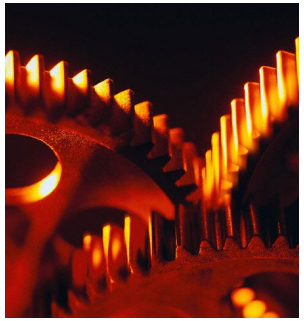
פרק 12
סיבוכיות

מורכבות של אלגוריתמים

כל תוכנית מחשב צורכת שני משאבים במהלך ריצתה:

(1) זמן מעבד (2) מרחב זיכרון



- כאשר כותבים תוכנית, חשוב לשקול את כמות משאבי החישוב שהיא צורכת.
- ישנן בעיות חישוביות עם פתרונות מצוינים, אבל הזמן שנדרש להרצתם, אפילו על מחשבי-על, הוא אלפי (או מיליוני) שנים. האם ניתן להחשיב זאת כפתרון סביר?

מדדי סיבוכיות

- על מנת לנתח את כמות המשאבים שתוכנית מחשב כלשהי דורשת, ראשית עלינו להגדיר את אופן המדידה של המשאבים.

- <u>זמן חישוב</u> ניתן למדוד באמצעות:
  - זמן הריצה בשניות
  - מספר פעולות המעבד שמתבצעות
- <u>כמות זיכרון</u> ניתן למדוד על ידי ספירת מספר הבתים שהתוכנית דורשת לריצתה.
- כפי שמייד נראה, הגדרות אלו מעט רופפות... נציג עתה מספר תכונות אותן היינו רוצים שהמדדים שלנו יקיימו, ונשפר בהדרגה את המדדים.

זמן כמספר הפעולות בתוכנית

<u>תכונה 1: אי תלות בסוג המעבד</u>

מעבדים שונים פועלים במהירויות שונות, מה שמשפיע במידה ניכרת על הזמן הדרוש לריצת התוכנית. אנו נרצה שהמדדים שלנו יאפיינו את **התוכנית בלבד**, ולא יושפעו מגורמים "סביבתיים" כמו סוג המעבד.

<u>פתרון</u>: נמדוד זמן לא באופן אבסולוטי, אלא **כמספר הפעולות** שהתוכנית מבצעת במהלך ריצתה.

עקרונית, יש כאן צורך להגדיר מהי בדיוק "פעולה". במדעי החישוביות, העוסקים בנושא זה, מגדירים בצורה מסודרת מודל סטנדרטי של מחשב, ומנתחים תוכניות במונחים של מודל זה. עם זאת, בקורס שלנו נסתפק בלומר שהכוונה לפעולות אלמנטריות שהמעבד מבצע בזמן קצר וקבוע, כמו פעולות חשבון, השמות, השוואות וכו'.

דוגמאות לספירת פעולות

```
a = x+y+z;
```

- פקודה זו מבצעת שתי פעולות חיבור ופעולת השמה אחת, ובסך הכול 3 פעולות.

```
for (i=0; i<10; ++i) {
  sum = sum+i;
}
```

- בקוד זה יש אתחול (i=0), ולאחר מכן 10 איטרציות שבכל אחת 4 פעולות: בדיקת תנאי, חיבור, השמה, וקידום של i. הקוד מסתיים כאשר i=10, אז מתבצעת השוואה נוספת שנכשלת, והלולאה מסתיימת. נקבל בסך הכול: 4·10+2=42 פעולות.

מדדי סיבוכיות, ניסיון ראשון

- בהינתן תוכנית כלשהי, נרצה לחשב עבורה את שני הגדלים הבאים:
  - T: מספר הפעולות שהתוכנית מבצעת.
  - S: כמות הזיכרון (בבתים) שהתוכנית דורשת.
- <u>בעיה</u>: T ו-S עשויים להשתנות על פי הקלט לתוכנית! למשל, ננסה לספור כמה פעולות מבצעת הפונקציה הבאה:

```
int summarize(int a[], int n)
{
  int sum=0, i;
  for (i=0; i<n; ++i)
    sum = sum+a[i];
  return sum;
}
```


מדדי סיבוכיות, ניסיון ראשון

- <u>פתרון אפשרי</u>: נגדיר **פונקציות** זמן ומקום לכל קלט אפשרי X:
  - T(X): מספר הפעולות שהתוכנית מבצעת עבור הקלט X.
  - S(X): כמות הזיכרון שהתוכנית דורשת עבור הקלט X.
- ואולם, למדידת סיבוכיות באופן זה יש בעיות רבות:
  - הפונקציות הללו בדרך כלל סבוכות מאוד, ודרוש מאמץ רב על מנת לחשב אותן (אם בכלל זה אפשרי..)
  - יתירה מכך, לא ברור כלל שהפונקציות הללו באמת מועילות! על פי רוב הן יהיו סבוכות כל כך, שאנו נתקשה להסיק מהן משהו על ההתנהגות הכללית של התוכנית.

