Complexity and Performance

זה לא קורס באלגוריתמים

- המטרה של המפגש הזה היא הכרת הנושא של יעילות אלגוריתם על קצה המזלג
- נציג בשיעור את הנושא של הסיבוכיות
- נציג דוגמא או שתיים של אלגוריתמים פרימיטיביים וננסה לייעל אותם
- אין מטרתו של קורס זה לדון לעומק בסיבוכיותם ויעילותם של אלגוריתמים
- את זה נשאיר לקורסים מתקדמים יותר - אשר נלמדים במהלך לימודי התאר.

Algorithm
אלגוריתם

- דרך שיטתית לביצוע משימה מסוימת על נתונים במספר סופי של צעדים
- שיטה (מכאנית) לפתרון בעיה חישובית
- *מערכת חוקים ברורה וחד-משמעית המפרטת צעדים רצופים לפתרון בעיה חישובית
- בכל תוכנית מחשב חבוי לפחות אלגוריתם אחד
- לכל אלגוריתם יש הגדרה מדויקת ולכן ניתן לממשו במחשב
- על האלגוריתם לקיים שני תנאים
  - ✓ עבור כל קלט נתון האלגוריתם מגיע לסופו
  - ✓ פלט נכון
- כלומר לכל קלט חוקי שהאלגוריתם מקבל הוא נותן פלט (דטרמיניסטיות)
- כל אלגוריתם פותר אוסף אינסופי של קלטים ולא רק קלט בודד
- דוגמאות
  - ✓ אלגוריתם למיון רשימת מספרים
  - ✓ האלגוריתם של אוקלידס למציאת המחלק המשותף הגדול ביותר של שני מספרים (GCD)


Input
Algorithm
Output

Complexity
סיבוכיות

- תת תחום במדעי המחשב אשר בוחן את:
- ✓ יעילות האלגוריתם
- ✓ המשאבים הנחוצים לפתרון בעיה חישובית (משאב הזיכרון ,הזמן,...)
- אנו נתמקד בעיקר בסיבוכיות הזמן (מספר הפעולות הבסיסיות הדרושות כפונקציה של גודל הקלט)

Elementary Operations
פעולות בסיסיות

- Elementary operation : פעולת מחשב בסיסית (אטומית) אשר מתבצעת במחשב במספר קבוע של צעדים

- Basic arithmetic operations (+ , – , ∗ , / , %)
- Basic relational operators (==, !=, >, <, >=, <=)
- Basic Boolean operations (AND,OR,NOT)
- Branch operations, return, …

Efficiency
of Algorithms

- יעילות של אלגוריתם נמדדת בעזרת מדדי סיבוכיות כגון סיבוכיות קוד, זמן,זיכרון,..
- זמן הריצה של אלגוריתם (תוכנית) מושפע מכמה גורמים
  - ✓ יעילות המעבד :משנה את זמן הריצה בגורם קבוע
  - ✓ יעילות המהדר : משנה את זמן הריצה בגורם קבוע
  - ✓ יעילות הקידוד :משנה את זמן הריצה בגורם קבוע
  - ✓ יעילות האלגוריתם :תלויה במספר הפעולות הבסיסיות שהאלגוריתם מבצע (פונקציה של אורך הקלט)
- אנו נתמקד בסיבוכיות הזמן והזיכרון
  - ✓ סיבוכיות זמן הריצה: של בעיה נתונה הוא מספר הפעולות הבסיסיות (פעולות אטומיות) שהאלגוריתם מבצע.
  - ✓ סיבוכיות זיכרון (מקום) :שטח הזיכרון שהאלגוריתם דורש (מס' משתנים, גודל מבנה הנתונים...)

סיבוכיות מקום

- במחשב יישנו משאב חשוב ביותר וזהו משאב הזיכרון
- כאשר אלגוריתם רץ הוא צורך זיכרון מהמחשב
- סיבוכיות המקום של אלגוריתם כלשהו היא סדר הגודל של מספר תאי הזיכרון שמנצל האלגוריתם בזמן ריצתו
- בנוסף לסיבוכיות הזמן של האלגוריתם אנו נדרשים לבדוק את סיבוכיות המקום שלו. סיבוכיות המקום נמדדת לפי מס' המשתנים, גודל מבנה הנתונים (מערך, רשימה מקושרת, ...) שהאלגוריתם דורש.

דוגמא 1:

```
int i, n;
for(i=0; i<n; i++)
    System.out.print("*");
```

- **מספר האטריציות של הלולאה הוא n**
- **סדר גודל זמן הריצה של האלגוריתם הנתון O(n)**
- **האלגוריתם מגדיר שני תאי זיכרון מסוג int כלומר גודל הזיכרון הוא מספר קבוע וקטן לכן סיבוכיות המקום היא O(1)**

```java
int i,n;
int[] a = new int[n];
for (i=0;i<n; i++)
    System.out.println(A[i]);
```

דוגמא 2:

- מספר האטירציות של הלולאה הוא n
- סדר גודל זמן הריצה של האלגוריתם הנתון O(n)
- האלגוריתם מגדיר n+3 תאי זיכרון מסוג int

כלומר גודל הזיכרון הוא פונקצה של n ולכן סיבוכיות המקום היא O(n)

```java
public void print(int[] a) {
   int i , n;
   n=a.length;
   for (i=0;i<n; i++)
    System.out.println(A[i]);
}
```

דוגמא 3

נתונה השיטה הבאה

- מספר האטירציות של הלולאה הוא n
- סדר גודל זמן הריצה של האלגוריתם הנתון O(n)
- האלגוריתם מגדיר 3 תאי זיכרון מסוג int

כלומר גודל הזיכרון הוא מספר קבוע ולכן סיבוכיות המקום היא O(1)

שימו לב!!!

אמנם השיטה מקבלת כפרמטר מצביע למערך בגודל n אבל השיטה לא מגדירה את המערך בעצמה ולפיכך אינה אחראית לזיכרון שנצרך כתוצאה מהגדרת מערך זה. השיטה תופסת מקום קבוע ולכן סיבוכיות המקום היא O(1) ולא O(n)

Running Time
זמן ריצה

- סיבוכיות זמן הריצה: של בעיה נתונה נמדדת לפי מספר הפעולות הבסיסיות (פעולות אטומיות - elementary operation) שהאלגוריתם מבצע.
- זמן הריצה של אלגוריתם עבור קלט בגודל n מסומן ב - T(n) .
- הערה: בחישוב סיבוכיות זמן של אלגוריתם צריך להתעלם מגורמים קבועים

דוגמא 1

נתון מערך מספרים בגודל n
צריך לחשב את סכום איבריו

$$sum = \sum_{i=1}^{n} A[i]$$

הערה: בשפות תכנות האיבר הראשון במערך נמצא במקום i=0

$$sum = \sum_{i=1}^{n} A[i]$$

פתרון

Naïve Algorithm
Input: An array of numbers A[n]
Output: The sum of the array numbers

```
sum ← 0
for each item i in A do
  sum ← sum + A[i]
end for
A←{1,2,1,1} ➔ sum←5
```

elementary Operation : + , =

סיבוכיות האלגוריתם
שמחשב סכום מספרים במערך

- מספר פעולות החיבור (+) הדרוש לחיבור מספרים במערך A[n] הוא n
- פעולות החיבור וההשמה (+,=) באלגוריתם המתואר חוזרות על עצמן n פעמים
- בכל פעם מחשבים sum ← sum + A[i]
- לכן זמן הריצה של האלגוריתם הוא ליניארי (תלויי בגודל הקלט) :

$$\Rightarrow \text{Running Time } T(n) = c \cdot n \quad \text{is linear in } n$$

בעיית תת הסדרה המקסימלית

דוגמא 2

נתון מערך A של מספרים שלמים באורך n
מצא את זוג האינדקסים :

(i,j) , i ≤ j

כך ש $\sum_{ii=i}^{j} A[ii]$ הוא מקסימלי

דוגמא: $1, \quad 2, \quad -10 \quad 4, \quad 5, \quad -7, \quad 6$

i j

Straight Forward Solution

Naïve Algorithm

פתרון נאיבי

```
(i_max , j_max) ← (-1,-1)
maxSum ← - ∞

for i=1 to n do
    for j=i to n do
        sum ← 0
        for k=i to j do
            sum ← sum + A[k]
        end for
        if sum > maxSum then
            maxSum ← sum
            (i_max , j_max) ← (i, j)
        end if
    end for
end for
```

ניתוח זמן הריצה
של האלגוריתם הנאיבי

בהינתן אלגוריתם כלשהו - היינו יכולים להיות מאד מרוצים לו והינו מצליחים לחשב את מספר הפקודות שהמחשב מבצע בעקבות הרצת תוכנית המבוססת על אותו אלגוריתם. מטרה זו היא כמעט בלתי אפשרית להשגה - אבל בהחלט ניתן לחשב בקירוב את המספר הזה.

הבא ונסתכל באלגוריתם שלנו. הבא ונתמקד אך ורק במספר פעולות החיבור שהוא מבצע. (מספר הפעולות הכולל שהאלגוריתם מבצע הוא גדול יותר ממספר פעולות החיבור) בהינתן זוג אינדקסים (i,j) נחשב את מספר פעולות החיבור :

i=1,j=1 → 1
i=1,j=2 → 2
i=1;j=3 → 3
:
i=1;j=n → n

i=2,j=1 → 0
i=2,j=2 → 1
i=2;j=3 → 2
:
i=2;j=n → n-1

i=3,j=1 → 0
i=3,j=2 → 0
i=3;j=3 → 1
:
i=3;j=n → n-2

i=n,j=1 → 0
i=n,j=2 → 0
i=n;j=3 → 0
:
i=n;j=n → 1

סיכום של פעולות החיבור שהאלגוריתם מבצע

	i=1	2	3	4	5	.	.	.	n
j=1	1								
2	2	1							
3	3	2	1						
4	4	3	2	1					
5	5	4	3	2	1				
.	.	.	.	.	.	1			
.	.	.	.	.	.	.	1		
.	.	.	.	.	.	.	.	1	
n	n	n-1	n-2	n-3	.	.	.	.	1

$$1 \cdot n + 2 \cdot (n-1) + 3 \cdot (n-2) + 4 \cdot (n-3) + \cdots\cdots + n \cdot 1$$

$$1 \cdot n + 2 \cdot (n-1) + 3 \cdot (n-2) + \ldots + n \cdot 1 =$$

$$= \sum_{i=1}^{n} \left[(n+1) - i\right] \cdot i =$$

$$= \sum_{i=1}^{n} \left[(n+1) \cdot i - i^2 \right] =$$

$$= (n+1) \sum_{i=1}^{n} i - \sum_{i=1}^{n} i^2 =$$

$$= \frac{(n+1)n(n+1)}{2} - \frac{n(n+1)(2n+1)}{6} =$$

$$= \frac{3(n+1)n(n+1) - n(n+1)(2n+1) =}{6}$$

$$= \frac{n(n+1)\left[3(n+1) - (2n+1)\right]}{6} =$$

$$= \frac{n(n+1)(n+2)}{6} = \frac{n^3 + 3n^2 + 2n}{6} \approx \frac{n^3_{n \to \infty}}{6}$$

אלגוריתם משופר
לבעיית תת-הסדרה המקסמלית

הרעיון : אם חברנו את תת הסדרה מ i עד j אז נוסיף לסכום שהתקבל מקודם את המספר במקום j+1 במערך A[j+1] ובכך נקבל את הסכום של תת-הסדרה החדשה שנמצאת במקום i עד j+1 . דוגמא:

1	2	1	-1	1	4

i=1 j=3 j+1

Sum(i , j) = 1 + 2 + 1 = 4
Sum(i , j+1) = sum(i,j) + A[j+1] = 4 + (-1) = 3

הפתרון המשופר

```
maxSum ← - ∞
for i=1 to n do
    sum←0
    for j=i to n do
        sum ← sum + A[ j ]
        if sum > maxSum then
            maxSum ← sum
            (i_max , j_max) ← (i, j)
        end if
    end for
end for
```

ניתוח זמן הריצה
של האלגוריתם המשופר

הבא ונתמקד במספר האיטרציות שהאלגוריתם מבצע:

i=1,j=1 →	n
i=2,j=2 →	n-1
i=3;j=3 →	n-2
:	:
i=n;j=n →	1

$$\Rightarrow n + (n-1) + (n-2) + \cdots\cdots 1 =$$

$$= \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \approx \frac{n^2_{n\to\infty}}{2}$$

Algorithm For Finding the Minimum

דוגמא 3

Input: An array of n numbers - A
Output: The minimum number in A

אלגוריתם א'

1. מיין את המערך בסדר עולה
2. החזר את האיבר הראשון במערך

אלגוריתם ב'

```
min ← ∞
for i←1 to n
    if A[i] < min then
        min ← A[i]
        place←i
    end if
end for
return min and place
```

ניתוח זמן הריצה
של האלגוריתם א' ו- ב'

אלגוריתם א'

תלוי בשיטת המיון של המערך

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

אלגוריתם ב'

מספר הפעולות הבסיסיות שהאלגוריתם מבצע

$T_{worst}(n) = 1 + n + n + n + n + 1 = 4n + 2$

$T_{best}(n)=1+n+n+1+1+1=2n+4$

$\Rightarrow \Theta(n)$

How To Compare Algorithms
איך להשוות בין אלגוריתמים

- בדוגמאות הקודמות חישבנו את מספר הפעולות הבסיסיות המתבצעות ע"י האלגוריתם.
- האם צריך לכלול בחישוב את כל הפעולות שהאלגוריתם מבצע?
- איך אנחנו יכולים להשוות בין יעילותם של אלגוריתמים?
- לשם מטרה זו אנחנו נשתמש במונחים של סדרי גודל (O, Ω, Θ)
- בחישוב סיבוכיות של אלגוריתמים אנחנו מתעלמים מהקבועים ומהאיברים בעלי סדר נמוך - את זמן הריצה של האלגוריתם קובעים לפי האיבר בעל הסדר הגבוה ביותר

לעיתים זמן הריצה בפועל של שני אלגוריתמים הוא שונה אבל הסיבוכיות שלהם זהה.
לדוגמא : נניח שיש לנו שני אלגוריתמים A ו- B אשר פותרים בעיה מסויימת עם זמני הריצה הבאים :

$$T_A(n) = n^2 + 1000$$

$$T_B(n) = n^2 + 10$$

$$T_A(n) \neq T_B(n)$$

$$\Theta(T_A(n)) = \Theta(T_B(n)) = \Theta(n^2)$$

שוויון אסימפטותית

אנו אומרים כי לשני האלגוריתמים
סיבוכיות זמן $\Theta(n^2)$

Worst, Best and Average Case

הגדרה:

- Worst case זמן הריצה הגדול ביותר של האלגוריתם על קלט כלשהו בגודל n
- Best case זמן הריצה הקטן ביותר של האלגוריתם על קלט כלשהו בגודל n
- Average case זמן הריצה הממוצע של האלגוריתם מעל כל הקלטים האפשריים בגודל n
- בדרך כלל נהוג להתמקד במקרה הגרוע ביותר

הערה1: בכל פעם שאנו בודקים סיבוכיות של אלגוריתם אנו חייבי לציין באיזה מקרה מדובר - הטוב ביותר, הגרוע ביותר או המקרה הממוצע.

הערה2: לרוב - המקרה החשוב ביותר הוא המקרה הגרוע ביותר (למה?)

קצבי הגידול של פונקציות

- אנו מעוניינים למצוא דרך שבעזרתה נוכל להשוות את ההתנהגות של פונקציות. לשמחתנו ישנה דרך מאד פשוטה לעשות זאת והיא :

✓בהינתן פונקציה כלשהי נשווה את הפונקציה לקבוצת פונקציות בסיסיות המוכרות לנו.

✓הפונקציות הבסיסיות החשובות ביותר הן :

1. $f(n) = k \cdot n \quad , k = 1,2,3 \ldots$ ← **ליניאריות**
2. $f(n)=\log^{k}_{n} \quad ,k=1,2,3\ldots$ ← **לוגריתמיות**
3. $f(n)=n^{k} \quad ,k=1,2,3\ldots$ ← **מונומים**
4. $f(n) = a^{n^{k}} \quad , k = 1,2,3 \ldots$ ← **מעריכיות**

השוואת קצבי גידול

(O, Ω, Θ)

משתמשים בסימונים האלה כדי להשוות בין קצבי הגידול של פונקציות שונות כאשר n שואף לאינסוף

1. O ← **חסם עליון אסימפטוטי**
2. Ω ← **חסם תחתון אסימפטוטי**
3. Θ ← **שוויון (חסם הדוק אסימפטוטי)**

דוגמא:

נתון כי זמן הריצה של אלגוריתם A הוא:

$$T_A(n) = 10n^3 + 100n^2 + 5n + 1000$$

$$\Rightarrow \lim_{n\to\infty} (T_A(n)) = O(n^3)$$

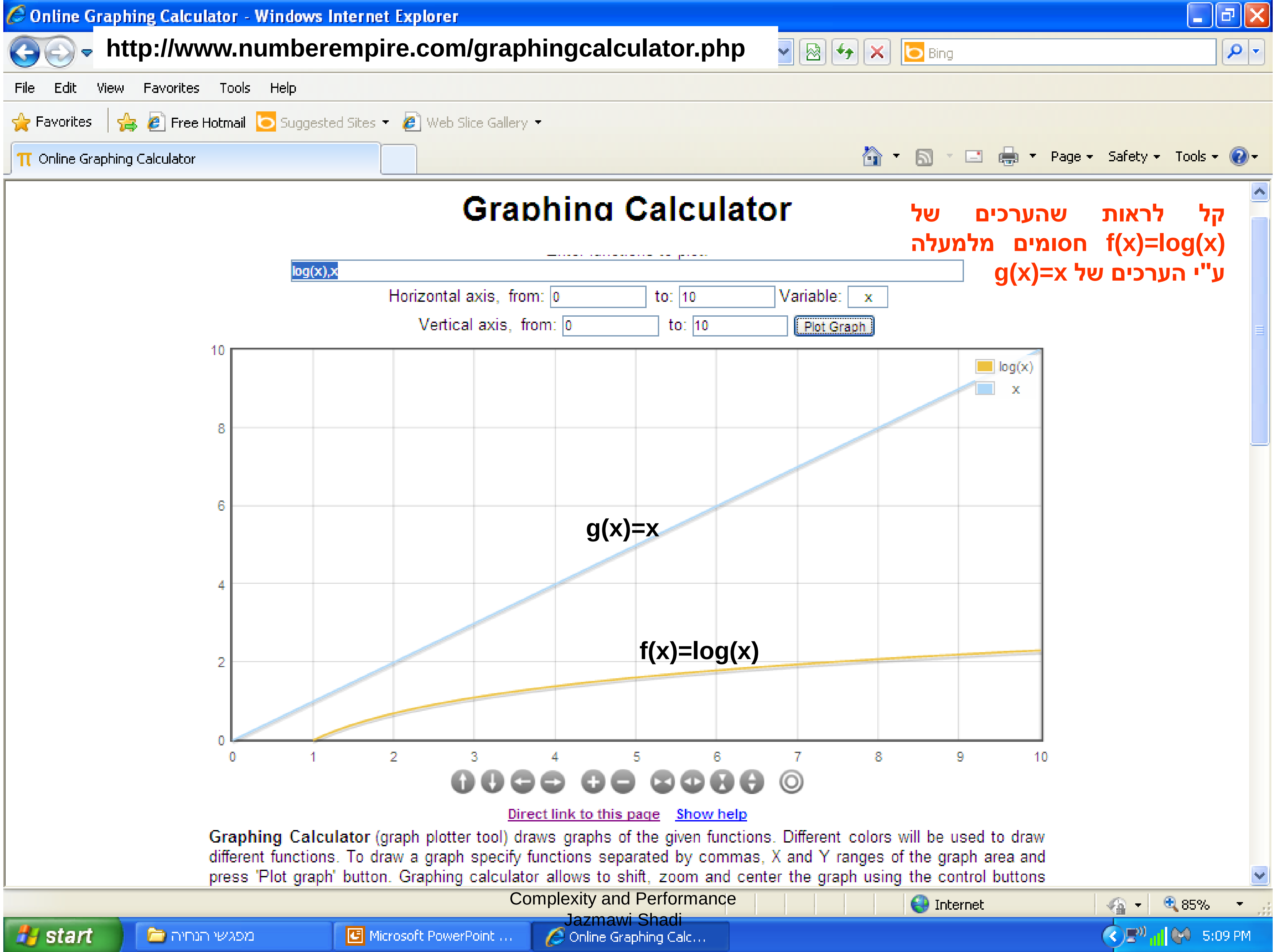
קל לראות כי כאשר n שואף לאינסוף

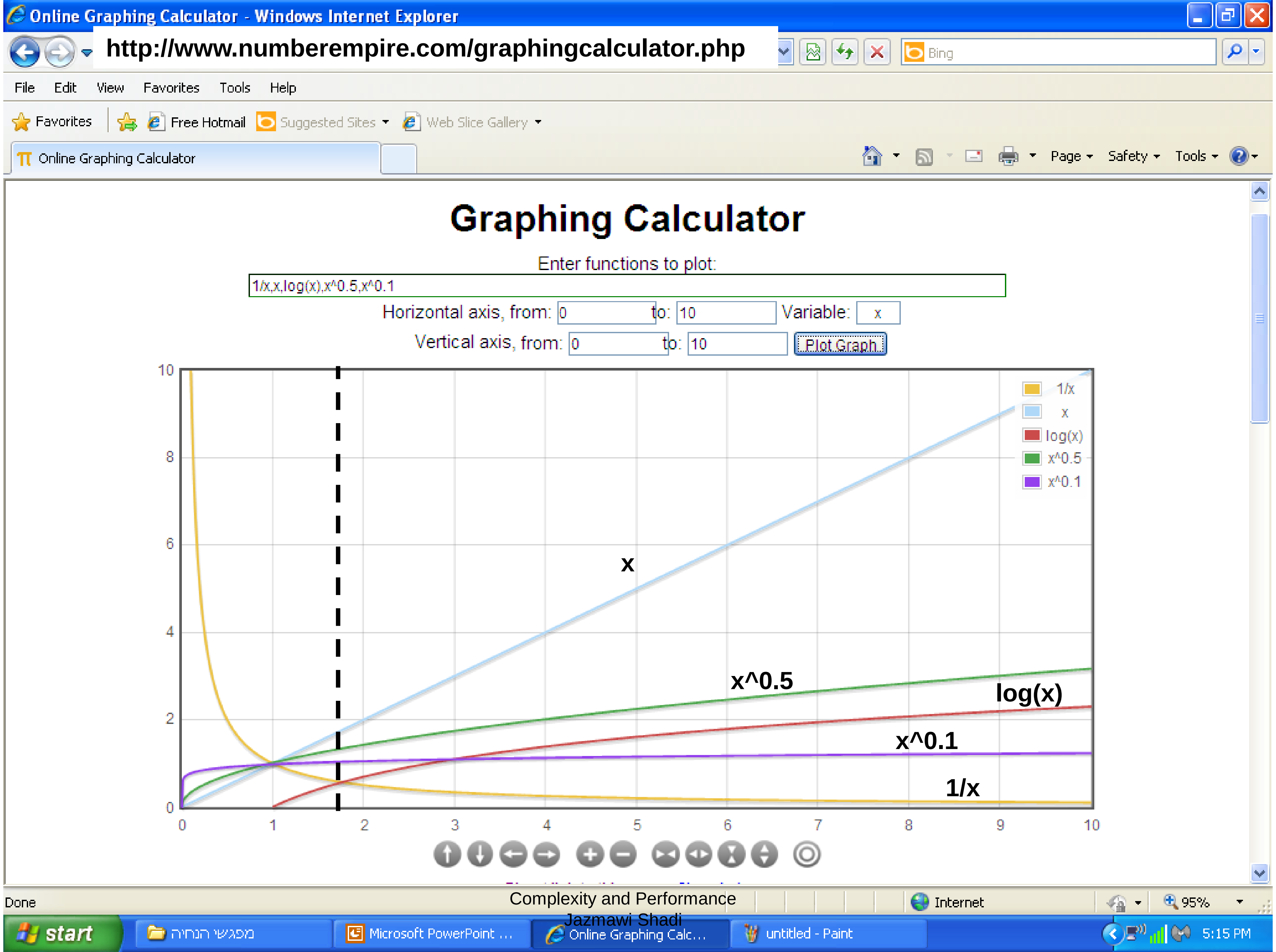
ערך של T(n) נשלט ע"י n^3

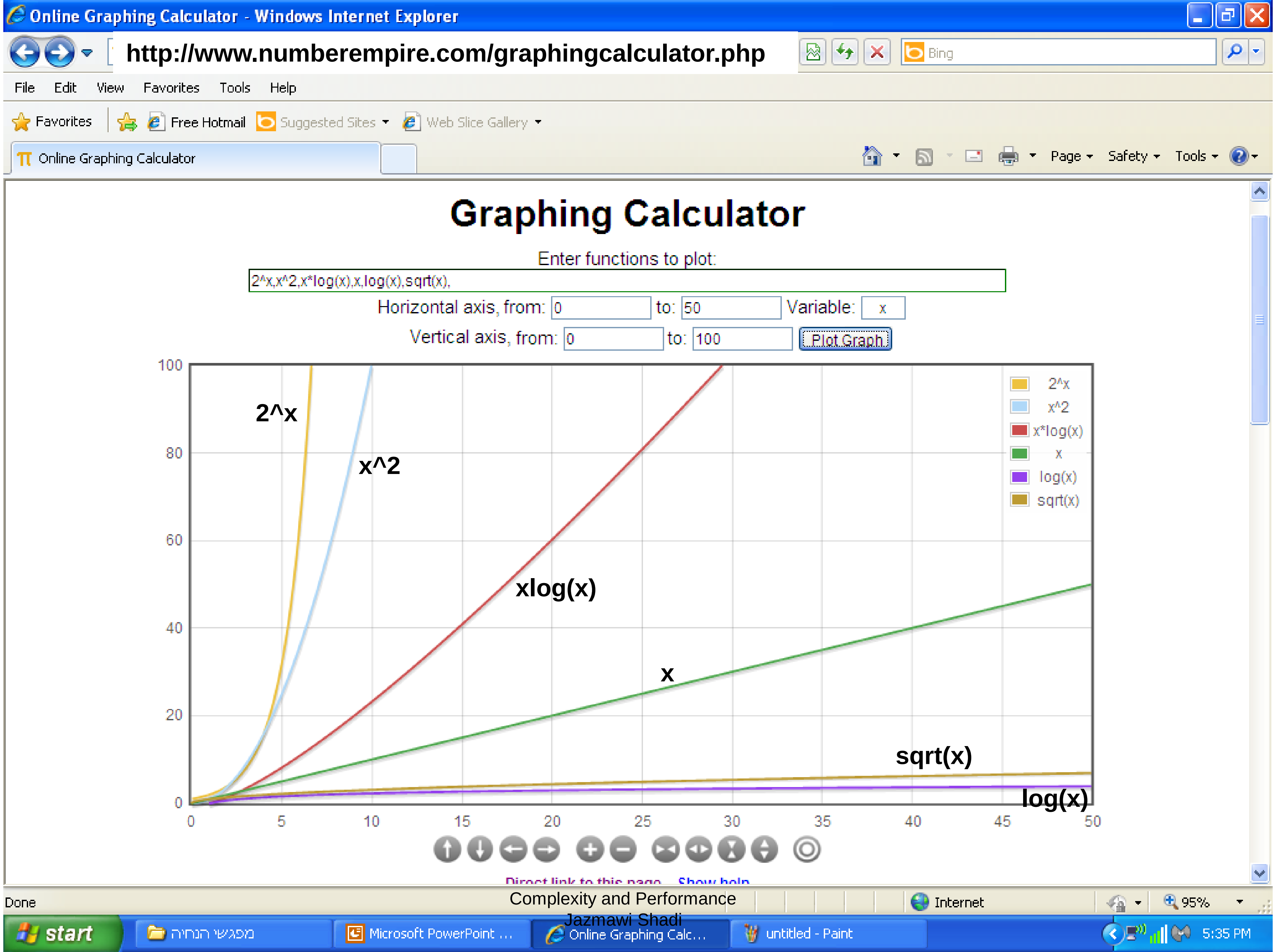
או בלשון מתמטית $T(n) = O(n^3)$

http://www.numberempire.com/graphingcalculator.php

קל לראות שהערכים של f(x)=log(x) חסומים מלמעלה
ע"י הערכים של g(x)=x




Online Graphing Calculator - Windows Internet Explorer
http://www.numberempire.com/graphingcalculator.php
Graphing Calculator
Enter functions to plot:
1/x,x,log(x),x^0.5,x^0.1
Horizontal axis, from: 0 to: 10 Variable: x
Vertical axis, from: 0 to: 10
Plot Graph
1/x
x
log(x)
x^0.5
x^0.1
x
x^0.5
log(x)
x^0.1
1/x


Online Graphing Calculator - Windows Internet Explorer
http://www.numberempire.com/graphingcalculator.php
Graphing Calculator
Enter functions to plot:
2^x,x^2,x*log(x),x,log(x),sqrt(x),
Horizontal axis, from: 0 to: 50 Variable: x
Vertical axis, from: 0 to: 100
Plot Graph
2^x
x^2
xlog(x)
x
sqrt(x)
log(x)
2^x
x^2
x*log(x)
x
log(x)
sqrt(x)
0
20
40
60
80
100
0
5
10
15
20
25
30
35
40
45
50

$T(n) \;\alpha\; c$ constant

$T(n) \;\alpha\; \log(n)$ logarithmic

$T(n) \;\alpha\; n$ linear

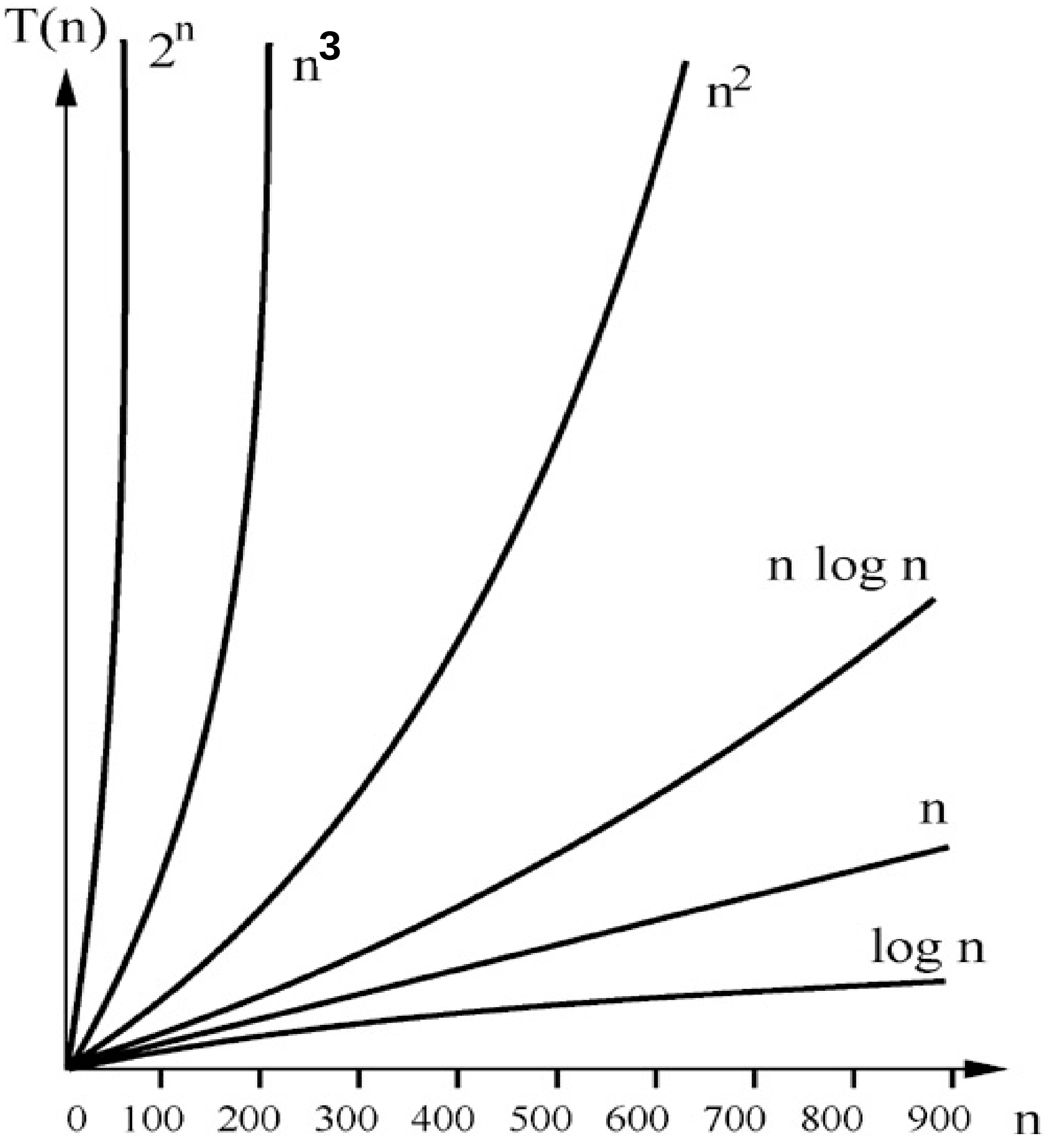
$T(n) \;\alpha\; n \cdot \log(n)$

$T(n) \;\alpha\; n^2$ quadratic

$T(n) \;\alpha\; n^3$ cubic

$T(n) \;\alpha\; n^k$ polynomial

$T(n) \;\alpha\; 2^n$ exponential

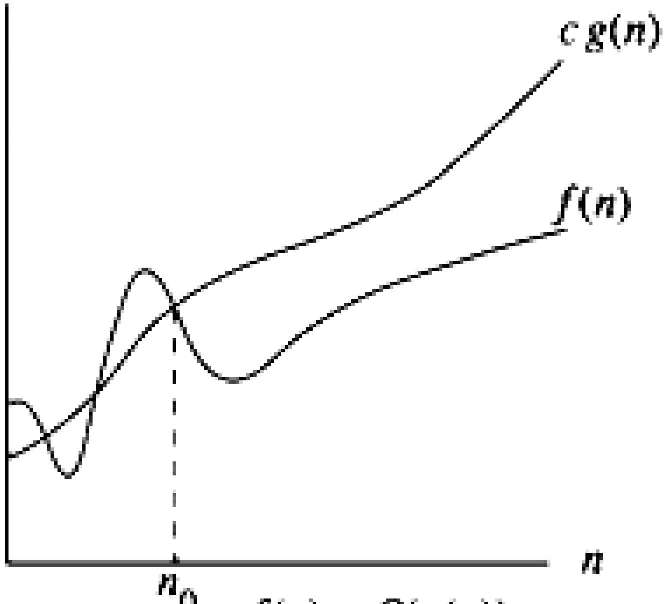


הגדרה1:

נאמר כי f(n)=O(g(n)) (הפונקציה g חוסמת אסימפטותית מלמעלה את הפונקציה f)
אם קיימים קבועים חיוביים n_0, c כך ש- $f(n) \le c \cdot g(n)$ לכל $n \ge n_0$
או :

$$f(n) = O(g(n))$$

$$\Leftrightarrow \exists c > 0, \lim_{n \leftarrow \infty} \frac{f(n)}{g(n)} \le c$$



$f(n) = O(g(n))$

הגדרה2:

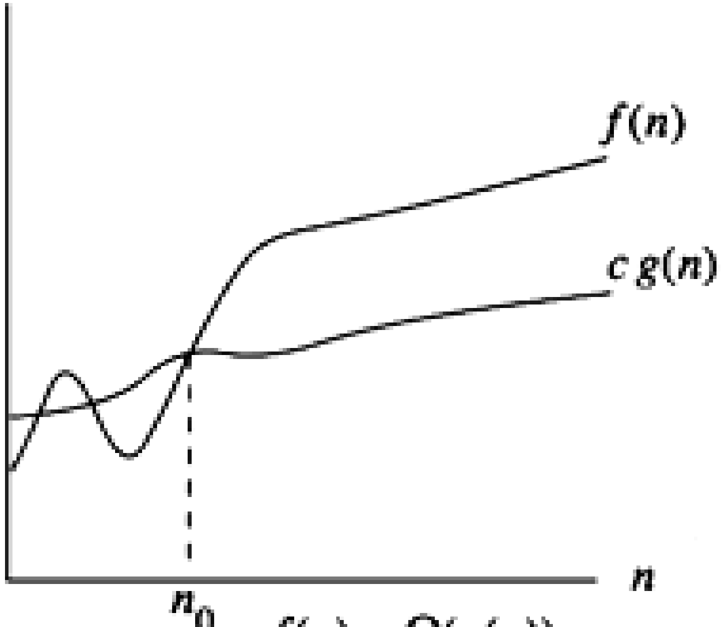
נאמר כי $f(n) = \Omega g(n)$ (הפונקציה g חוסמת אסימפטותית מלמטה את הפונקציה f)

אם קיימים קבועים חיוביים c, n_0 כך ש- $f(n) \geq c \cdot g(n)$ לכל $n \geq n_0$

או :

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

$$\Leftrightarrow \exists c > 0, \lim_{n \to \infty} \frac{g(n)}{f(n)} \leq c$$



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

if *g*(*n*) is Ω(*f*(*n*)) then *f*(*n*) is *O*(*g*(*n*))

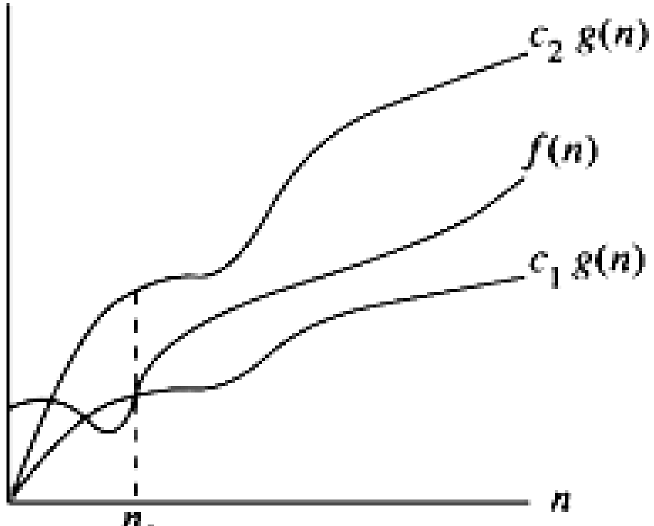
הגדרה3:

נאמר כי $f(n) = \Theta g(n)$ (הפונקציה g שווה אסימפטוטית לפונקציה f)

אם קיימים קבועים חיוביים n_0, c_1, c_2 כך ש- $c_2 \cdot g(n) \le f(n) \le c_1 \cdot g(n)$ לכל $n \ge n_0$

או :

$$f(n) = \Theta(g(n)) \Leftrightarrow \exists c > 0, \lim_{n \to \infty} \frac{f(n)}{g(n)} = c$$



$f(n) = \Theta(g(n))$

דוגמאות

$$f(n) = n, g(n) = n^2$$

$$\lim_{n\to\infty} = \frac{f(n)}{g(n)} = \frac{n}{n^2} = \frac{1}{n} = \frac{1}{\infty} = 0 \le c$$

$$\Rightarrow f(n) = O(g(n))$$

$$f(n) = n \cdot \log(n), g(n) = \log(n)$$

$$\lim_{n\to\infty} = \frac{f(n)}{g(n)} = \frac{n \cdot \log(n)}{\log(n)} = \frac{n}{1} = \frac{\infty}{1} = \infty$$

$$\Rightarrow f(n) = \Omega(g(n))$$

$$f(n) = 100n^2, g(n) = n^2$$

$$\lim_{n\to\infty} = \frac{f(n)}{g(n)} = \frac{100n^2}{n^2} = \frac{100}{1} = 100 = c$$

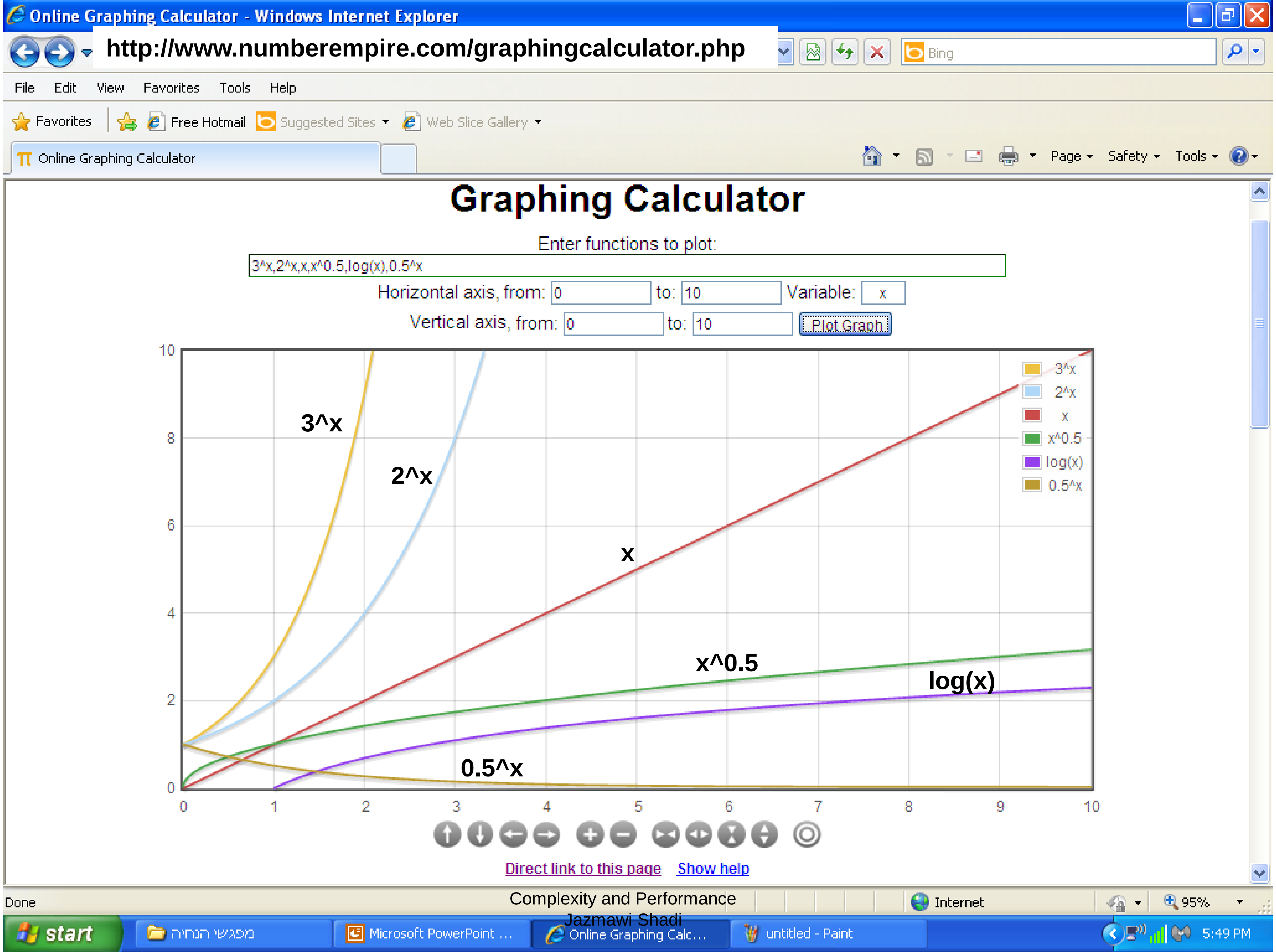
$$\Rightarrow f(n) = \Theta(g(n))$$

תרגיל:

לכל זוג פונקציות f , g קבע אם:

1. $f = O(g)$
2. $f = \Omega(g)$
3. $f = \Theta(g)$

f(n)	g(n)	
$(0.5)^n$	$n^{0.5}$	f(n) = Og(n)
$\log(n)$	$\log(2n)$	f(n) = Θg(n)
2^n	3^n	f(n) = Og(n)
$\log_2^{n^2}$	$\log_6^{n^6}$	f(n) = Θg(n)
n	log(n)	f(n) = Ωg(n)


Online Graphing Calculator - Windows Internet Explorer
http://www.numberempire.com/graphingcalculator.php
Graphing Calculator
Enter functions to plot:
3^x,2^x,x,x^0.5,log(x),0.5^x
Horizontal axis, from: 0 to: 10 Variable: x
Vertical axis, from: 0 to: 10
Plot Graph
3^x
2^x
x
x^0.5
log(x)
0.5^x
Direct link to this page
Show help

