

מבני נתונים

תרגול 2 – המשך סיבוכיות, חסם תחתון ומיון מנייה

היום

- חזרה על זמני ריצה ואסימפטוטיקה
- חסם תחתון למיוני השוואה
- מיון מהיר Quick Sort
- מיון מנייה Counting Sort
- עבודה עצמית

תרגול שעבר

○ זמני ריצה, אסימפטוטיקה וסימונים Ω, Θ, O

```
for(int i=0; i<n; i++) {  
    // Basic step 1  
    // Basic step 2  
}
```

$O(n)$

```
for(int i=0; i<50; i++) {  
    // Basic step 1  
    // Basic step 2  
}
```

***O*(1)**

```
for(int i=0; i<n; i++) {  
    for(int j=1; j<=m; j++) {  
        j++;  
    }  
}
```

$$T(n) = n \cdot \frac{m}{2} = O(n \cdot m)$$

```
for(int i=1; i<=n; i++) {  
    for(int j=i; j<=n; j++) {  
        // Basic step 1  
    }  
}
```

$$T(n) = n + (n - 1) + \dots + 1 = \frac{n(n + 1)}{2} = O(n^2)$$

```
for(int i=1; i<n; i*=2) {  
    // Basic step 1  
}
```

בכל איטרציה מבצעים פעולה אחת, כמה איטרציות יש?
נעקוב אחרי i , הערכים שהוא מקבל הינם: $1, 2, 4, 8, \dots, n$
נניח כי $n = 2^k$ כאשר k זה כמות האיטרציות, ולכן $k = \log_2(n)$
מכיוון שבכל איטרציה מבצעים פעולה אחת, נקבל כי הסיבוכיות היא
 $O(\log_2(n))$


```
for(int i=1; i<n; i*=8) {  
    // Basic step 1  
}
```

בכל איטרציה מבצעים פעולה אחת, כמה איטרציות יש?
נעקוב אחרי i , הערכים שהוא מקבל הינם: $1, 8, 64, \dots, n$
נניח כי $n = 8^k$ כאשר k זה כמות האיטרציות, ולכן $k = \log_8(n)$
מכיוון שבכל איטרציה מבצעים פעולה אחת, נקבל כי הסיבוכיות היא
 $O(\log_8(n))$

```
for(int i=1; i<=n; i*=2) {  
    for(int j=1; j<=i; j++) {  
        // Basic step 1  
    }  
}
```

$$T(n) = 1 + 2 + 4 + 8 + \dots = \frac{2^{\log_2 n} - 1}{2 - 1} = O(n)$$

```
for(int i=1; i<=n; i++) {  
    for(int j=i; j<=m; j++) {  
        // Basic step 1  
    }  
}
```

$$T(n) = m + (m - 1) + \dots + (m - n) = n \cdot m - (1 + 2 + \dots + n) = n \cdot m - \frac{n(n + 1)}{2} = O(n \cdot m - n^2)$$

```
void foo(int n) {  
    if(n==1) return;  
    for(int i=0; i<n; i++)  
        foo(n-1);  
}
```

נסמן ב- $T(n)$ את מספר הפעולות ש- $foo(int)$ מבצעת על הקלט n , מכאן $T(1) = 1$
בנוסף $T(n) = n \cdot T(n - 1)$
מכאן נקבל כי:

$$T(n) = n \cdot T(n - 1) = n \cdot (n - 1) \dots 2 \cdot T(1) = n! = O(n!)$$

```
int foo(int n) {
    if(n == 1 || n == 2) return 1;
    return foo(n-2) + foo(n-1);
}
```

נסמן ב- $T(n)$ את מספר הפעולות ש- $foo(int)$ מבצעת על הקלט n , מכאן $T(2) = T(1) = 1$
 בנוסף $T(n) = T(n+1) + T(n-2)$
 לבסוף נקבל כי:

$$T(n) = c_1 \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n = O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$$