פישוט הפונקציות T ו-S

<u>תכונה 2: פשטות התלות בקלט</u>

נרצה מדדי סיבוכיות שמתארים את השפעת הקלט על המשאבים הדרושים לתוכנית, אך באופן פשוט ואינפורמטיבי ככל הניתן.

<u>פתרון</u>: לכל קלט אפשרי X נתאים מספר טבעי, N(X), שייצג את "המורכבות" של X (נראה דוגמאות מייד). עתה, במקום לחשב את T(X) ו-S(X) לכל קלט אפשרי, נתאר אותן כפונקציה של מורכבות הקלט N בלבד.

פישוט הפונקציות T ו-S

- כיצד נבחר את N(X)? ובכן, מקובל לייצג באמצעות N את **אורך הקלט**. ואולם, לעיתים ישנה יותר מאפשרות אחת לעשות כן! נתבונן במספר דוגמאות:
- מערך: במקרה זה מקובל לבחור את N(X) כמספר האיברים במערך.
- מחרוזת: גם במקרה זה N(X) הוא בדרך כלל אורך המחרוזת. ואולם, ניתן גם לחשוב אפשרויות נוספות, כמו למשל מספר המילים במחרוזת.
- מערך דו-ממדי: על פי רוב, נבחר את N(X) להיות מספר האיברים הכולל במערך הדו-ממדי. לחילופין, עבור מערך דו-ממדי ריבועי, ניתן גם לבחור את N(X) כאורך הצלע של המערך.
- מספר שלם: כשהקלט הוא מספר, N(X) יכול להיות המספר עצמו, מספר הספרות העשרוניות שלו, או אף מספר הביטים בייצוג הבינארי שלו.

פישוט הפונקציות T ו-S

- משבחרנו את N(X), **פונקציות הסיבוכיות** מוגדרות כך:

> סיבוכיות זמן T(N): כמות הפעולות הנדרשות לתוכנית עבור קלטים באורך N (או באופן כללי יותר, קלטים עם "מורכבות" N).
>
> סיבוכיות מקום S(N): כמות הזיכרון הנדרשת בתוכנית עבור קלטים באורך (מורכבות) N.

- חשוב לשים לב שעבור בחירות שונות של N(X), נקבל פונקציות סיבוכיות **שונות**. לפיכך, עבור כל פונקצית סיבוכיות, יש למעשה לציין גם את ההגדרה של הערך N שמופיע בה. עם זאת, בפועל ההגדרה של N היא הרבה פעמים ברורה, ואין צורך לציין אותה.

דוגמאות לפונקציות סיבוכיות

```
int summarize(int a[], int n)
{
  int sum=0, i;
  for (i=0; i<n; ++i)
    sum = sum+a[i];
  return sum;
}
```

הצהרות על משתנים אינן נחשבות כפעולות בחישובי זמן ריצה – אלא אם יש אתחולים, שנספרים כהשמות

- הפונקציה מקבלת מערך **a** ואת אורכו **n**. נגדיר את N להיות אורך המערך **a**. כעת, בכל איטרציה של הלולאה מתבצעות 5 פעולות: בדיקת התנאי, גישה למערך, חיבור, השמה, וקידום **i**. הלולאה עצמה מתבצעת N פעמים. נוסיף את האתחולים ואת הבדיקה ה-N+1 בסוף הלולאה, ונקבל:

$$T(N) = 5N + 3$$

דוגמאות לפונקציות סיבוכיות

```
void average(int matrix[N][M])
{
  int i, j;
  for (i = 0; i < N; ++i)
    for (j = 0; j < M; ++j)
      printf("value: %d\n", matrix[i][j]);
}
```

- הערה: נחשיב הדפסת תו בודד, וכן משתנה מספרי למסך, כפעולה בודדת. לכן, שורת ה-**printf()** מערבת 10 פעולות: גישה למערך, הדפסת 8 תווים קבועים והדפסת משתנה מספרי.
- נכתוב את סיבוכיות זמן הריצה כתלות בממדי המערך N, M. נקבל:

$$T(N,M) = (12 \cdot M + 4) \cdot N + 2$$

איזה קלט?

- בכל הדוגמאות הנ"ל, ההתנהגות של הפונקציה הייתה אחידה עבור כל קלט באורך N. אולם, מה לגבי הדוגמה הבאה?

```
int sum_absolute(int a[], int n)
{
  int sum=0, i;
  for (i=0; i<n; ++i)
    sum = sum + (a[i]>0 ? a[i] : -a[i]);
  return sum;
}
```

פעולה נוספת עבור אברי מערך שליליים

- כלומר, בפישוט T ו-S יצרנו בעיה חדשה: ייתכן שעבור שני קלטים באותו האורך, התוכנית תרוץ זמן שונה, או תדרוש כמות זיכרון שונה! במילים אחרות, T ו-S החדשות כלל אינן מוגדרות היטב...

