Lecture 1 of Geiger & Itai's slide brochure
www.cs.technion.ac.il/~dang/courseDS

מבני נתונים

<u>מטרת הקורס</u>:

1. הכרות עם מבני נתונים ומימושיהם היעילים
2. פיתוח כלים לניתוח יעילות
3. בחירת מבני נתונים לפתרון בעיות

<u>מבני נתונים</u>: מחסנית, תור, מילון, תור עדיפויות, טבלת ערבול...

<u>שימושים</u>: מיון, מימוש שפות תכנות, ארגון קבצים, ועוד ועוד ועוד.

ספר הלימוד העיקרי (קיים גם תרגום):
Cormen, Leiserson, Rivest, "Introduction to Algorithms" (MIT Press).

תוכנית הקורס

1. מבני נתונים בסיסיים וסימונים אסימפטוטיים
2. מערכים ורשימות מקושרות
3. עצים ועצי חיפוש
4. עצי AVL
5. עצי 2-3
עצי דרגות
6. רשימות דילוגים
סיבוכיות משוערכת
7. טבלאות ערבול
8. אחזקת קבוצות זרות
9. מיון
10. מיון
11. טיפול במחרוזות
12. גרפים
13. איסוף אשפה
14. הרצאת חזרה

מבני נתונים בסיסיים וסימונים אסימפטוטיים

חומר קריאה לשיעור זה

Chapter 2 – Growth of functions (23 - 41)
Chapter 4 – Recurrences (53 – 60)
Chapter 11.1 – Stacks and Queues (200 – 204)

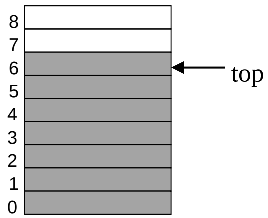
הגדרות בסיסיות

<u>מבנה נתונים</u> הוא אוסף של פעולות על קבוצת נתונים.
<u>מימוש של מבנה נתונים</u> הוא אוסף פרוצדורות, אחת לכל פעולה, המממשות את הפעולות של מבנה הנתונים.

<u>מהי מחסנית?</u>

האם היא מערך עם מציין ל- top ?

אתחול:	top = 0
הוספת איבר:	top = top + 1; A[top] = x;
ראש המחסנית:	A[top]
הוצאת איבר:	top = top – 1



לא! זהו מימוש של מחסנית באמצעות מערך ומצביע.

מחסנית כמבנה נתונים

Last In -- First Out : LIFO   אחרון נכנס – ראשון יוצא

<u>מחסנית מוגדרת ע"י הפעולות הבאות:</u>

create(S)	מחזיר מחסנית S ריקה חדשה.
push(S,x)	מכניס איבר בעל ערך x למחסנית S.
top(S)	מחזיר את האיבר שבראש המחסנית S (המחסנית אינה משתנה).
pop(S)	מוציא את האיבר שבראש המחסנית S.
is_empty(S)	מחזיר true אם המחסנית S ריקה ו-false אחרת.

מחסנית כמבנה נתונים (המשך)

<u>פעולות המחסנית מקיימות את הכללים הבאים:</u>

1. אפשר לבצע pop, top רק על מחסנית לא ריקה.
2. מיד לאחר create(S), is_empty(S) מחזיר ערך true.
3. לאחר ביצוע push, ואחריו pop, המחסנית לא משתנה.

כל מימוש חייב לאפשר את הפעולות ולקיים את הכללים.
יש טענות הנובעות רק מהכללים, בלי תלות במימוש.

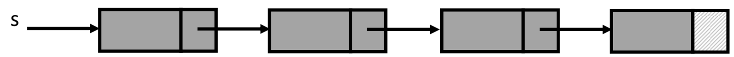
<u>לדוגמא:</u>

```
create(S);
push(S,17);
pop(S);
print is_empty(S);
```

מה יודפס ?

