$$a^{\log_b c} = c^{\log_b a}$$

$$a = b^{\log_b a}$$

תרגילים

תרגילים נוספים

1. נתון מערך A בגודל n. ידוע ש- $\lceil\sqrt{n}\rceil$ - n האיברים הראשונים שלו ממוינים.
הציעו אלגוריתם שממיין את A בסיבוכיות זמן ליניארית.

2. נתון מערך A בגודל n של מספרים ממשיים, ומספר ממשי נוסף z.
א. הציעו אלגוריתם שמכריע <u>האם</u> קיימים שני אינדקסים שונים i ו- j כך ש- $A[i]+A[j] = z$.
ב. מה צריך לשנות באלגוריתם כדי שיחזיר גם את i ו- j הנ"ל, אם הם קיימים?

3. א. נתחו את סיבוכיות הזמן של Alg1 כתלות ב- n.

```
Alg1 (n)
    k ← n
    while k > 0
        k ← ⌊k/5⌋
        for j ← 1 to ⌊log n⌋
            for i ← 1 to ⌊n/2⌋
                print(i)
```

ב. נתחו את סיבוכיות הזמן של Alg2 כתלות ב- n ו- m.
הניחו כי חישוב ביטוי מהצורה a^b דורש זמן $\Theta(b)$.

```
Alg2 (n, m)
    while n > 0
        a ← m^n
        i ← 1
        while i ≤ a
            i ← i *2
        n ← n -1
```

תרגילים נוספים

4. הוכיחו או הפריכו כל אחת מהטענות הבאות באופן פורמלי, תוך שימוש בהגדרות לחסמים אסימפטוטיים:

א. $\log(n\text{-}\log n) = \Theta(\log n)$ ב. $2^{2\log n} = \Theta(n\log n)$

ג. $2^{2n+1} + 3^n = O(2^{2n})$ ד. $\binom{n}{3} = \Theta(n^3)$

5. סדרו את הפונקציות הבאות לפי סדר גודל אסימפטוטי:

$$t_1(n) = 2^{\sqrt{n}} \qquad t_2(n) = \log(n^n \cdot n!) \qquad t_3(n) = (\log\log n)^{\log n}$$

6. שני סעיפים מתוך בעיה 3-4 מספר הלימוד.

יהיו $f(n)$ ו- $g(n)$ שתי פונקציות חיוביות אסימפטוטית. הוכיחו או הפריכו:

ג. אם $f(n)=O(g(n))$ אז $\log(f(n))=O(\log(g(n)))$. נתון גם כי שתי הפונקציות אינן חסומות.

ד. אם $f(n)=O(g(n))$ אז $2^{f(n)} = O(2^{g(n)})$

פתרון 1

תחילה נמיין את $\lceil\sqrt{n}\rceil$ האיברים האחרונים של A, בעזרת מיון בועות למשל.

לאחר מכן נמזג את תת-המערך $A[1..n-\lceil\sqrt{n}\rceil]$ עם תת-המערך $A[n-\lceil\sqrt{n}\rceil+1 .. n]$.

סיבוכיות זמן:

- שלב המיון ידרוש $\Theta((\sqrt{n})^2) = \Theta(n)$.
- שלב המיזוג ידרוש $\Theta((n-\lceil\sqrt{n}\rceil) + \lceil\sqrt{n}\rceil) = \Theta(n)$.

סה"כ $\Theta(n)$.

תזכורת: סיבוכיות הזמן של מיזוג שני מערכים ממוינים בגדלים n ו- m היא $\Theta(n+m)$.

פתרון 2

פתרון ישיר: נעבור על כל הזוגות האפשריים (לולאה בתוך לולאה) ונבדוק את סכומם.

יש $\binom{n}{2}$ זוגות אפשריים, ולכן סיבוכיות הזמן היא $\Theta(n^2)$.

פתרון יעיל:

בשלב הראשון נמיין את A בעזרת מיון מיזוג.

לאחר מכן נעבור על איברי A, ולכל איבר $A[i]$ נחפש ע"י חיפוש בינארי ב- A את z-$A[i]$. אם החיפוש הצליח מתישהו, נחזיר true, אחרת false.

סיבוכיות זמן: שלב המיון ידרוש $\Theta(n\log n)$. בנוסף n פעמים חיפוש בינארי - $\Theta(n\log n)$.

סה"כ $\Theta(n\log n)$.

<u>המשך פתרון 2</u>

<u>פתרון יעיל יותר (מתואר בקוד-דמה):</u>

```
Find-Sum (A, n, z)
1.   Merge-Sort(A, n)
2.   l←1,  r←n
3.   while  l < r
4.        if A[l]+A[r] = z
5.             return true
6.        else if A[l]+A[r] < z
7.             l ← l+1
8.        else
9.             r ← r-1
10.  return false
```

סיבוכיות זמן:

שלב המיון ידרוש $\Theta(n\log n)$. שורות 2-10 לוקחות $\Theta(n)$.

סה"כ $\Theta(n\log n)$.

<u>הערות:</u>

- הפתרון השני יעיל יותר מבחינת זמן ריצה, אבל לא בסדר גודל, אלא בקבועים (שלב המיון הוא "צוואר-בקבוק").
- שימו לב שהצגנו פתרון אחד במילים, ואילו את פתרון השני ב- pseudo-code.
- הוכחת הנכונות של הפתרון הראשון היא טריוויאלית, ונובעת מנכונותם של מיון מיזוג וחיפש בינארי. בשביל להוכיח את נכונות הפתרון השני יש לנסח שמורת לולאה מתאימה.

<u>המשך פתרון 2</u>

ב. בפתרון הראשון, אם מצאנו i ו- j מתאימים פשוט נחזיר אותם.

נשים לב שבפתרונות היעילים יותר אנו ממיינים את המערך, ולכן אם מצאנו i ו- j מתאימים, אלו לא בהכרח האינדקסים של איברים אלו <u>במערך המקורי</u>.
לפיכך, נשתמש במערך עזר בגודל n, שישמור לכל איבר את האינדקס המקורי שלו. בזמן המיון, כאשר אנו משנים את מיקומם של איברים, נשנה יחד איתם גם את מיקומם של האינדקסים הללו, וכך בסוף המיון נחזיר את האינדקסים המקוריים של האיברים (אם מצאנו).

פתרון 3

א. $\Theta(n\log^2 n)$

ב. לכל k בין 1 ל- n, מבצעים:

- חישוב m^k פעם אחת, בזמן $\Theta(k)$
- $\log(m^k)$ איטרציות של הלולאה הפנימית, שבכל אחת מספר קבוע של פעולות.

בסה"כ:

$$\sum_{k=1}^{n}\left(\Theta(k)+\Theta(\log(m^k))\right)=\sum_{k=1}^{n}\left(\Theta(k)+\Theta(k\log m)\right)=\sum_{k=1}^{n}\Theta(k\log m)=\Theta(n^2\log m)$$

פתרון 4 א'

$$\log(n\text{-}\log n) \le \log n \qquad (c=1,\ n_0=1)$$

$$\underbrace{\log(n\text{-}\log n) \ge}_{n\ge 4} \log(n\text{-}½n) = \log(½n) = \underbrace{\log n\text{-}1 \ge}_{n\ge 4} \log n - ½\log n = ½\log n \quad (c=½,\ n_0=4)$$