איזה קלט?

עבור קלטים באורך N מסוים, קיימות מספר דרכים מקובלות להגדיר את T(N) ו-S(N). הגישות השונות מלמדות על מאפיינים שונים של התנהגות האלגוריתם.

1. <u>גישת המקרה הגרוע ביותר</u>:

   על פי גישה זו, T(N) ו-S(N) מוגדרים להיות כמות הזמן והמקום הדרושים עבור **הקלט הגרוע ביותר** באורך N. כלומר, פונקציות אלה מתארות את כמות המשאבים המקסימאלית לה אנו נידרש עבור קלטים באורך N.

   שימו לב שהקלט הגרוע ביותר עבור T(N) עשוי להיות **שונה** מהקלט הגרוע ביותר עבור S(N).

איזה קלט?

2. <u>גישת המקרה הנפוץ</u>:

   גישה זו בוחנת את התנהגות האלגוריתם עבור **מרבית הקלטים**. כלומר, אנו מתעלמים מהמקרים הקיצוניים, ומתעניינים מה קורה בדרך כלל – מהי כמות המשאבים הנדרשת עבור קלט "סביר"?

3. <u>גישת המקרה הממוצע</u>:

   גישה זו בוחנת את כמות המשאבים שהאלגוריתם דורש **בממוצע** על פני כל הקלטים באורך N. כלומר, אם נריץ את האלגוריתם פעמים רבות על קלטים שונים, כמה "בערך" ידרוש האלגוריתם בכל ריצה? שימו לב שבניגוד לגישה הקודמת, כאן איננו מתעלמים מהמקרים הקיצוניים, אלא אנו ממצעים על פני כל הקלטים האפשריים.

סוגי קלטים: דוגמה

```
int sum_absolute(int a[], int n)
{
  int sum=0, i;
  for (i=0; i<n; ++i)
    sum = sum + (a[i]>0 ? a[i] : -a[i]);
  return sum;
}
```

- <u>המקרה הגרוע ביותר</u> הינו מערך שמכיל כולו ערכים שליליים (ואז יש להפוך את כל ערכיו לפני הסכימה). לכן, בגישת המקרה הגרוע ביותר נקבל:

$$T(N) = 7 \cdot N + 3$$

סוגי קלטים: דוגמה

- עבור שתי הגישות האחרות, עלינו להניח הנחות סטטיסטיות על הקלט. למשל, אם איננו יודעים דבר על הקלט נוכל להניח כי 50% מן הערכים חיוביים, ו-50% שליליים. במצב זה בחצי מהפעמים הלולאה הפנימית תבצע 6 פעולות (איבר חיובי) ובחצי מהפעמים היא תבצע 7 פעולות. לכן, גישת המקרה הממוצע תיתן:

$$T(N) = 6.5 \cdot N + 3$$

- לעומת זאת, אם הקלט שלנו מכיל 99% מערכים חיוביים לחלוטין ו-1% שליליים לחלוטין, גישת המקרה הממוצע תיתן:

$$T(N) = 6.01 \cdot N + 3$$

- ואילו גישת המקרה הנפוץ פשוט תיתן $T(N) = 6 \cdot N + 3$

סיכום ביניים

- הגישה שעל פי רוב היא הפשוטה ביותר לחישוב הינה גישת "המקרה הגרוע", וזאת כיוון שהיא בוחנת את התנהגות התוכנית עבור קלט בודד, ואינה דורשת ניתוח סטטיסטי. גישה זו היא גם הנפוצה ביותר, וזו הגישה שנשתמש בה בקורס זה. לפיכך, מעתה נגדיר:

$T(N)$: זמן הריצה המקסימאלי הדרוש עבור קלטים באורך N.

$S(N)$: כמות הזיכרון המקסימאלית הדרושה עבור קלטים באורך N.

אלו יהיו המדדים אותם נרצה לחשב עבור התוכניות שלנו.

- עם זאת, כפי שמייד נראה, על פי רוב לא יהיה לנו צורך לחשב פונקציות אלה במדויק, ונוכל להסתפק בשערוך שלהן.

ניתוח אסימפטוטי של הסיבוכיות

תכונה 3: אי תלות במימוש

נרצה מדדי סיבוכיות שבאמצעותם נוכל לאפיין מורכבות לא רק של תוכניות C, אלא גם של **אלגוריתמים**. כלומר, נרצה דרך לתאר את כמויות המשאבים שדורש אלגוריתם כלשהו בלא שאנו תלויים בקוד המדויק שמממש אותו, ובאופן שאינו רגיש לפרטי המימוש הטכניים (כמו סוגי הלולאות, הטיפוסים של המשתנים ועוד).

פתרון: כדי לתת מענה לבעיה זו, לא נתעניין יותר בחישוב המדויק של הפונקציות T ו-S, ובמקום זאת, ניתן **קירוב אסימפטוטי** שלהן. קירוב זה הינו עיוור לפרטי מימוש טכניים, ונותן אפיון הרבה יותר כללי של האלגוריתם.