תרגיל:קבע מהי הפונקציה הדומיננטית בכל אחת מהפונקציות הנתונות

Expression	Dominant term(s)	$O(\dots)$
$5 + 0.001n^3 + 0.025n$	$0.001n^3$	$O(n^3)$
$500n + 100n^{1.5} + 50n\log_{10} n$	$100n^{1.5}$	$O(n^{1.5})$
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$	$2.5n^{1.75}$	$O(n^{1.75})$
$n^2 \log_2 n + n(\log_2 n)^2$	$n^2 \log_2 n$	$O(n^2 \log n)$
$n \log_3 n + n \log_2 n$	$n \log_3 n,\ n \log_2 n$	$O(n \log n)$
$3 \log_8 n + \log_2 \log_2 \log_2 n$	$3 \log_8 n$	$O(\log n)$
$100n + 0.01n^2$	$0.01n^2$	$O(n^2)$
$0.01n + 100n^2$	$100n^2$	$O(n^2)$
$2n + n^{0.5} + 0.5n^{1.25}$	$0.5n^{1.25}$	$O(n^{1.25})$
$0.01n \log_2 n + n(\log_2 n)^2$	$n(\log_2 n)^2$	$O(n(\log n)^2)$
$100n \log_3 n + n^3 + 100n$	n^3	$O(n^3)$

http://www.cs.auckland.ac.nz

תן תיאור לגבי זמן הריצה (Big-Oh notation) לכל אחד מהתוכניות הבאות.
(הערה זמן הריצה תלויי כאן במספר הפעולות אשר ++S מבצעת)

A $O(\sqrt{n})$

```
s = 0;
for(i=0;i<sqrt(n)/4;i++)
   s++;
for(j=0 ;j<sqrt(n)/2;j++)
   s++;
for(k=0;k<16+j;k++)
   s++;
```

B $O(\sqrt{n})$

```
s = 0;
for(i=0;i<sqrt(n)/4;i++)
    for(j=i;10+i;j++)
        for(k=j;k<16+j;k++)
              s++;
```

C $O(n^5)$

```
s = 0;
for(i=1;i<2*n;i++)
   for(j=1;j<i*i;j++)
      for(k=1;k<j;k++)
          s++;
```

עבור הפונקציה f ציינו מהי **סיבוכיות זמן הריצה ומהי סיבוכיות המקום** שלה

```java
public static void f (int n) {
        int i, j, k;
        for (i = 1; i < n; i *= 5)
            for (j = 0; j < n; j++)
                for (k = n; k > j; k--)
                    System.out.println("hello");
}
```

O(log(n))

O(n)

O(n)

O(1)

הסבר: עלות איטרציה אחת בלולאה j היא n-j. סך הכל עלות :

$$\sum_{j=0}^{n-1} n-j = \frac{(n+1)\cdot n}{2}$$

זוהי בעצם עלות אטרציה אחת של לולאת ה- i.

מספר האטרציות של i הוא log(n)

לכן הסיבוכיות היא : $\Theta(n^2 \log(n))$

סיבוכיות זמן ריצה: $O(n^2 \log(n))$

סיבוכיות מקום: $O(1)$

עבור הפונקציה f ציינו מהי **סיבוכיות זמן הריצה ומהי סיבוכיות המקום** שלה

```java
public void f (int n) {
        int i, j, size = 150;
        int[ ] arr = new int[size];

        for(i = 0; i < n; i++){
            for(j = 0; j < n; j += 2)
                System.out.println("hello");
        }
}
```

שתי לולאות מקוננות אשר כל אחת מבצעת n איטרציות.
לכן זמן הריצה : $\Theta(n^2)$

המערך הוא בגודל קבוע.
לכן סיבוכיות המקום: $\Theta(1)$

עבור הפונקציה f ציינו מהי **סיבוכיות זמן הריצה ומהי סיבוכיות המקום** שלה

```
public void f (int n, int k) {
      int[ ] arr;
      int count=0;
      for (int i = 0; i < n; i++)
          for (int j = k; j > 1; j /= 5){
             arr=new int[i*j];
             arr=null;
             count++;
      }
      for (int i = 0; i < k; i++)
          for (int j = k; j > 1; j--)
              count++;
  }
```

$\Theta(n)$

$\Theta(\log k)$

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

$\Theta(k)$

$\Theta(k)$

$\Theta(k^2)$

סיבוכיות זמן ריצה: $\Theta(k^2 + n \cdot \log k)$

סיבוכיות מקום: $\Theta(k \cdot n)$

כדאי לזכור

O(n) ➔
K>1 קבוע

```
i←1
while i<n
  i←i+1
```

```
i←1
while i<n
  i←i+k
```

```
i←n
while i>1
  i←i-k
```

O(logn) ➔
K>1 קבוע

```
i←1
while i<n
  i←i*2
```

```
i←1
while i<n
  i←i*k
```

```
i←n
while i>1
  i←i/k
```

Searching Algorithms

1. linear search
2. stateSearch

חיפוש איבר במערך לא ממוין

3. linear
4. quadratic
5. binary

חיפוש איבר במערך ממוין

Search Algorithm

```
public int search (int[ ] data, int num) {
        int pos = 0;
        while ( (data[pos] != num) && (pos <(data.length - 1)) )
            pos++;
        if (data[pos] == num)
            return pos;
        else
            return -1;
}
```

worst-case → n
best-case → 1
average-case → n/2

חיפוש ליניארי על מערך לא ממוין

עובר איבר אחרי איבר במערך
עוצר אם האיבר נמצא או אם הגיע לסוף למערך

עוצר במקום שהאיבר נמצא

StateSearch Algorithm

```
public int stateSearch (int[ ] data, int num) {
    final int FOUND = 0, ABSENT = 1, SEARCHING = 2;
    int pos = 0, state = SEARCHING;
    do {
        if (pos >= data.length)
            state = ABSENT;
        else if (data[pos] == num)
            state = FOUND;
        else pos++;
    } while ( state == SEARCHING) ;
    switch (state) {
       case FOUND: return pos;
       case ABSENT: return -1;
       default: return -1;
    }
}
```

worst-case → n
best-case → 1
average-case → n/2

עובר איבר אחרי איבר במערך

עוצר אם האיבר נמצא או אם הגיע לסוף למערך

עוצר במקום שהאיבר נמצא

Linear Search Algorithm

```
public static int linear (int[ ] data, int num) {
        int pos = 0;
        while ( (data[pos] < num) && (pos < (data.length - 1)) )
            pos++;
        if (data[pos] == num)
            return pos;
        else return -1;
}
```

worst-case → n
best-case → 1
average-case → n/2

חיפוש ליניארי על מערך ממוין

Quadratic Search Algorithm

הרעיון: לשפר את האלגוריתם של linear search ע"י דילוג (מהר יותר) על אותם אברים בעלי ערך קטן יותר מהערך של האיבר הרצוי. נבצע זאת ע"י שימוש בטכניקת הקפיצות

שאלה: מהו גודל כל קפיצה?

תשובה: כל צעד שווה ל- sqrt(n) כאשר n הוא גודל המערך

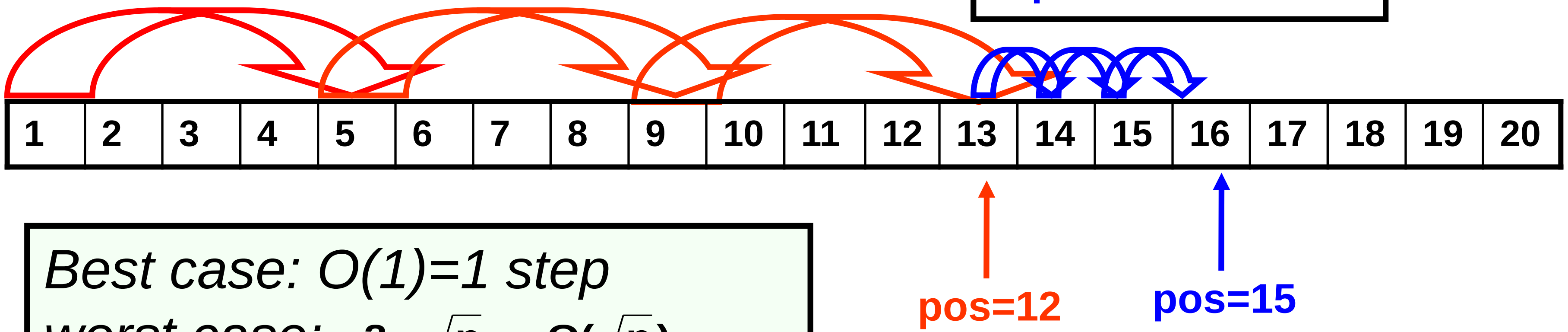
מחושב פעם אחת
בתחילת התוכנית

שלב 1

num=16
pos=0
jumpSize = sqrt(20)=4
----
data[pos + jumpSize] < num → pos=4
data[pos + jumpSize] < num → pos=8
data[pos + jumpSize] < num → pos=12
data[pos + jumpSize] < num → break

שלב 2

pos=12
----
data[pos] < num
data[pos] != num
→pos=13
data[pos] < num
data[pos] != num
→pos=14
data[pos] < num
data[pos] != num
→pos=15



Best case: O(1)=1 step
worst case: $2 \cdot \sqrt{n} = O(\sqrt{n})$
Average case : $\frac{1}{2}\sqrt{n}+\frac{1}{2}\sqrt{n}=O(\sqrt{n})$

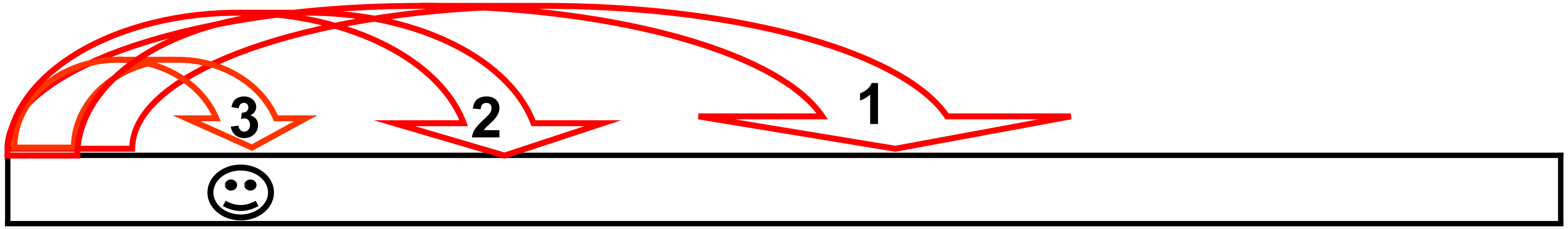
```
public int quadratic(int[ ] data, int num) {
        final int FOUND=0, ABSENT=1, SEARCHING=2, CLOSE_ENOUGH=3;
        int state = SEARCHING, pos = 0, jumpSize;
        jumpSize = (int) (Math.sqrt(data.length));
        do {
            if ( (pos + jumpSize) >= data.length)
                state = CLOSE_ENOUGH;
            else if (data[pos + jumpSize] > num )
                state = CLOSE_ENOUGH;
            else pos = pos + jumpSize;
        } while (state != CLOSE_ENOUGH);
        state = SEARCHING;
        do {
            if (pos >= data.length) state = ABSENT;
            else if ( data[pos] > num ) state = ABSENT;
            else if ( data[pos] == num ) state = FOUND;
            else pos++;
        } while (state == SEARCHING) ;
        if (state == ABSENT) return -1;
        else return pos;
}
```

