תרגיל כיתה I
מבוא לסיבוכיות

1 חסמי סיבוכיות — O, Ω, Θ.

נגדיר באופן פורמלי את חסמי הסיבוכיות ונשתמש בהם ע"מ למדוד זמן ריצה או כמות זכרון של תכנית.

1.1 הגדרות

עבור $f(n)$ ו-$g(n)$ פונקציות חיוביות.

1. $f(n)$ היא $\mathrm{O}(g(n))$ ("או גדול של $g(n)$") אם קיימים קבועים חיוביים c ו-n_0 כך שלכל $n > n_0$ מתקיים:

$$f(n) \le c \cdot g(n)$$

דוגמאות:

(א) $2n = \mathrm{O}(n)$ (נבחר $c = 3$ ו-$n_0 = 1$ למשל)

(ב) $4n^2 = \mathrm{O}(n^3)$ (נבחר $c = 1$ ו-$n_0 = 5$ כי $4n^2 < n \cdot n^2 = n^3$ עבור $n \ge 5$)

2. $f(n)$ היא $\Omega(g(n))$ ("אומגה של $g(n)$") אם קיימים קבועים חיוביים c ו-n_0 כך שלכל $n > n_0$ מתקיים:

$$f(n) \ge c \cdot g(n)$$

דוגמאות:

(א) $n^2 = \Omega(n)$ (נבחר $c = n_0 = 1$)

(ב) $n = \Omega(7\log n)$ (נבחר $c = \frac{1}{7}$ ו-$n_0 = 1$)

3. $f(n)$ היא $\Theta(g(n))$ ("תטה של $g(n)$") אם $f(n) = \mathrm{O}(g(n))$ וגם $f(n) = \Omega(g(n))$.

דוגמא: למשל ראינו כי $2n = \mathrm{O}(n)$, מתקיים גם $2n = \Omega(n)$.

1.2 שימוש בסימונים

בד"כ נשתמש בסימונים $\mathrm{O}, \Omega, \Theta$ כדי לחסום את זמן הריצה של אלגוריתם מסויים. נתייחס לזמן הריצה "האמיתי" של האלגוריתם כאל הפונקציה f בהגדרה, ואל החסם שאנחנו מחפשים כאל הפונקציה g באותה ההגדרה. נאמר כי זמן הריצה של האלגוריתם הוא $\mathrm{O}(n^2)$ (למשל) אם לכל n גדול מספיק, זמן הריצה קטן מ-$c \cdot n^2$ עבור קבוע כלשהו c.

O הוא חסם עליון: אם זמן ריצת התכנית הוא $\mathrm{O}(n^2)$, אז עבור n-ים מספיק גדולים התכנית רצה לא יותר מאשר קבוע כפול n^2. בד"כ נרצה למצוא חסם עליון קטן ככל שנוכל.

Ω הוא חסם תחתון: אם זמן ריצת התכנית הוא $\Omega(n^2)$, אז עבור n-ים מספיק גדולים התכנית רצה לפחות $c \cdot n^2$ זמן עבור c - קבוע כלשהו. בד"כ נרצה למצוא חסם תחתון גבוה ככל שנוכל.

1.3 תרגילים

1. יהי $P(n)$ פולינום מדרגה k שמקדמיו חיוביים. כלומר:

$$P(n)=\sum_{i=0}^{k}a_in^i \qquad a_i\geq 0,\quad a_k>0$$

נוכיח כי מתקיים $P(n)=\mathrm{O}(n^k)$.

<u>הוכחה:</u>

$$P(n)=\sum_{i=0}^{k}a_in^i\leq\sum_{i=0}^{k}a_in^k=n^k\cdot\sum_{i=0}^{k}a_i$$

$\left(\sum_{i=0}^{k}a_i\right)$ הינו קבוע, לכן נבחר:

$$c=\sum_{i=0}^{k}a_i,\ \ n_0=1$$

ואז עפ"י הגדרה $P(n)=\mathrm{O}(n^k)$.

2. נוכיח כי עבור $P(n)$ כנ"ל מתקיים גם $P(n)=\Omega(n^k)$.

<u>הוכחה:</u>

$$P(n)=\sum_{i=0}^{k}a_in^i=a_kn^k+\sum_{i=0}^{k-1}a_in^i\geq a_kn^k$$

לכן נבחר:

$$c=a_k,\ \ n_0=1$$

ואז עפ"י הגדרה $P(n)=\Omega(n^k)$.

<u>מסקנה:</u> $P(n)=\Theta(n^k)$.