התנהגות אסימפטוטית של פונקציה

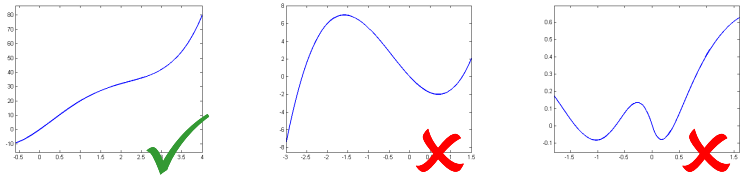
- במקום חישוב מדויק של T ו-S, נרצה הערכה **לסדר הגודל** שלהן.
- הכלי הדרוש לנו נקרא ניתוח אסימפטוטי של פונקציות. ניתוח זה בוחן את **קצב הצמיחה** של פונקציה כאשר אנו מגדילים את N.
- השאלה שנשאל, בעצם, הינה **עד כמה נעשית הבעיה קשה כאשר מגדילים את הקלט?** כלומר, האם עבור קלט ארוך יותר הזמן הנדרש כמעט ולא גדל, או שמה הוספת ולו ביט אחד לקלט מזניקה את זמן הריצה עשרות מונים?

מדדים אסימפטוטיים של פונקציה

המדדים האסימפטוטיים מתייחסים לפונקציות **חיוביות ומונוטונית לא-יורדות** בלבד.

-פונקציה f(n) היא <u>חיובית</u> אם לכל n טבעי, f(n)>0.

-פונקציה f(n) היא <u>מונוטונית לא-יורדת</u> אם עבור כל שני טבעיים m>n, מתקיים f(m)≥f(n).



מדדים אסימפטוטיים של פונקציה

- נשים לב שבכל תוכנית סבירה (ובהנחה שבחרנו את מדד המורכבות N(X) כראוי), קלט מורכב יותר יגדיל את דרישות החישוב. לכן, הפונקציות T(N) ו-S(N) הן אמנם חיוביות ומונוטוניות.
- כעת, עבור פונקציה כללית f(n) חיובית ומונוטונית, נגדיר 3 מדדים:

<u>מדד O</u>: **חסם עליון** על ההתנהגות של f(n); מדד זה מתאר בעצם מה "הכי גרוע" שיכולה להיות f(n).

<u>מדד Ω</u>: **חסם תחתון** להתנהגות של f(n); מדד זה מתאר מה "הכי טוב" שניתן לצפות מ-f(n).

<u>מדד Θ</u>: **חסם הדוק** עבור ההתנהגות של f(n); מדד זה מתאר בדיוק כיצד מתנהגת f(n).

מדדים אסימפטוטיים: הגדרה

- בטבלה הבאה מפורטות ההגדרות של שלושת המדדים האסימפטוטיים.
- מדד אסימפטוטי מסוים מתקיים אם קיים קבוע C חיובי (או שני קבועים חיוביים C_1 ו-C_2), כך שאי השוויון הרשום מתקיים לכל n החל מ-N טבעי כלשהו.
- $g(n)$ נקראת גם **קרוב אסמפטוטי** של $f(n)$.

מדד אסימפטוטי	לכל $n \geq N$ צריך להתקיים:
$f(n)=O(g(n))$	$f(n) \leq C \cdot g(n)$
$f(n)=\Omega(g(n))$	$f(n) \geq C \cdot g(n)$
$f(n)=\theta(g(n))$	$C_1 \cdot g(n) \leq f(n) \leq C_2 \cdot g(n)$

דוגמאות להתנהגות אסימפטוטית

- נניח אלגוריתם <u>שלכל קלט</u> תמיד מבצע אותו מספר k של פעולות. כלומר, לכל n:

$$T(n)=k$$

- קל להראות שבמקרה זה:

$$T(n)=\theta(1)$$

- <u>הוכחה</u>: נבחר $c_1=k-1,\ c_2=k+1$, ואז מתקיים:

$$(k-1)\cdot 1 \ \leq\ T(n)\ \leq\ (k+1)\cdot 1$$

- <u>מסקנה</u>: כל אלגוריתם בעל זמן ריצה קבוע (שאינו תלוי בגודל הקלט) הוא בעל זמן ריצה מסדר גודל $\theta(1)$.