שלב 1
מדלגים בעזרת קפיצות עד שערך האיבר הנוכחי גדול מערך האיבר הרצוי

שלב 2
רצים על האברים – החל מהמקום שהתקבל לאחר הקפיצות , ובודקים אם האבר קיים (מפעילים בשלב זה linear search)

***Binary Search Algorithm

הרעיון: לשפר את האלגוריתם של quadratic search ע"י חישוב דינאמי של אורך הקפיצות (במקום חישוב קבוע). הקפיצות בהתחלה הן גדולות אחר כך קטנות יותר. הקפיצה הראשונה בגודל חצי המערך , השנייה $\frac{1}{4}$, השלשית $\frac{1}{8}$ הרבעית $\frac{1}{16}$



Best case: O(1)=1 step
worst case: $\log_2^n$
Average case : $\log_2^n$

גרסה 1

```
public int binary  (int[ ] data, int num) {
        int mid, low = 0, high = data.length-1;
        while (low <= high){
            mid = (low+high)/2; // low + (high-low)/2
            if ( data[mid] == num )
            return mid;
            else if ( num < data[mid] )
                 high = mid-1;
            else low = mid + 1;
        }
        return -1;
    }
```

גרסה 2

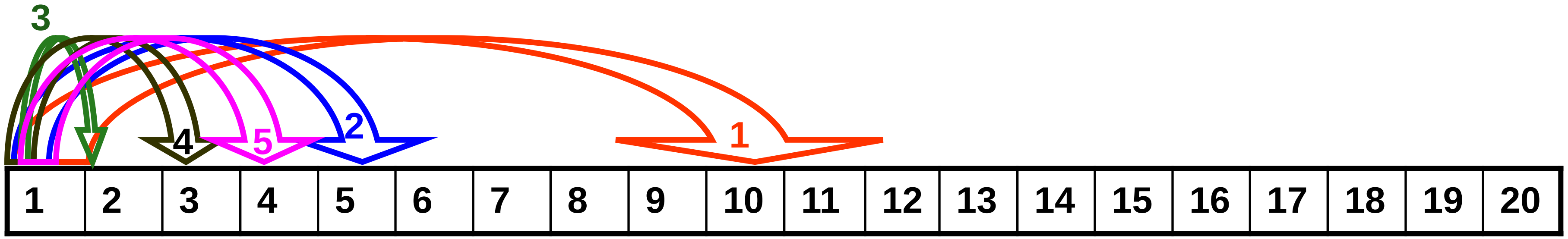
```java
public int binary (int[ ] data, int num) {
      int mid, lower = 0, high = (data.length - 1);
      do {
         mid = ((lower + high) / 2);
         if (num < data[mid])
            high = mid - 1;
         else lower = mid + 1;
      } while ( (data[mid] != num) && (lower <= high) );

      if (data[mid] == num)
         return mid;
      else return -1;
}
```

num=4
lower =0   high=19
----
lower =0   high=19 → mid=9
num < data[mid] → high = mid – 1=8
lower =0   high=8 → mid=4
num < data[mid] → high = mid – 1=3
lower =0   high=3 → mid=1
num > data[mid] → lower = mid + 1=2
lower =2   high=3 → mid=2
num > data[mid] → lower = mid + 1=3
lower =3   high=3 → mid=3
num == data[mid] → return mid

Termination

- When the lower and high bounds of the unknown area pass each other, the unknown area is empty and we terminate (unless we've already found the value)
- Goal: Locate a value, or decide it isn't there
- Intentional Bound: We've found the value
- Necessary Bound: The lower and high bounds of our search pass each other
- Plan: Pick a component midway between the high and lower bounds. Reset the lower or high bound, as appropriate.



Sorting Algorithms

Selection sort
Insertion sort
Bubble sort

Selection Sort

for every "first" component in the array find the smallest component in the array; exchange it with the "first" component

scan right starting with 3
1 is the smallest. swap(1,3)

3 9 6 1 2

scan right starting with 9
2 is the smallest. swap(9,2)

1 9 6 3 2

scan right starting with 6
3 is the smallest. swap(6,3)

1 2 6 3 9

scan right starting with 6
6 is the smallest. swap(6,6)

1 2 3 6 9

1 2 3 6 9

Best case: $O(n^2)$
worst case: $O(n^2)$
Average case : $O(n^2)$

The worst-case runtime complexity is O(n2).

Selection Sort Algorithm

המיון בסדר יורד

```
public static void select (int[ ] data) {
        int first, current, largest;
        for (first = 0; first < data.length - 1; first++) {
            largest = first;
            for (current = first + 1; current < data.length;current++) {
                if ( data[current] > data[largest] )
                    largest = current;
            }
            if (largest != first) {
                swap( data,largest , first );
            }
        }
}
public static void swap(int[ ] data,int largest,int first){
        int temp = data[largest];
        data[largest] = data[first]; // Make the swap
        data[first] = temp;
}
```

Insertion Sort

for ***every “newest” component remaining in the array***
- ✓ ***temporarily remove it;***
- ✓ ***find its proper place in the sorted part of the array;***
- ✓ ***slide largest values one component to the right;***
- ✓ ***insert the “newest” component into its new position;***

3 is sorted **Shift nothig. Insert 9**	**3**	**9**	**6**	**1**	**2**
3 and 9 are sorted **Shift 9 to the right. Insert 6**	**3**	**9 →**	**6**	**1**	**2**
3, 6 , 9 are sorted **shift 9,6,3 to the right. Insert 1**	**3 →**	**6 →**	**9 →**	**1**	**2**
1,3,6,9 are sorted **shift 9,6,3 to the right. Insert 2**	**1**	**3 →**	**6 →**	**9 →**	**2**
	1	**2**	**3**	**6**	**9**

Best case: **O(n)**
worst case: $\mathbf{O(n^2)}$
Average case : $\mathbf{O(n^2)}$

The worst-case runtime complexity is O(n2

Insertion Sort Algorithm

```
public static void insert (int[ ] data) {
      int newest, current, newItem;
      boolean seeking;
      for (newest = 1; newest < data.length; newest++) {
         seeking = true;
         current = newest;
         newItem = data[newest];
         while (seeking) {
            if (data[current - 1] < newItem) {
               data[current] = data[current -1];
               current--;
               seeking = (current > 0);
            }
            else seeking = false;
         }
         data[current] = newItem;
      }
   }
```

Insertion Sort Algorithm

- **Insertion sort** is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. However, insertion sort provides several advantages: Simple implementation
- Efficient for (quite) small data sets
- Adaptive (i.e., efficient) for data sets that are already substantially sorted: the time complexity is $O(n + d)$, where d is the number of inversions
- More efficient in practice than most other simple quadratic (i.e., $O(n2)$) algorithms such as selection sort or bubble sort; the best case (nearly sorted input) is $O(n)$
- Stable: i.e., does not change the relative order of elements with equal keys
- In-place: i.e.only requires a constant amountO(1)of additional memory space
- Online: i.e., can sort a list as it receives it
- When humans manually sort something(for example,a deck of playing cards), most use a method that is similar to insertion sort.

http://en.wikipedia.org/wiki/Insertion_sort

Bubble Sort

for every "last" component
for every component from the first to the "last"
compare that component to each remaining component;
exchange them if necessary

```
public static void bubble (int[ ] data) {
        int last, current, temp;
        for (last = data.length-1; last > 0; last--) {
            for (current = 0; current < last; current++) {
                if ( data[current] > data[current + 1] ) {
                    temp = data[current];
                    data[current] = data[current + 1];
                    data[current + 1] = temp;
                }
            }
        }
}
```

Best case: **$O(n^2)$**
worst case: **$O(n^2)$**
Average case : **$O(n^2)$**

לאחר אטירציה אחת של הלולאה החיצונית

7, 5, 2, 4, 3, 9
5, 7, 2, 4, 3, 9
5, 2, 7, 4, 3, 9
5, 2, 4, 7, 3, 9
5, 2, 4, 3, 7, 9
5, 2, 4, 3, 7, 9

The worst-case runtime complexity is O(n2)

https://youtu.be/yIQuKSwPlro

QuickSort

- מיון מהיר (באנגלית: Quicksort) הוא אלגוריתם מיון השוואתי אקראי מהיר במיוחד. סיבוכיות הזמן הממוצעת של האלגוריתם היא $\Theta(n\log(n))$ פעולות (כמו, למשל, מיון מיזוג), אך במקרה הגרוע עלול האלגוריתם לדרוש $O(n^2)$ פעולות (כמו, למשל, מיון בועות). בפועל, אלגוריתם מיון מהיר נחשב לאלגוריתם המיון ההשוואתי היעיל ביותר הידוע זאת מאחר שהסיכוי למקרה הגרוע הוא מאוד נמוך.
- אלגוריתם מיון מהיר הוא אלגוריתם רקורסיבי הפועל בשיטת הפרד ומשול. צעדיו הם כדלקמן:
  - ✓ בהינתן סדרת איברים, בחר איבר מהסדרה באקראי (נקרא: pivot, או "איבר ציר").
  - ✓ סדר את כל האיברים כך שהאיברים הגדולים מאיבר הציר יופיעו אחרי האיברים הקטנים מאיבר הציר.
  - ✓ באופן רקורסיבי, הפעל את האלגוריתם על סדרת האיברים הגדולים יותר ועל סדרת האיברים הקטנים יותר.
  - ✓ תנאי העצירה של האלגוריתם הוא כאשר ישנו איבר אחד, ואז האלגוריתם מודיע כי הסדרה ממוינת
- יעילות האלגוריתם תלויה בבחירת איבר הציר. אם - כתוצאה מ"מזל טוב" - איבר הציר הוא תמיד האיבר האמצעי בגודלו בסדרה, האלגוריתם לוקח $\Theta(n\log(n))$ אם, מאידך - כתוצאה מ"מזל רע" - איבר הציר הוא האיבר הקטן ביותר או האיבר הגדול ביותר, אזי האלגוריתם לוקח $O(n^2)$. לכן, ישנה חשיבות רבה למציאת איבר הציר: לעתים משתמשים אלגוריתמים בפועל בשיטת "חציון משלושה" על מנת לבחור איבר שעשוי להיות "מתאים יותר" מאשר איבר שרירותי.
- באופן תאורטי, ניתן למצוא את האיבר האמצעי בסדרה בזמן לינארי, מה שמבטיח זמן ריצה של $n\log n$, אולם אין שימוש רב בשיטה זו בפועל בשל זמן הריצה הגדול בפועל של תהליך זה.
- אחת התכונות של האלגוריתם היא שזמן הריצה שלו עבור קלטים קטנים במיוחד גדול ביחס לאלגוריתמים פשוטים, כגון מיון בחירה (selection sort). לכן, ביישומים רבים תנאי העצירה של האלגוריתם הוא עבור מספר קבוע כלשהו של איברים (7, למשל), ומשלב זה ואילך מתבצע המיון באמצעות אלגוריתם "מיון בחירה" או אלגוריתם דומה.

http://en.wikipedia.org/wiki/Quicksort

סיכום סיבוכיות

Performance Comparison

▪**Quicksort:**

✓ **Best case: O(n*log*2n)**

✓ **Worst case: O(n2) – when the pivot is always the second-largest or second-smallest element (since medianLocation won't let us choose the smallest or largest)**

✓ **Average case over all possible arrangements of n array elements: O(n*log*2n)**

✓ **Worst case space complexity: O(n) (naive)**
O(log n) In-place

▪**Selection Sort and Insertion Sort**

✓ **Average case: O(n2)**

לצפיה בסרטון אנימציה :

http://www.youtube.com/watch?v=o2dm4X-t8L0

http://www.youtube.com/watch?v=aQiWF4E8flQ

http://www.youtube.com/watch?v=J87baBOZ3Nk

http://www.cs.auckland.ac.nz/~jmor159/PLDS210/qsort.html

תרגיל 1

נתון מערך ממוין Arr המכיל n מספרים. בהינתן מספר k
כלשהו, קבע האם קיימים במערך זוג מספרים
בעלי אינדקסים שונים כך שסכומם הוא k

דוגמה:

Arr

1	2	3	4	5	6

(5+6) true ← K=11
(1+5) true ← K=6
false ← K=12

פתרון נאיבי

נעבור בעזרת שתי לולאות על כל זוגות המספרים השונים במערך.