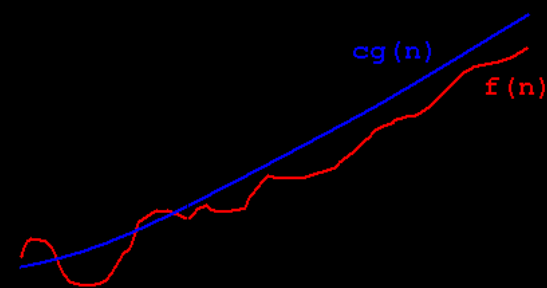
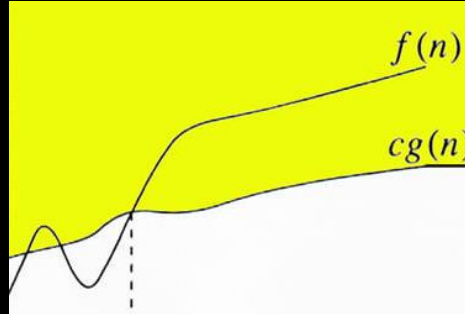
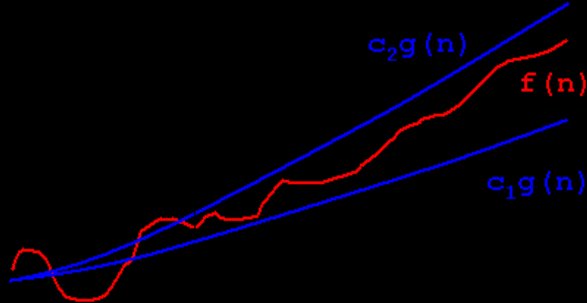
(פתרון נלמד בקורס בדידה)

```
void foo(int n) {  
    int x=2;  
    while(x<n) {  
        x=x*x*x;  
    }  
}
```

נסמן ב- k את מספר האיטרציות של הלולאה, הלולאה תסתיים כאשר $2^{3^k} > n$ ולכן:
 $k > \log_3 \log_2 n$ ולכן הסיבוכיות היא $O(\log \log n)$

```
for(int i=2; i<n; i=i*i) {  
    // Basic step  
}
```

נסמן ב- k את מספר האיטרציות של הלולאה, הלולאה תסתיים כאשר $2^{2^k} > n$ ולכן:
 $k > \log_2 \log_2 n$ ולכן הסיבוכיות היא $O(\log \log n)$



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

חסם הדוק

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

חסם תחתון

$$f(n) \in O(g(n))$$

חסם עליון

$$\begin{aligned} &\exists c_1 > 0 \\ &\exists c_2 > 0, n_0 \geq 0 : \forall n > n_0 \end{aligned}$$

$$c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n) \geq 0$$

$$\exists c > 0, n_0 \geq 0 : \forall n > n_0$$

$$f(n) \geq c \cdot g(n) \geq 0$$

$$\exists c > 0, n_0 \geq 0 : \forall n > n_0$$

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

$$0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

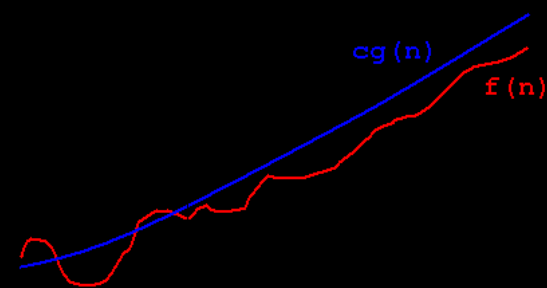
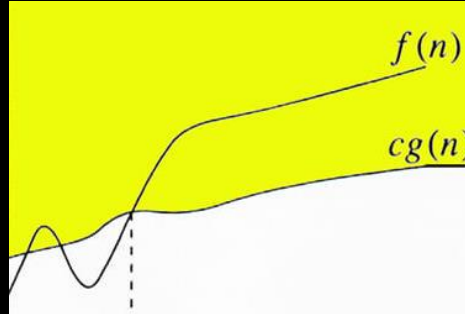
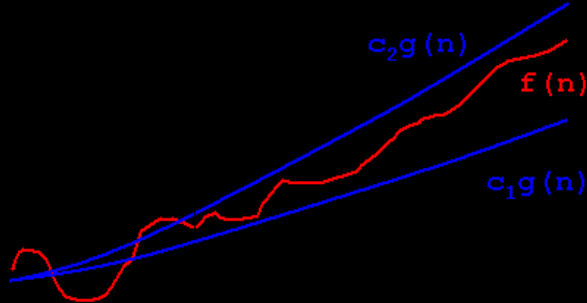
$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c : 0 < c < \infty$$