דוגמאות להתנהגות אסימפטוטית

- נניח אלגוריתם שבמקרה הגרוע ביותר מבצע מספר **פולינומיאלי** של פעולות. כלומר, זמן הריצה שלו הוא מהצורה

$$T(n) = a_d \cdot n^d + a_{d-1} \cdot n^{d-1} + \ldots + a_1 \cdot n + a_0$$

- במקרה זה, T הוא מסדר הגודל של החזקה הגדולה ביותר:

$$T(n) = \theta(n^d)$$

- <u>הוכחה</u>: לשם כך עלינו למצוא c_1 ו-c_2 המקיימים את אי השוויון הבא לכל $n \geq N$:

$$c_1 \cdot n^d \leq \left(a_d \cdot n^d + a_{d-1} \cdot n^{d-1} + \ldots + a_0\right) \leq c_2 \cdot n^d$$

דוגמאות להתנהגות אסימפטוטית

$$c_1 \cdot n^d \leq \left(a_d \cdot n^d + a_{d-1} \cdot n^{d-1} + \ldots + a_0\right) \leq c_2 \cdot n^d$$

- נחלק ב-n^d:

$$c_1 \leq \left(a_d + \frac{a_{d-1}}{n} + \ldots + \frac{a_1}{n^{d-1}} + \frac{a_0}{n^d}\right) \leq c_2$$

- עבור n-ים גדולים מספיק, כל השברים בסוגריים נעשים קטנים מאוד! למעשה, ישנו N מסוים שהחל ממנו הסכום שלהם **כולו** נעשה קטן מ-1 (בערך מוחלט), ואז אנו יודעים ש:

$$a_d - 1 \leq \left(a_d + \frac{a_{d-1}}{n} + \ldots + \frac{a_1}{n^{d-1}} + \frac{a_0}{n^d}\right) \leq a_d + 1$$

דוגמאות להתנהגות אסימפטוטית

$$a_d - 1 \leq \left(a_d + \frac{a_{d-1}}{n} + \ldots + \frac{a_1}{n^{d-1}} + \frac{a_0}{n^d}\right) \leq a_d + 1$$

- נכפיל בחזרה ב-n^d, ונקבל שלכל $n \geq N$ מתקיים:

$$\left(a_d - 1\right) \cdot n^d \leq T(n) \leq \left(a_d + 1\right) \cdot n^d$$

ושני המספרים שכופלים את n^d משני צדי אי השוויון הם בדיוק הקבועים c_1 ו-c_2 שחיפשנו!

- <u>מסקנה</u>: כל פולינום הוא מסדר הגודל של החזקה הגבוהה ביותר בו.

דוגמאות להתנהגות אסימפטוטית

- נניח כעת אלגוריתם שבמקרה הגרוע ביותר דורש מספר **אקספוננציאלי** (מעריכי) של פעולות. זהו אלגוריתם שזמן הריצה שלו הוא מהצורה:

$$T(n) = a_0 \cdot r_0^{\,n} + a_1 \cdot r_1^{\,n} + \ldots + a_d \cdot r_d^{\,n} + P(n)$$

כאשר P(n) הוא פולינום כלשהו, והערכים $r_0 \ldots r_d$ הינם מספרים חיוביים מסודרים בסדר יורד (r_0 הגדול ביותר).

- במקרה זה ניתן לחלק ב-$r_0^{\,n}$, ולהראות (בהוכחה דומה מאוד) ש:

$$T(n) = \theta(r_0^{\,n})$$

- <u>מסקנה</u>: סכום של חזקות ופולינומים הוא מסדר הגודל של החזקה עם הבסיס הגדול ביותר.

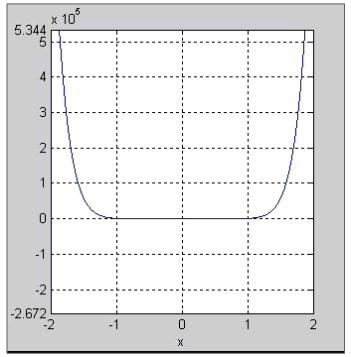
דוגמאות להתנהגות אסימפטוטית

כמה דוגמאות להתנהגות אסימפטוטית של פולינומים ואקספוננטים:

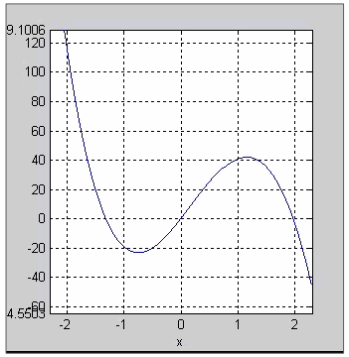
$T(n)=5$	$T(n)=\theta(1)$
$T(n)=3n+2$	$T(n)=\theta(n)$
$T(n)=n^3+4n+10$	$T(n)=\theta(n^3)$
$T(n)=2\cdot 3^n+n^{50}+2n$	$T(n)=\theta(3^n)$