מימושי מחסנית

1. מימוש בעזרת מערך (כפי שהוסבר).
2. מימוש בעזרת רשימה מקושרת:



<u>הגדרת צומת:</u>

```
typedef  struct  node {
        DATA_TYPE info;
        struct node *next;
}  NODE;

typedef   NODE*   STACK;
```

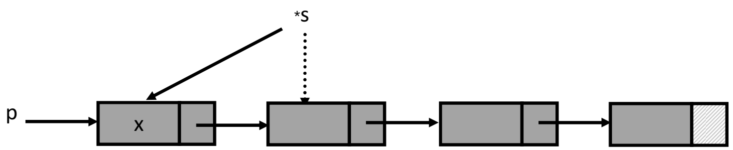
<u>יצירת מחסנית ריקה:</u>
<u>:Create(S)</u>

```
void create (STACK  *S) {
      *S = NULL ;
}
```


הכנסת איבר

<u>פעולת push(S,x):</u>

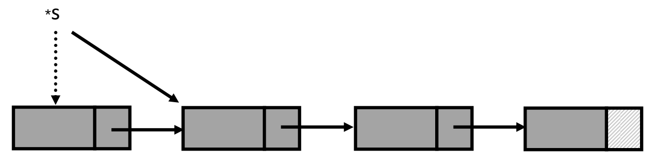
```
void  push (STACK  *S,  DATA_TYPE x){
      NODE *P;
      P = malloc (sizeof (NODE));
      P → info = x;
      P → next = *S ;
      *S = P ;
}
```



אחזור והוצאת איבר

פעולת pop(S):

```
void  pop ( STACK  *S){
      STACK  t ;
      t = (*S) → next;
      free (*S) ;
      *S = t ;
}
```



פעולת top(S):

```
DATA_TYPE  top ( STACK  *S){
      return (*S) → info ;
}
```


תור כמבנה נתונים

First In -- First Out : FIFO   ראשון נכנס – ראשון יוצא

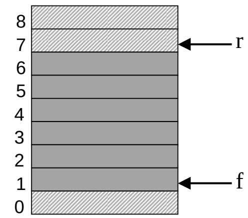
תור מוגדר ע"י הפעולות הבאות:

create(Q)	מחזיר תור ריק.
head(Q)	מחזיר את ערך האיבר שבראש התור Q (התור אינו משתנה).
enqueue(Q,x)	מכניס איבר עם ערך x לסוף התור Q.
dequeue(Q)	מוציא את האיבר שבראש התור Q.
is_empty(Q)	מחזיר true אם התור Q ריק ו-false אחרת.

מימוש של תור בעזרת מערך

מערך Q בן n איברים עם שני מציינים

r = מציין את מקום האיבר שאחרי סוף התור
f = מציין את מקום האיבר שבראש התור

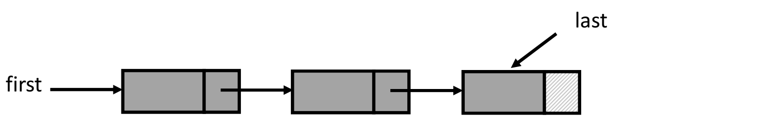



```
head(Q):        Q[f];
enqueue(Q,x):   Q[r] = x;
                r = (r+1) %n;
dequeue(Q):     f = (f+1) % n;
is_empty(Q):    f = = r;
create(Q):      f = r = 0;
```

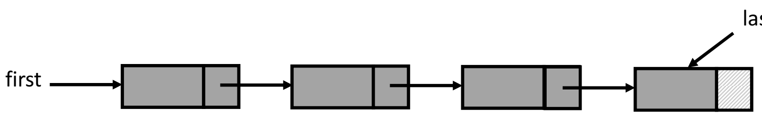
הפעולות האריתמטיות נעשות mod n.
המימוש מלא אם f = = (r +1)% n .

יש לבדוק כי מתקיים
"לא מלא" לפני enqueue
ו"לא ריק" לפני dequeue.

מימוש של תור ע"י רשימה מקושרת



הכנסה בסוף התור (last) :



הוצאה מראש התור (first).
שימו לב שלא ניתן להוציא איבר באמצעות המצביע last.

הערות לגבי מבני נתונים

<u>המשתמש במבנה</u>:
מכיר את הפעולות והשפעתן על הנתונים.
אינו נדרש להכיר את פרטי המימוש.