3. נוכיח כי עבור מספר קבוע $k\geq 2$ מתקיים:

$$\sum_{i=0}^{n}k^i=\mathrm{O}(k^n)$$

<u>הוכחה:</u> נפתח טור הנדסי

$$\sum_{i=0}^{n}k^i=\frac{k^{n+1}-1}{k-1}=\frac{k}{k-1}k^n-\frac{1}{k-1}\leq 2\cdot k^n$$

לכן עפ"י ההגדרה $\sum_{i=0}^{n}k^i=\mathrm{O}(k^n)$.

2 פעולות בסדרי גודל

מה קורה כשיש בתכנית כמה לולאות?
נתבונן במקרה של 2 לולאות: אחת לוקחת $f_1(n)$ זמן והשניה $f_2(n)$ זמן.
נניח ש-$f_1(n) = \mathrm{O}(g_1(n))$ ו-$f_2(n) = \mathrm{O}(g_2(n))$.

1. אם הלולאות מופיעות אחת אחר שניה ("מחוברות בטור"), אזי הזמן הכולל הינו סכום הזמנים: $f(n) = f_1(n) + f_2(n)$.

<u>טענה:</u> $f_1(n) + f_2(n) = \mathrm{O}(g_1(n) + g_2(n)) = \mathrm{O}(\max\{g_1(n),\ g_2(n)\})$

<u>הוכחה:</u> לפי הנתון מתקיים

$$\begin{array}{lll} \exists c_1,\ n_1 & \forall n > n_1 & f_1(n) \le c_1 \cdot g_1(n) \\ \exists c_2,\ n_2 & \forall n > n_2 & f_2(n) \le c_2 \cdot g_2(n) \end{array}$$

לכן עבור $n > \max\{n_1,\ n_2\}$ מתקיים:

$$f_1(n) + f_2(n) \le c_1 \cdot g_1(n) + c_2 \cdot g_2(n) \le \max\{c_1,\ c_2\} \cdot (g_1(n) + g_2(n))$$

אם נבחר $c = \max\{c_1,\ c_2\},\ n_0 = \max\{n_1,\ n_2\}$, נקבל $f_1(n) + f_2(n) = \mathrm{O}(g_1(n) + g_2(n))$.
בנוסף $g_1(n) + g_2(n) \le 2 \cdot \max\{g_1(n),\ g_2(n)\}$
מכאן $f_1(n) + f_2(n) = \mathrm{O}(g_1(n) + g_2(n)) = \mathrm{O}(\max\{g_1(n),\ g_2(n)\})$

2. אם הלולאות מקוננות, אזי הזמן הכולל הינו מכפלת הזמנים: $f(n) = f_1(n) \cdot f_2(n)$.

<u>טענה:</u> $f_1(n) \cdot f_2(n) = \mathrm{O}(g_1(n) \cdot g_2(n))$

<u>הוכחה:</u> עבור $n > \max\{n_1,\ n_2\}$ מתקיים:

$$f_1(n) \cdot f_2(n) \le (c_1 \cdot g_1(n)) \cdot (c_2 \cdot g_2(n)) = (c_1 \cdot c_2) \cdot (g_1(n) \cdot g_2(n))$$

לכן אם נבחר $c = c_1 \cdot c_2,\ n_0 = \max\{n_1,\ n_2\}$, נקבל $f_1(n) \cdot f_2(n) = \mathrm{O}(g_1(n) \cdot g_2(n))$.

3. האם תמיד מתקיים $f_1(n)^{f_2(n)} = \mathrm{O}(g_1(n)^{g_2(n)})$?

<u>תשובה:</u> לא.

<u>דוגמא נגדית:</u> נקח $g_1(n) = g_2(n) = n$, $f_1(n) = f_2(n) = 2n$ (ראינו כי $2n = \mathrm{O}(n)$).
אזי:

$$f_1(n)^{f_2(n)} = (2n)^{2n} = 2^{2n} \cdot n^{2n} = 2^{2n} \cdot n^n \cdot n^n$$

מכיון שלכל קבוע c, לכל $n > c$ מתקיים $2^{2n} \cdot n^n > c$,
לא קיים קבוע המקיים $2^{2n} \cdot n^n \cdot n^n \le c \cdot n^n$.
מכאן $(2n)^{2n} \ne \mathrm{O}(n^n)$.

3 חישוב זמן ריצה של תכניות

נסמן ב-$T(n)$ זמן ריצה של תכנית מסויימת עבור קלט n.
בכל קטעי הקוד להלן נתבונן בזמן ריצת לולאות, כי שאר הפעולות לוקחות זמן קבוע $O(1)$.