הדגמה: התנהגות אסימפטוטית

$\frac{1}{1000}2^n+1000n^{10}$

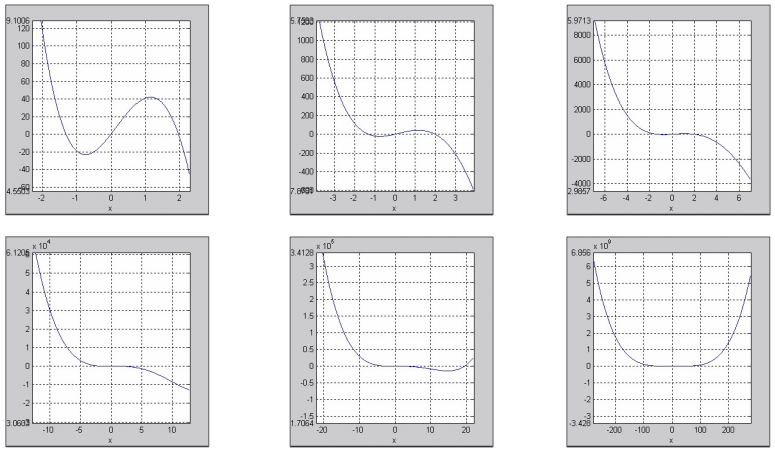


$n^4-20n^3+10n^2+50n$



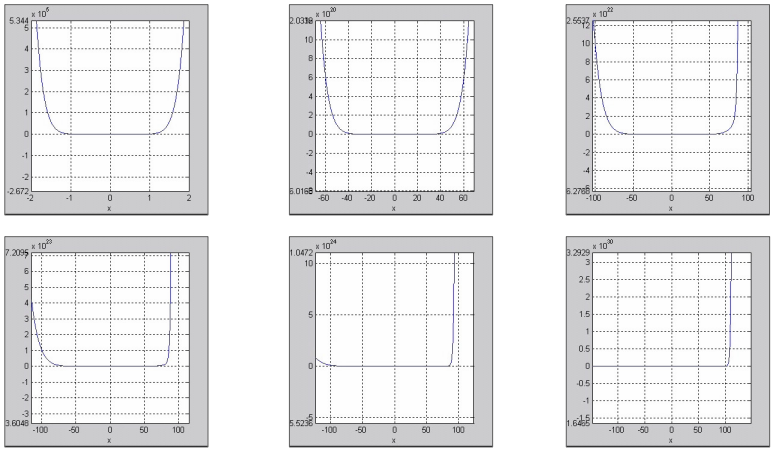
הדגמה: פולינום בסקאלות שונות

ממרחק מספיק גדול, $n^4-20n^3+10n^2+50n$ נראה בדיוק כמו n^4 !



הדגמה: אקספוננט בסקאלות שונות

וממרחק מספיק גדול, $\frac{1}{1000}2^n+1000n^{10}$ נראה בדיוק כמו 2^n !



כללי אצבע לסדרי גודל של פונקציות

סדר גודל עולה

$\theta(1)$	$\theta(n^2)$
$\theta(\log n)$	$\theta(n^3)$
$\theta(\log^2(n))$	$\theta(2^n)$
$\theta(n^{1/2}) = \theta(\sqrt{n})$	$\theta(3^n)$
$\theta(n)$	$\theta(3^n n^2)$
$\theta(n \log n)$	$\theta(n!)$
$\theta(n \log^2(n))$	$\theta(n^n)$

דוגמאות להתנהגות אסימפטוטית

דוגמאות נוספות להתנהגות אסימפטוטית של פונקציות (הסבירו!):

$T(n) = \log_2 n + \log_5 n$	$T(n) = \theta(\log_2 n) = \theta(\log_5 n)$
$T(n) = n + n \log n$	$T(n) = \theta(n \log n)$
$T(n) = n^2 \log n + n^3$	$T(n) = \theta(n^3)$
$T(n) = \log(n^3)$	$T(n) = \theta(\log n)$
$T(n) = 3^{2n}$	$T(n) = \theta(9^n)$
$T(n) = 2^n + (\log n)^n$	$T(n) = \theta((\log n)^n)$

ניתוח סיבוכיות זמן של תוכנית

- הפונקציה **f1()** מחפשת איבר במערך **a** שנמצא גם ב-**b**.

```
int f1(int a[], int b[], int n)
{
  int i, j;

  for(i=0; i < n; i++) {
    for(j = 0; j < n; j++)
      if(a[i] == b[j])
        return i;
  }
  return -1;
}
```

$\Theta(n^2)$

ניתוח סיבוכיות זמן של תוכנית

- מה יקרה לסיבוכיות אם נחליף את **n** ב-**i** בלולאה הפנימית?

```
int f1(int a[], int b[], int n)
{
  int i, j;

  for(i=0; i < n; i++) {
    for(j = 0; j < i; j++)
      if(a[i] == b[j])
        return i;
  }
  return -1;
}
```

$\Theta(n^2)$

ניתוח סיבוכיות זמן של תוכנית

- הפונקציה **f2()** מקבלת מספר טבעי ומחלקת אותו ב-2 עד שמתקבל 1.
- רמז: על מנת לנתח פונקציה זו, דרך אחת היא לחשוב כיצד מתנהג הייצוג הבינארי של **n** לאורך הלולאה.

```
void f2(int n)
{
  while( n > 1)
    if(n % 2)
      n /= 2;
    else
      n += 1;
}
```