<u>בזמן פיתוח</u>: ניתן להחליף את מימוש מבני הנתונים מבלי לפגוע בשימושים.
מאפשר לתכנת מימושים פשוטים ואחר כך להחליפם.
ב-++C הדבר קל במיוחד: במקום להגדיר טיפוסים מופשטים (ADT) ב-C כפי שעשינו בשברי הקוד עד כה, ניתן לכתוב מספר מחלקות המממשות את הטיפוס המופשט, ובעת שינוי ממשק פשוט לשנות את ההגדרה (typedef) של הטיפוס המופשט.
<u>לדוגמא</u>:

```
// typedef   ArrayBasedStack  Stack;
typedef   ListBasedStack  Stack;
```

<u>איכות המימוש נקבעת ע"י</u>:
- ניתוח יעילות:
  - זמן – מספר צעדי החישוב הנדרשים, לכל פעולה.
  - מקום – כמות הזיכרון הנדרשת.
- פשטות התכנות (מאפשר אחזקה יעילה).

זמן ריצה של אלגוריתם

זמן ריצה של אלגוריתם A עבור קלט x יסומן ב- $t_A(x)$. זמן הריצה נמדד ע"י מספר פקודות מכונה שהאלגוריתם מבצע על קלט נתון. מדד זה מתעלם מהבדלי המהירות בין הפקודות. (למשל, חבור לעומת כפל).

הגודל של קלט x יסומן ב- $|x|$. לדוגמא, בתוכנית המסכמת איברי מערך x, גודל הקלט הוא מספר איברי המערך.

זמן הריצה הגרוע ביותר (worst case) של אלגוריתם A עבור קלט שגודלו n מוגדר ע"י $t_A(n) = \max\{t_A(x) \mid |x| = n\}$.

<u>דוגמא</u>:

```
sum = 0
for (i = 0; i < n; i++)
        sum = sum + a[i];
```

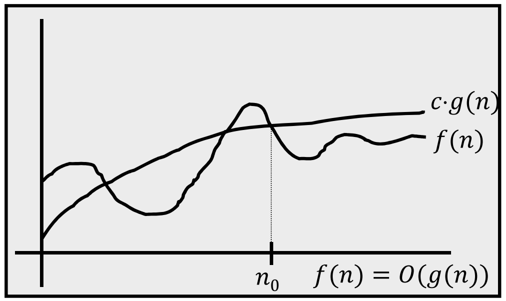
זמן הריצה הגרוע ביותר של אלגוריתם זה מקיים: $n + 1 \le t_{A(n)} \le c_1 \cdot n + c_2$
כאשר c_1, c_2 הם קבועים התלויים במימוש הפקודות בשפת מכונה.

סיבוכיות והסימון O

<u>הגדרה</u>: יהיו $f(n), g(n)$ פונקציות חיוביות. נאמר שהפונקציה $f(n)$ נמצאת בקבוצת הפונקציות $O(g(n))$ אם קיימים קבועים $0 < c$, $0 \le n_0$ כך שלכל $n_0 \le n$ מתקיים:

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

כמו כן נאמר ש- $g(n)$ מהווה <u>חסם עליון אסימפטוטי</u> לפונקציה $f(n)$ ונסמן זאת ע"י $f(n) = O(g(n))$ במקום הסימון הרגיל $f(n) \in O(g(n))$.



נשתמש בסימון זה כאשר הפונקציה $f(n)$ היא פונקציה שקשה לתאר במדויק, למשל זמן הריצה של אלגוריתם, בעוד $g(n)$ פשוטה יותר לתיאור.

דוגמאות פולינומיאליות

<u>טענה</u>: עבור k קבוע מתקיים $f(n) = a_0 + a_1 n + a_2 n^2 + \cdots + a_k n^k = O(n^k)$

<u>הוכחה</u>: נמצא קבועים $0 < c$, $0 \le n_0$ כך שלכל $n_0 \le n$ יתקיים $f(n) \le c \cdot n^k$.

יהי $a_{max} = \max\{1, a_0, .., a_k\}$. אזי

$$f(n) \le a_{max}\left(1 + n + n^2 + \cdots + n^k\right) = a_{max}\frac{n^{k+1}-1}{n-1} \le a_{max}\frac{n^{k+1}}{n/2} = 2a_{max}n^k$$

נכון לכל $2 \le n$

לפיכך נבחר $n_0 = 2$ וכן $c = 2a_{max}$.