1.
```
scanf ("%d", &n);
k = 2;
while (k < n) {
   k += 1;
}
```

__פתרון:__ נסמן ב-$k(i)$ את ערך המשתנה k באיטרציה ה-i-ית.

$k(0) = 2$
$k(i) = k(i-1) + 1 = k(i-2) + 2 = \ldots = k(0) + i = 2 + i$

הלולאה מתבצעת כל עוד מתקיים: $k(i) = 2 + i < n$
מכאן: $i \le n - 2 \quad \Longleftarrow \quad \boxed{T(n) = O(n)}$

2.
```
scanf ("%d", &n);
k = 2;
while (k < n) {
   k = k * 2;
}
```

__פתרון:__

$k(0) = 2$
$k(i) = k(i-1) \cdot 2 = (k(i-2) \cdot 2) \cdot 2 = k(i-2) \cdot 2^2 = \ldots = k(0) \cdot 2^i = 2^{i+1}$
$2^{i+1} \le n \quad \Longrightarrow \quad \log_2 2^{i+1} = i + 1 \le \log_2 n \quad \Longrightarrow \quad \boxed{T(n) = O(\log n)}$

3.
```
scanf ("%d", &n);
k = 2;
while (k < n) {
   k = k * k * k;
}
```

__פתרון:__

$k(0) = 2$
$k(i) = (k(i-1))^3 = ((k(i-2))^3)^3 = (k(i-2))^{3^2} = \ldots = k(0)^{3^i} = 2^{3^i}$
$2^{3^i} \le n \quad \Longrightarrow \quad 3^i \le \log_2 n \Longrightarrow \quad i \le \log_3 \log_2 n \quad \Longrightarrow \quad \boxed{T(n) = O(\log \log n)}$

4.
```
scanf ("%d", &n);
k = n;
while (k > 1) {
   k = floor (k / 5);
   for (j = 1; j <= log (n); j++)
      for (i = 1; i <= floor (n / 2); i++)
         x += 1;
}
```

פתרון: לולאת ה-for הפנימית מתבצעת $\frac{n}{2}$ פעמים (ע"ס הדוגמאות הקודמות), לולאת ה-for החיצונית עושה $\log_2 n$ איטרציות, ולולאת ה-while מתבצעת ב-$\log_5 n$ איטרציות. לכן סה"כ:

$$\boxed{T(n) = \mathrm{O}(\frac{n}{2} \cdot \log_2 n \cdot \log_5 n) = \mathrm{O}(n \cdot \log^2 n)}$$

5.
```
scanf ("%d", &n);
k = 2;
for (i = 1; i <= n; i++) {
   for (j = 1; j <= i; j++)
      x += 1;
   while (k < i)
      k = k * k;
}
```

פתרון: לולאת ה-for הפנימית מתבצעת i פעמים לכל ערך של i, לולאת ה-while עושה איטרציה אחת בלבד (!) לחלק מערכי i האפשריים ועבור שאר הערכים שלו אינה מתבצעת כלל, לכן סיבוכיותה היא $\mathrm{O}(1)$. מכאן שתי הלולאות הפנימיות ביחד לוקחות $\mathrm{O}(i)+\mathrm{O}(1) = \mathrm{O}(i)$ זמן. הלולאה החיצונית מתבצעת ב-n איטרציות. לכן סה"כ:

$$\boxed{T(n) = \sum_{i=1}^{n} \mathrm{O}(i) = \mathrm{O}\left(\sum_{i=1}^{n} i\right) = \mathrm{O}\left(\frac{n(n+1)}{2}\right) = \mathrm{O}\left(n^2\right)}$$

6.
```
int array[n];

void bubble (int *array, n) {
   int i, j, temp;

   for (i = 0; i < n; i++)
      for (j = n – 1; j > i; j––)
         if (array[j–1] > array[j]) {
            temp = array[j–1];
            array[j–1] = array[j];
            array[j] = temp;
         }
}
```

פתרון: $\boxed{T(n) = \mathrm{O}\left(n^2\right)}$ (נסו להגיע לבד).

---

4 היררכיה של מחלקות סיבוכיות

אלגוריתמים "יעילים" יחסית

- קבוע: $O(1)$
- לוגריתמי: $O(\log n)$, $O(\log^2 n)$, $\vdots$, $O(\log^k n)$
- $\vdots$
- פולינומי: $O(n)$, $O(n^2)$, $\vdots$, $O(n^k)$

$\vdots$

- תת-אקספוננציאלי: $O(2^{\log^2 n})$, $\vdots$, $O(2^{\log^k n})$

$\vdots$

- אקספוננציאלי: $O(2^n)$, $O(3^n)$, $\vdots$, $O(n^n) = O(2^{n \log n})$

$\vdots$

- אקספוננציאלי כפול: $O(2^{2^n})$, $\vdots$