$\Theta(\log n)$

ניתוח סיבוכיות זמן של תוכנית

- הפונקציה **f3()** מקבלת מספר טבעי ומבצעת סדרת לולאות ריקות...
- רמז: כיצד היה משתנה מספר האיטרציות בלולאה השנייה לו החלפנו את הערך 6 בערך 1? או בערך 100?

```
void f3(int n)
{
  int i, m=1;

  for(i = 0; i < n; i++)
    m *= n;

  while( m > 6)
    m /= 3;
}
```

$\Theta(n \cdot \log n)$

ניתוח סיבוכיות זמן של תוכנית

- הפונקציה **f4()** סופרת את מספר האיטרציות שהיא מבצעת:

```
int f4(int n)
{
  int i, j, k=1, count;

  for(i = 0; i < n; i++) {
    k *= 3;
    for(j = k; j; j /= 2)
      count++;
  }
  return count;
}
```

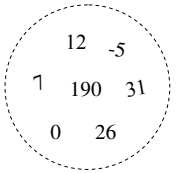
$\Theta(n^2)$

דוגמה: ניתוח סיבוכיות של אלגוריתם

- נביא עתה דוגמה לניתוח הסיבוכיות של **אלגוריתם**, ללא פירוט הקוד שממש אותו.
- נדון בבעיה שיש לה מספר פתרונות אפשריים: הבעיה של **ייצוג קבוצות** וביצוע פעולות על קבוצות אלה.
- אנו נראה כי יש יותר מפתרון אחד נכון! לכל פתרון יהיו היתרונות והחסרונות היחסיים שלו, והבחירה בסופו של דבר באיזה פתרון להשתמש תלויה בסדר העדיפויות שלנו מבחינת משאבי חישוב.
- למשל, מצב נפוץ מאוד הוא **מאזן בין משאבי זיכרון וזמן**. אנו נראה כי ניתן לפתור את הבעיה מהר יותר כאשר יש לנו יותר זיכרון, או לחילופין לחסוך בזיכרון על חשבון זמן ריצה ארוך יותר.

ייצוג קבוצות

- **קבוצה** הינה אוסף של איברים (במקרה שלנו מספרים שלמים), ללא סדר בין האיברים. האיברים בקבוצה בהכרח **שונים** זה מזה, כלומר לא ייתכן שאותו איבר יופיע בקבוצה "פעמיים".
- **גודל הקבוצה** הוא מספר האיברים שהיא מכילה.
- **קבוצה ריקה** היא קבוצה שאינה מכילה איברים כלל (גודלה אפס).
- אנו נרצה אפשרות לייצג קבוצות במחשב, ולהיות מסוגלים לבצע עליהן את הפעולות הבאות:
  - להוסיף ולהוציא איברים מהקבוצה
  - לבדוק אם איבר מסוים נמצא בקבוצה
  - למזג שתי קבוצות לקבוצה אחת חדשה



קבוצות במחשב: אופציה I

- **כיצד נייצג קבוצה של מספרים שלמים במחשב?**
- אופציה I: נאחסן את כל האיברים **במערך**.
- מעשית, אנו נניח שהקבוצות שלנו מכילות לכל היותר N איברים. אנו נקצה את המערך מראש בגודל המקסימאלי N, ואת אברי המערך נאחסן בתחילת המערך. נעזר במשתנה נוסף, נניח X, על מנת לציין את מספר האברים בקבוצה, ואברי הקבוצה לפיכך יהיו במקומות ה-0 ועד ה- X-1 במערך.
- עבור בחירה זו של ייצוג קבוצות, כיצד נבצע את 4 הפעולות שהגדרנו? ומה הסיבוכיות של כל פעולה כזו?

קבוצות במחשב: אופציה I

- נתאר את **האלגוריתם** לביצוע כל אחת מן הפעולות:

1. הוספת איבר לקבוצה:
   א. סרוק את המערך מתחילתו ועד סופו.
   ב. אם לא נמצא האיבר, כתוב אותו בסוף המערך ועדכן את גודל המערך בהתאם

   סיבוכיות זמן: $\Theta(n)$

2. הסרת איבר מהקבוצה:
   א. סרוק את המערך מתחילתו ועד סופו.
   ב. אם נמצא האיבר, הזז את כל האיברים שאחריו במערך מקום אחד אחורנית, תוך דריסת האיבר.

   סיבוכיות זמן: $\Theta(n)$

קבוצות במחשב: אופציה I

3. מציאת איבר בקבוצה:
   א. סרוק את המערך מתחילתו ועד סופו.
   ב. אם נמצא האיבר – החזר 1, אחרת החזר 0

   סיבוכיות זמן: $\Theta(n)$

4. מיזוג שתי קבוצות:
   א. העתק את תוכן המערך הראשון למערך היעד.
   ב. סרוק את המערך השני איבר-איבר. לכל איבר, חפש אותו במערך הראשון, ואם הוא לא נמצא בו, כתוב את האיבר למערך היעד. אחרת, המשך ללא כתיבת האיבר ליעד.