עבור כל זוג של מספרים נבדוק אם סכומו הוא k.

```
public boolean checkSum (int[ ] arr, int k) {
        for (int i = 0 ; i < arr.length ; i++)
            for (int j = 0 ; j < arr.length ; j++)
                if ( (i != j) && (arr[ i ] + arr[ j ]) == k)
                    return true;
      return false;
}
```

$$\underbrace{n+n+\ldots+n}_{\text{n פעמים}} = \Theta(n^2)$$

סיבוכיות זמן ריצה: $\Theta(n^2)$

שיפור משמעותי

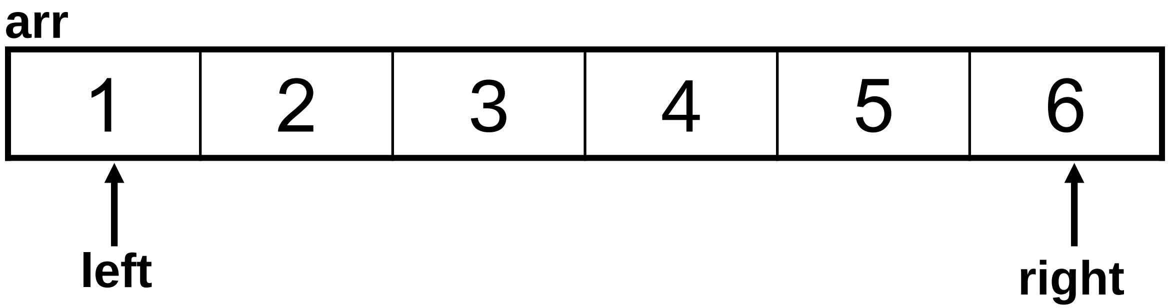
עד כה לא ניצלנו בצורה משמעותית את העובדה שהמערך ממוין. עבור כל מספר arr[i] במערך נבדוק ע"י חיפוש בינארי אם המספר בעל הערך k -arr[i] גם הוא נמצא במערך.

```
public boolean checkSum (int[ ] arr,int k) {
        int pos;
        for (int i = 0; i < arr.length; i++) {
          pos = binary( arr, k - arr[i] );
          if(pos!=-1 && pos!=i)
            return true;
        }
        return false;
}
```

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

שיפור משמעותי מאד

נבצע מעבר יחיד על איברי המערך ע"י שימוש בשני אינדקס left ו- right.



- **אם left=right אז מחזירים false.**
- **אם מתקיים arr[left] + arr[right] = k אז מחזירים true**
- **כל עוד מתקיים arr[left] + arr[right] < k אז ++left**
- **כל עוד מתקיים arr[left] + arr[right] > k אין טעם לקדם את left, שכן בגלל שהמערך ממוין גם הפעם יתקיים arr[left] + arr[right] > k . לכן נקטין את הערך של right .**

האלגוריתם המשופר

```
public boolean checkSum (int[ ] arr, int k)  {
        int i=0 , j=arr.length-1;
        while( i != j ) {
            if ( arr[i] + arr[j] == k )
                return true;
            if ( arr[i] + arr[j] < k )
                i++;
            if ( arr[i] + arr[j] > k)
                j--;
        }
        return false;
}
```

סיבוכיות :

$\Theta(n)$

תרגיל 2

נתון מערך arr המכיל **n** מספרים. כמו כן נתון שהמספרים נעים בין [0-100]. כתוב אלגוריתם (יעיל ככול האפשר) אשר בודק עבור כל מספר במערך את מספר ההופעות שלו.

פתרון נאיבי: נעבור בעזרת שתי לולאות על כל המספרים השונים במערך. עבור כל מספר נבדוק כמה פעמים הוא מופיע. $O(n^2)$

```
public static void count (int[ ] arr) {
        for ( int i=0 ; i<arr.length ; i++) {
           int s=0;
          for ( int j=0 ; j<arr.length ; j++)
             if( arr[i] == arr[j] )
                s++;
          System.out.println(arr[ i ]+ ":" +s);
        }
}
```

```
Output:
{5,2,8,2,2,1,1,1}
5:1
2:3
8:1
2:3
2:3
1:3
1:3
1:3
```

שיפור משמעותי

```java
public static void count (int[ ] arr){
	quicksort (arr);
	int s=1 , i;
	for( i=1 ; i<arr.length ; i++ ) {
	    if ( arr[i] != arr[i-1] ) {
	       System.out.println(arr[i-1] + ":" + s);
	      s=0;
	    }
	    s++;
	}
	System.out.println(arr[i-1] + ":" + s);
}
```

נמיין קודם את המערך

```
Input:
{5,2,8,2,2,1,1,1});
Output:
1:3
2:3
5:1
8:1
```

סיבוכיות: $O(n \cdot \log n)$

סיבוכיות מקום: $O(\log n)$

שיפור משמעותי מאד

סיבוכיות מקום:
O(1)

נצל את העובדה שטווח המספרים הוא קבוע 100-0

```
public static void count(int[ ]arr){
        final int N=100;
        int[ ] temp=new int[N];
        for (int i=0 ; i<arr.length ; i++)
            temp [ arr[ i ] ] += 1;

        for ( int i=0 ;i<N ;i++)
            if ( temp[ i ] != 0)
                System.out.println( i + ":" + temp[ i ]);
}
```

O(n)

O(1)

```
Output:
{{5,2,8,2,2,1,1,1});
1:3
2:3
5:1
8:1
```

נגדיר מערך temp בגודל 100. כל כניסה במערך מייצגת מספר מהמערך arr כלומר האינדקס של המערך temp הוא מספר מהמערך arr. נסרוק את המערך arr ובעלות של o(1) נקדם את התוכן של המערך temp במקום temp [arr[i]] בהתאם.

סיבוכיות זמן: O(n)

סיבוכיות מקום: O(1)

תרגיל 3

כתבו שיטה סטטית אשר מקבלת שני מערכים באותו גודל double a[ ] ו double b[ ] ,
מספר נוסף x ואת גודל המערכים n :

public static int find(double a[ ] , double b[ ] , int n , double x)

נתון כי :

כל אברי מערך a[ ] שונים זה מזה וכל אברי מערך b[ ] שונים זה מזה.

כמו כן נתון כי המערך a[ ] ממוין בסדר יורד ואילו המערך b[ ] ממוין בסדר עולה

הפונקציה תחזיר אינדקס i כך שמתקיים :

$$2 \cdot a[i] - 3 \cdot b[i] = x \,.$$

אם לא קיים אינדקס כזה , הפונקציה תחזיר 1-

השיטה שתכתבו צריכה להיות יעילה ככל הניתן.

- הביטוי 2a[i] - 3b[i] מגדיר סדרה ממוינת של מספרים:
- נתון כי a ממוין בסדר יורד, לכן כאשר i גדל, a[i] קטן ומקטין את הביטוי.
- כמו כן נתון כי b ממוין בסדר עולה, לכן כאשר i גדל, b[i] גדל ושוב מקטין את הביטוי (בגלל מינוס)
- לכן הסדרה 2a[i] - 3b[i] ממוינת בסדר יורד:
- לכן נשתמש בחיפוש בינארי על 2a[i] – 3b[i]

```
public static int find(double a[ ], double b[ ], int n, double x) {
        int left= 0, right= n-1, mid;
        double exp;
        while (left <= right) {
            mid = (left + right) / 2;
            exp = 2*a[mid] - 3*b[mid];
            if(exp == x) {
                return mid;
            }
            if (exp > x) {
                left = mid + 1;
            }
            else {
                right = mid - 1;
            }
        }
        return -1;
}
```

זהו חיפוש בינארי עם שינוי קטן: במקום לחפש ערך של מערך מחפשים את exp ששווה ל x

האם אפשר היה לבנות מערך c[i]=2*a[i]- 3*b[i] ולהשתמש בחיפוש בינארי בלי שינויים?

סיבוכיות זמן : O(logn)

שאלה 1 – 25 נקודות (להגשה)

א. כתבו שיטה `public static boolean single(int[] values)` המקבלת מערך `values` מלא במספרים, ומחזירה true אם קיים במערך איבר המופיע פעם אחת בלבד במערך, ו-false אחרת.

למשל, עבור המערך {3, 1, 4, 1, 4, 1} השיטה תחזיר true כיוון ש-3 מופיע פעם אחת בלבד, ואילו עבור המערך {3, 1, 4, 1, 4, 3} השיטה תחזיר false כיוון שאין איבר שמופיע רק פעם אחת.

השיטה שתכתבו צריכה להיות יעילה ככל הניתן. תשובה שאינה יעילה מספיק, כלומר שתהיה בסיבוכיות גדולה יותר מזו הנדרשת לפתרון הבעיה לא תקבל את מלוא הנקודות.

ב. מה סיבוכיות הזמן ומה סיבוכיות המקום של השיטה שכתבתם? הסבירו תשובתכם.

ג. כיצד תשתנה תשובתכם אם השיטה מקבלת פרמטר נוסף שהוא משתנה חיובי שלם k, והיא צריכה להחזיר true אם יש איבר שמופיע במערך k פעמים או יותר. כתבו שיטה זו, גם היא **יעילה ככל הניתן**. חתימת השיטה תהיה:

```
public static boolean kTimes(int[] values, int k)
```

ד. מה סיבוכיות הזמן ומה סיבוכיות המקום של השיטה שכתבתם לסעיף ג? הסבירו תשובתכם.

```java
public static boolean single (int [ ] values) {
        if (values.length == 0) return false;
        if (values.length == 1) return true;

        QuickSort.quickSort(values);

        boolean firstTime = true;

        for (int i = 1; i < values.length; i++) {
            if (values[i] == values[i - 1])
                firstTime = false;
            else if (values[i] != values[i - 1] && firstTime)
                return true;
            else
                firstTime = true;
        }

        return firstTime;
}
```

O(nlogn)

O(n)

O(nlogn) + O(n) ==> total of O(nlogn))

```java
public static boolean kTimes(int[ ] values, int k) {
    // if array i sempty or the size of it is less then k
    if (values.length == 0 || values.length < k) return false;
    if (values.length > 0 && k == 1) return true;

    QuickSort.quickSort(values);

    int kCounter = 1;

    for (int i = 1; i < values.length; i++) {
        if (values[i] == values[i - 1]) {
            kCounter++;
            if (kCounter >= k) return true;
        }
        else if (values[i] != values[i - 1])
            kCounter = 1;
    }

    return false;
}
```

O(nlogn) + O(n) ==> total of O(nlogn))

שאלה 1 – 25 נקודות (להגשה)

נתאר את בעיית מציאת "בור" במערך דו-ממדי ריבועי:

קלט: מערך דו-ממדי ריבועי בגודל $n \times n$ המלא באפסים ואחדים בלבד.