<u>עוד דוגמאות</u>:

קבוע:	$f(n) = 10006 = O(1)$
ליניארי:	$f(n) = 4n + 27 = O(n)$
ריבועי:	$f(n) = n\log_2 n + 3n^2 = O(n^2)$

הדוגמא האחרונה נובעת מאי השוויון $\log_2 n < n$ עבור $1 \le n$.

דוגמאות נוספות

<u>לוגריתמי</u>

- לכל a ו- b קבועים מתקיים:

$$f(n) = log_a n = O(log_b n)$$

וזאת מכיוון שמתקיים $log_a n = log_a b \cdot log_b n$.

- לכל ε חיובי (שברים ושלמים) מתקיים:

$$f(n) = log_a n = O(n^{\varepsilon})$$

<u>בין ליניארי וריבועי</u>

$$f(n) = 8 + 15\,n + 9n\log_2 n = O(n \log_2 n)$$

וזאת כיוון שמתקיים $f(n) \le 32\,n \log_2 n$ עבור $2 \le n$.

<u>אקספוננציאלי</u>

$$f(n) = n^k + a^n = O(a^n) \qquad (a > 1)$$

$$f(n) = a^n = O(b^n) \qquad (a \le b)$$

דוגמאות שליליות

האם מתקיים ?

$$f(n) = 3^n = O(2^n)$$

$$f(n) = e^n = O(n^k) \qquad (\text{קבוע } k)$$

$$f(n) = \log n = O(\log\log n)$$

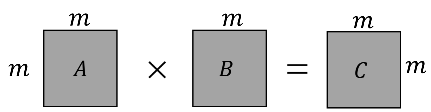
התשובה שלילית בשלושת המקרים. נוכיח את המקרה הראשון.
נניח בשלילה שקיימים קבועים $0 < c$, $0 \le n_0$ כך שלכל $n_0 \le n$ תקיים:
$3^n \le c \cdot 2^n$.
אבל אי שוויון זה אינו מתקיים עבור $n > \log_{3/2} c$. סתירה.

סיבוכיות זמן

<u>דוגמא ראשונה</u>: סכום איברי מערך. ראינו שמתקיים $T(n) \le c_1 \cdot n + c_2$
ולפיכך $T(n) = O(n)$.

<u>דוגמא שניה</u>: כפל מטריצות ריבועיות בגודל $m \times m$.

גודל הקלט $n = 2m^2$.



מחשבים m^2 איברים במטריצת התוצאה C, כאשר כל איבר מחושב לפי ההגדרה:

$$C[i,j] = \sum_{k=1}^{m} A[i,k] \cdot B[k,j]$$

סיבוכיות הזמן כפונקציה של m היא $O(m^3)$.

סיבוכיות הזמן כפונקציה של גודל הקלט, n, היא $O(n^{3/2})$, כיוון שמתקיים:

$$T(n) = O(m^3) = O\left(\frac{n^{3/2}}{2^{3/2}}\right) = O\left(n^{3/2}\right)$$

קבוע

סיבוכיות זמן (המשך)

<u>דוגמא שלישית</u>: חיפוש בינרי של x במערך ממוין בן n איברים.
בכל צעד משווים את x לאיבר האמצעי במערך העכשווי.

- אם האיבר האמצעי שווה ל- x החיפוש נגמר.
- אם האיבר האמצעי גדול מ- x ממשיכים עם חלק המערך המכיל את המספרים הקטנים.
- אם האיבר האמצעי קטן מ- x ממשיכים עם חלק המערך המכיל את המספרים הגדולים.

<u>ניתוח זמן הריצה</u>:
נסמן ב-T את סיבוכיות הזמן כתלות ב-n. מתקיימת <u>משוואת הנסיגה</u> הבאה:

$$T(1) = c_1 \qquad T(n) = c + T(\lceil n/2 \rceil)$$

בהנחה ש- n חזקה שלמה של 2 נקבל:

$$T(n) = c + c + T\left(\frac{n}{4}\right) = \overbrace{c + c + \cdots + c}^{i} + T\left(\frac{n}{2^i}\right) =$$

$$= c + c + \cdots + c + T\left(\frac{n}{2^{\log_2 n}}\right) = c\log_2 n + T(1) = O(\log n)$$

הטענה נכונה גם ללא ההנחה על n. פרטים על שיטות פתרון למשוואות נסיגה ניתן למצוא בספר הלימוד ובתרגולים.