   סיבוכיות זמן: $\Theta(n \cdot m)$

נניח גודל מערך אחד הוא n וגודל מערך שני הוא m

קבוצות במחשב: אופציה II

- אופציה שנייה לייצוג קבוצה במחשב: נאחסן את אברי הקבוצה במערך כמו באופציה I, אך נאחסן אותם **ממוינים**.
- כפי שנראה, שינוי זה משפר משמעותית את הסיבוכיות של הפעולות שאנו מבצעים על הקבוצות.

1. הוספת איבר לקבוצה:
   א. סרוק את המערך איבר-איבר, עד שנמצא המקום בו יש להוסיף את האיבר החדש כך שהמערך יישאר ממוין.
   ב. הזז את כל האיברים הנותרים במערך מקום אחד קדימה, וכתוב את האיבר החדש במקום שהתפנה.

סיבוכיות זמן: $\Theta(n)$

קבוצות במחשב: אופציה II

2. הסרת איבר מהקבוצה:
   א. חפש את האיבר באמצעות חיפוש בינארי.
   ב. אם נמצא האיבר, הזז את כל האיברים שאחריו במערך מקום אחד אחורנית, תוך דריסת האיבר.

סיבוכיות זמן: $\Theta(n)$

3. מציאת איבר בקבוצה:
   א. חפש את האיבר באמצעות חיפוש בינארי.
   ב. אם נמצא האיבר – החזר 1, אחרת החזר 0

סיבוכיות זמן: $\Theta(\log n)$

קבוצות במחשב: אופציה II

4. מיזוג שתי קבוצות:
   א. באמצעות פונקציית merge שלמדנו.

סיבוכיות זמן: $\Theta(n+m)$

- לסיכום, האופציה השנייה אמנם טובה משמעותית מן הראשונה:

	אופציה I	אופציה II
הוספת איבר	$\Theta(n)$	$\Theta(n)$
הסרת איבר	$\Theta(n)$	$\Theta(n)$
חיפוש איבר	$\Theta(n)$	$\Theta(\log n)$
מיזוג קבוצות	$\Theta(n \cdot m)$	$\Theta(n+m)$

קבוצות במחשב: אופציה III

- האופציה השלישית לייצוג קבוצה במחשב שונה למדי.
- בייצוג זה, אנו נניח כי האיברים שאנו רוצים לאחסן בקבוצה הינם בטווח 0..k-1 **בלבד**.
- נייצג את הקבוצה כמערך בגודל k. התא ה-i במערך יכיל 1 אם המספר i נמצא בקבוצה, ו-0 אם אינו נמצא בקבוצה.
- למשל, אם הקבוצה שלנו מכילה את המספרים 2,3 ו-5 אזי התאים 2,3 ו-5 במערך יכילו 1, והיתר יכילו 0.

0	0	1	1	0	1	0	...

קבוצות במחשב: אופציה III

1. הוספת איבר לקבוצה:
   א. סמן את התא המתאים במערך ב-1.
   סיבוכיות זמן: Θ(1)

2. מציאת איבר בקבוצה:
   א. בדוק את התא המתאים במערך, אם הוא מכיל 1 החזר 1, אחרת החזר 0.
   סיבוכיות זמן: Θ(1)

3. הסרת איבר מהקבוצה:
   א. סמן את התא המתאים במערך ב-0.
   סיבוכיות זמן: Θ(1)

קבוצות במחשב: אופציה III

4. מיזוג שתי קבוצות:
   א. סרוק את המערך הראשון. בכל תא בו נמצא 1, כתוב 1 למקום המתאים במערך היעד.
   ב. סרוק את המערך השני. בכל תא בו נמצא 1, כתוב 1 למקום המתאים במערך היעד.
   סיבוכיות זמן: Θ(k)

- באלגוריתם זה רוב הפעולות מתבצעות באופן יעיל מאוד! היכן המלכוד?
- ובכן, המלכוד הוא שכל מערך תופש **גודל k בזיכרון**. לפיכך, אם טווח האיברים שאנו רוצים לאחסן הוא קטן – הרווחנו. ואולם על פי רוב אין זה המצב, ואז נשלם בתצרוכת זיכרון **גדולה מאוד**.

סיכום: סיבוכיות של ייצוג קבוצות

	אופציה I	אופציה II	אופציה III
הוספת איבר	Θ(n)	Θ(n)	Θ(1)
הסרת איבר	Θ(n)	Θ(n)	Θ(1)
חיפוש איבר	Θ(n)	Θ(log n)	Θ(1)
מיזוג קבוצות	Θ(n·m)	Θ(n+m)	Θ(k)
זיכרון	Θ(n)	Θ(n)	Θ(k)