נגדיר ש- k הוא **בור** (sink) אם בשורה ה- k - ית כל הערכים הם 0, ובעמודה ה- k -ית כל הערכים הם 1 (חוץ מהאיבר [k][k] עצמו שהוא 0).

פלט: האם קיים מספר k המהווה בור במערך? **אם כן, יש להחזיר את ערכו אחרת יש להחזיר -1**.

.

לדוגמא: במערך A 3 הוא "בור", ובמערך B אין בור.

A

0	1	0	1	1	0
1	0	1	1	0	0
0	0	0	1	0	1
0	0	0	0	0	0
1	0	1	1	0	0
0	1	0	1	1	1

B

0	1	0	0	0	1
1	0	0	1	1	1
0	0	0	0	0	0
1	1	1	1	1	1
0	1	0	1	0	1
1	0	0	0	1	0

כתבו שיטה **יעילה** הפותרת את הבעיה. השיטה תחזיר את המספר k המהווה בור במערך, אם קיים אחד כזה, ו- -1 אם לא קיים בור במערך. כתבו והסבירו מה סיבוכיות השיטה שכתבתם.

חתימת השיטה תהיה:

```
public static int isSink (int [][] mat)
```

שימו לב, השיטה צריכה להיות יעילה ככל הניתן. שיטה שתעבוד בסיבוכיות גבוהה מזו הנדרשת (במקום או בזמן) לא תקבל את מירב הנקודות. פתרון נכון שיהיה בסיבוכיות $O(n2)$ יזכה את כותבו ב - 10 נקודות בלבד.

```java
public static int isSink (int [][] mat)
{
    int col,row=0;
    for(col=0; col<mat.length;col++)
    {
        while(row<mat.length && mat[row][col] ==1)
            row++;
    }
    if(col==mat.length)
    {
        // if the possible sink is in the "last" column - we set row to be <mat.length
        if(row==mat.length)
            row--;

        int rowSum=0, colSum=0;
        // go over the k row - the row we reached the "end" of the matrix
        for(col=0;col<mat.length;col++)
            rowSum += mat[row][col];

        if(rowSum==0) // the k row is all "0" zero
        {
            for(col=0;col<mat.length;col++) // go over the k column
                rowSum += mat[col][row];
            if(rowSum == mat.length-1) // the k colum is all "1" ones
                return row; // found a sink
        }
    }
    return -1;
}
```

run-time complexity: O(n)
memory complexity: O(1)

שאלה 1 - 25 נקודות

כתבו שיטה סטטית המקבלת כפרמטרים מערך **ממוין** a של מספרים שלמים, ומספר שלם נוסף x.
השיטה צריכה להחזיר את מספר המופעים של המספר x במערך a.

חתימת השיטה היא:

```
public static int count (int []a, int x)
```

לדוגמא, עבור המערך

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
-5	-5	1	1	1	1	1	1	1	1	2	2	2	2	2	3	3	3	67	67	99

והמספר x = -5 השיטה תחזיר את הערך 2.

עבור אותו מערך והמספר x = 2 השיטה תחזיר את הערך 5.

עבור אותו מערך והמספר x = 8 השיטה תחזיר את הערך 0.

שימו לב:

- **השיטה שתכתבו צריכה להיות יעילה ככל הניתן. תשובה שאינה יעילה מספיק כלומר, שתהיה בסיבוכיות גדולה יותר מזו הנדרשת לפתרון הבעיה (במקום או בזמן) תקבל מעט נקודות בלבד.**
- **אל תשכחו לתעד את השיטה שכתבתם.**
- **כתבו מה סיבוכיות הזמן וסיבוכיות המקום של השיטה שכתבתם.**
- **אפשר להניח שהמערך אינו null ואינו ריק.**

```
public static int count (int [ ]a , int x)  {
        int left=0 , right=a.length – 1 , mid;
        int lower=0 , upper=a.length - 1;

        // check bound from the left
        while(left<=right)
        {
            mid=(left+right)/2;
            if ( x <= a[mid] )
                right=mid-1;
            else
                left=mid+1;
        }

        // check bound from the right
        while(lower<=upper)
        {
            mid=(lower+upper)/2;
            if ( x < a[mid] )
                upper=mid-1;
            else
                lower=mid+1;
        }

        return lower-left;
}
```

Modification of Binary search will solve the problem

שאלה 2 - 25 נקודות

נתונה השיטה הבאה:

```
public static int f(int[] a, int[] b, int[] c)
{
    final int N = a.length;
    int j, k=0, g = 0, t = 0;
    for(int i=0; i<N; i++)
    {
        for(j=0; j<N; j++)
            if(b[j] == a[i])
                break;
        if(j == N)
        {
            c[t] = a[i];
            if(g == 0 || c[t] > k)
            {
                k = c[t];
                g = 1;
            }
            t++;
        }
    }
    return k;
}
```

א. בהינתן שהמערכים a ו-b שניהם מגודל N ומלאים במספרים שלמים חיוביים, והמערך c הוא מגודל N והוא ריק, מה מבצעת השיטה f? הסבירו בקצרה מה מבצעת השיטה, ולא כיצד היא מבצעת זאת.
מעתיק את כל האיברים שנמצאים במערך a ולא נמצאים במערך b למערך c ומחזיר את הערך המקסימאלי שמופיע במערך a שלא מופיע במערך b

ב. מה סיבוכיות זמן הריצה של השיטה, ומהי סיבוכיות המקום של השיטה?
Runtime : O(n^2)
Space Complexity O(1)

ג. כתבו את השיטה f כך שתבצע את מה שביצעה בסעיף א' בסיבוכיות זמן ריצה קטנה יותר. שימו לב שהתוכן של מערך C הוא מה שחשוב אבל סדר האיברים במערך C אינו חשוב בכלל!

השיטה שתכתבו צריכה להיות יעילה ככל הניתן. תשובה שאינה יעילה מספיק כלומר, שתהיה בסיבוכיות גדולה יותר מזו הנדרשת לפתרון הבעיה (במקום או בזמן) תקבל מעט נקודות בלבד.

ד. מה סיבוכיות זמן הריצה ומהי סיבוכיות המקום של השיטה שכתבתם?
Runtime : O(n log n)
Space Complexity : O (log n)

אל תשכחו לתעד את מה שכתבתם!
אתם יכולים להוסיף שיטות פרטיות כרצונכם, אבל אל תשכחו להוסיף את הסיבוכיות שלהן לחישוב הסיבוכיות של השיטה שאתם כותבים.

ג' גרסה 1

```
public static int f(int [] a, int [] b, int [] c){
        quickSort(a);
        quickSort(b);

        int i, j;
        int k=0;
        for(i=0, j=0; i < a.length && j < b.length;){
            if (a[i] < b[j])
                c[k++] = a[i++];
            else if (a[i] == b[j]){
                i++;
                j++;
            }
            else
                j++;
        }

        for(; i<a.length; i++)
            c[k++] = a[i++];
        if (k==0)
            return 0;
        return c[k-1];
}
```

```
public static int f2 ( int[ ] a , int[ ] b , int[ ] c ) {

        int k=0,t=0;
        quickSort(b); // O(n log n)

        for (int i=0;i<a.length;i++)  { // O(n)
            if ( binarySearch(b,a[i]) == -1 )  {// O(log n)
               c[t]=a[i];

               if (a[i]>k)
                   k=a[i];

                t++;
            }
        }
        return k;
}
```

שאלה 1 – 25 נקודות (להגשה)

נתון כלי לצבירת מי גשם. חתך הכלי מתואר על ידי מערך הגבהים heights.

למשל, עבור מערך הגבהים {2, 1, 1, 4, 1, 1, 2, 3}

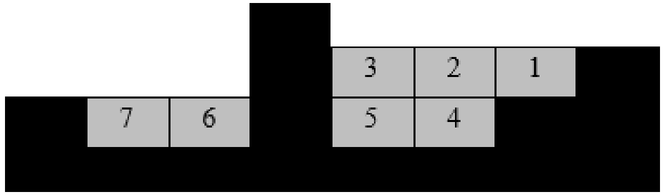
צורת הכלי היא:

כתבו שיטה סטטית המקבלת כפרמטר מערך גבהים ומחזירה את כמות הגשם שניתן לצבור בכלי עם מערך גבהים נתון.

ניתן להניח שכל הערכים במערך (הגבהים) הם מספרים שלמים חיוביים ממש.

למשל, עבור הכלי לעיל, ניתן לצבור 7 יחידות גשם (האיזורים האפורים שבאיור להלן).

רמז: יש לחשב בכל מקום בכלי מה גובה העמודה המקסימלית / המינימלית מימינו ומשמאלו.



חתימת השיטה היא:

```
public static int waterVolume (int [] heights)
```

שימו לב:

השיטה שתכתבו צריכה להיות יעילה ככל הניתן, גם מבחינת סיבוכיות הזמן וגם מבחינת סיבוכיות המקום. תשובה שאינה יעילה מספיק כלומר, שתהיה בסיבוכיות גדולה יותר מזו הנדרשת לפתרון הבעיה תקבל מעט נקודות בלבד.

ניתן להשתמש בשיטות עזר ככל הנדרש. **בחישוב הסיבוכיות צריך לחשב גם את הזמן והמקום של שיטות העזר.**

אל תשכחו לתעד את השיטה שכתבתם.

כתבו מה סיבוכיות הזמן וסיבוכיות המקום של השיטה שכתבתם.

```java
//O(n)+O(n)
 public static int waterVolume2(int[] heights) {
    int total=0;
    int[] leftMax=leftMax(heights);
    int[] rightMax=rightMax(heights);
    for(int i=0;i<heights.length;i++)
       total=total+Math.min(leftMax[i],rightMax[i])-heights[i];
    return total;
 }

 private static int[] leftMax(int[] a)// for each i value of max before I
{
    int [] res=new int[a.length];
    res[0]=a[0];
    for(int i=1;i<a.length;i++)
       res[i]=Math.max(res[i-1],a[i]);
    return res;
 }

 private static int[] rightMax(int[] a)// for each i value of max after i
 {
    int [] res=new int[a.length];
    res[a.length-1]=a[a.length-1];
    for(int i=a.length-2;i>=0;i--)
       res[i]=Math.max(res[i+1],a[i]);
    return res;
 }
```

```
/////// O(n)+O(1)
 public static int waterVolume(int[] heights) {
    int left=0;
    int right=heights.length-1;
    int total=0;
    int leftMax=heights[left];
    int rightMax=heights[right];

    while(left<=right)  {
       if(leftMax<rightMax){
          if(heights[left]>leftMax)
             leftMax=heights[left];
          else  {
             total+=leftMax-heights[left];
             left++;
          }
       }
       else {
          if(heights[right]>rightMax)
             rightMax=heights[right];
          else {
             total+=rightMax-heights[right];
             right--;
          }
       }
    }
    return total;
 }
```

END