סיבוכיות זמן (המשך)

דוגמא רביעית:

```
S = 0;
for ( i = 1;  i < n;  i ++)
          for ( j = 0;  j< n;  j + = i )
                    S++ ;
```

אנליזה גסה: שתי לולאות מקוננות. $T(n) \le n\,(n-1) = O(n^2)$.

אנליזה עדינה:

כאשר $i = 1$,	מתבצעות n פעולות.
$i = 2$,	$\lfloor n/2 \rfloor$
$i = 3$,	$\lfloor n/3 \rfloor$
$\vdots$	$\vdots$
$i = n$,	$\lfloor n/n \rfloor = 1$

וסה"כ: $T(n) \le n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n} = n\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right) = n \cdot H_n$

$\boxed{H_n \text{ נקרא המספר ההרמוני ה- } n}$

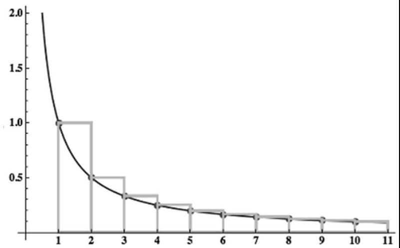
סיבוכיות זמן (המשך)

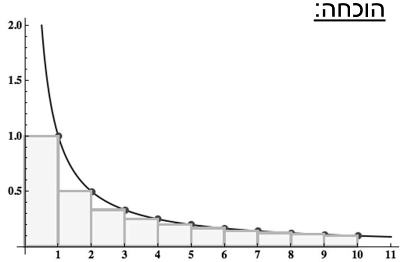
בדוגמא בשקף הקודם קיבלנו:

$$T(n) \le n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n} = n\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right) = n \cdot H_n$$

טענה: $\ln(n+1) = \int_1^{n+1} \frac{dx}{x} \le \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \le 1 + \int_1^{n} \frac{dx}{x} = 1 + \ln(n)$

הוכחה:





$$\ln(n+1) = \int_1^{n+1} \frac{dx}{x} \le \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

$$\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \le 1 + \int_1^{n} \frac{dx}{x} = 1 + \ln(n)$$

סיבוכיות זמן (המשך)

בדוגמא לפני שני שקפים קיבלנו:

$$T(n) \le n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n} = n\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right) = n \cdot H_n$$

וכיוון שמתקיים: $H_n = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \le 1 + \ln(n)$

מכאן נקבל כי:

$$T(n) \le nH_n \le n(1 + \ln(n)) \le n(2\ln(n)) = 2n\ln(n) = O(n\log n)$$

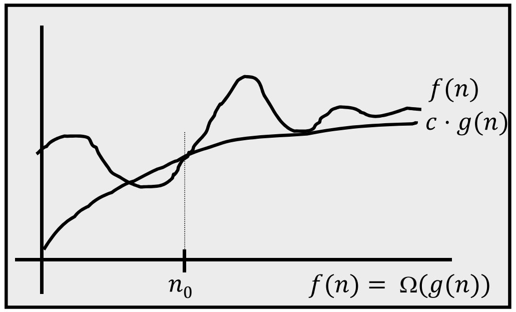
$\boxed{\text{נכון לכל } e \le n}$

הערה: לשם דיוק היה עלינו לכפול את כל המשוואות בקבוע. בכל מקרה, היינו מקבלים $T(n) = O(n \cdot \log n)$

חסם תחתון אסימפטוטי

הגדרה: יהיו $f(n), g(n)$ פונקציות חיוביות. נאמר שהפונקציה $f(n)$ נמצאת בקבוצת הפונקציות $\Omega(g(n))$ (אומגה) אם קיימים קבועים $0 < c$, $0 \le n_0$ כך שלכל $n_0 \le n$ מתקיים:

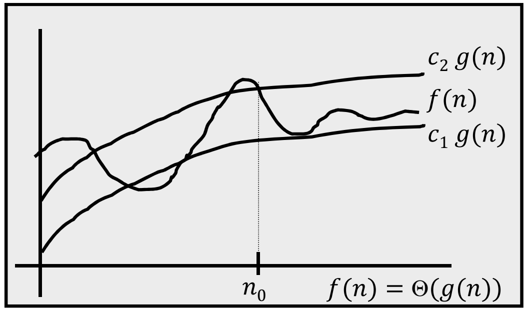
$$f(n) \ge c \cdot g(n)$$



קיימות הגדרות נוספות לחסם תחתון אשר שקולות להגדרה לעיל עבור פונקציות מונוטוניות.

חסם הדוק אסימפטוטי

<u>הגדרה</u>: יהיו $f(n), g(n)$ פונקציות חיוביות. נאמר שמתקיים $f(n) = \Theta(g(n))$ (תטה) אם $f(n) = O(g(n))$ וגם $f(n) = \Omega(g(n))$.



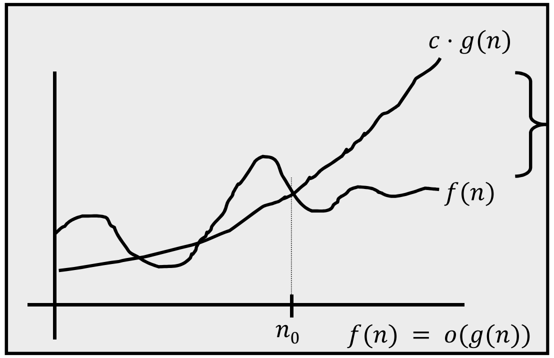
<u>הגדרה (אקוויולנטית)</u>: נאמר שמתקיים $f(n) = \Theta(g(n))$ אם קיימים קבועים $0 \le n_0, 0 < c_1, 0 < c_2$ כך שלכל $n_0 \le n$ מתקיים:

$$c_1\cdot g(n) \le f(n) \le c_2\cdot g(n)$$

הסימון o קטן

<u>הגדרה</u>: יהיו $f(n), g(n)$ פונקציות חיוביות. נאמר שהפונקציה $f(n)$ נמצאת בקבוצת הפונקציות $o(g(n))$ אם <u>לכל קבוע</u> $0 < c$ קיים קבוע $0 \le n_0$ כך שלכל $n_0 \le n$ מתקיים:

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



$f(n)$ זניחה אסימפטוטית יחסית לפונקציה $g(n)$

<u>הגדרה (אקוויולנטית)</u>: נאמר שמתקיים $f(n) = o(g(n))$ אם $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

<u>דוגמאות</u>: $\log n = o(n), \quad n - 100 \neq o(n)$

מגבלות הסימון האסימפטוטי

נוח להשתמש בסימונים אסימפטוטיים מפני שהסימון מתעלם מקבועים ומאפשר ניתוח זמנים פשוט יותר. בנוסף, סימונים אלו מאפשרים לבחור אלגוריתמים ישימים יותר עבור קלטים גדולים. אנו נשתמש בסימונים אלה לאורך הקורס.
אבל לסימון יש מגבלות מסוכנות...
ברור שנעדיף תוכנית הרצה בזמן $T(n) = n^2$ על פני תוכנית הרצה בזמן קבוע של 10^{160} (ממומשת למשל ע"י טבלה ענקית של תשובות לכל אפשרות) כיוון שבתוכניות ממשיות אנו משתמשים בגודל קלט n סופי הקטן באופן משמעותי מ-10^{80} (מספר האטומים ביקום).

<u>מסקנה</u>: צריך לוודא שהקבועים n_0, c המתחבאים בהגדרות האסימפטוטיות O, Θ, Ω אמנם "סבירים".

<u>דוגמא</u>. נניח שאלגוריתם A רץ בזמן $T_A(n) = 100\,n$ ואילו אלגוריתם B רץ בזמן $T_B(n) = 5n\log_2 n$. בניתוח אסימפטוטי עדיף אלגוריתם A כיוון שהאלגוריתם ליניארי, אבל עבור קלטים המקיימים $n < 2^{20}$ עדיף אלגוריתם B.