$$0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq \infty$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$$

$$0 \leq \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$



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

חסם הדוק

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

חסם תחתון

$$f(n) \in O(g(n))$$

חסם עליון

$$\begin{aligned} &\exists c_1 > 0 \\ &\exists c_2 > 0, n_0 \geq 0 : \forall n > n_0 \end{aligned}$$

$$c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n) \geq 0$$

$$\exists c > 0, n_0 \geq 0 : \forall n > n_0$$

$$f(n) \geq c \cdot g(n) \geq 0$$

$$\exists c > 0, n_0 \geq 0 : \forall n > n_0$$

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

$$0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

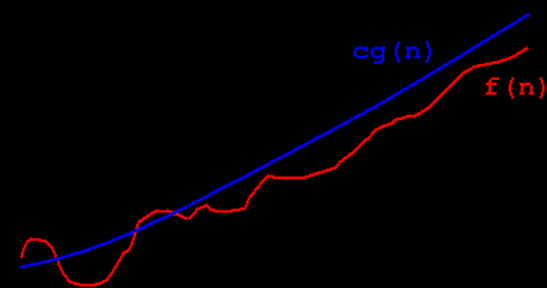
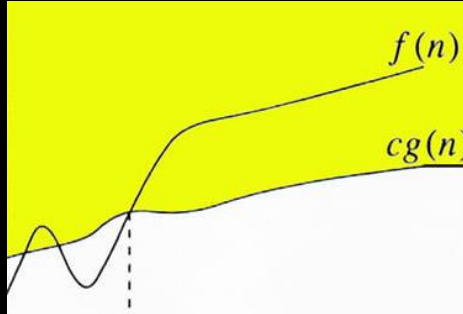
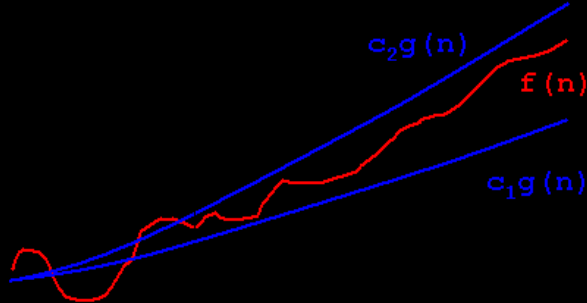
$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c : 0 < c < \infty$$

$$0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq \infty$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$$

$$0 \leq \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$



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

חסם הדוק

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

חסם תחתון

$$f(n) \in O(g(n))$$

חסם עליון

$$\begin{aligned} &\exists c_1 > 0 \\ &\exists c_2 > 0, n_0 \geq 0 : \forall n > n_0 \end{aligned}$$

$$c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n) \geq 0$$

$$\exists c > 0, n_0 \geq 0 : \forall n > n_0$$

$$f(n) \geq c \cdot g(n) \geq 0$$

$$\exists c > 0, n_0 \geq 0 : \forall n > n_0$$

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

$$0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

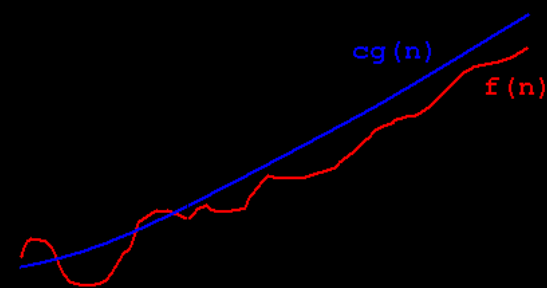
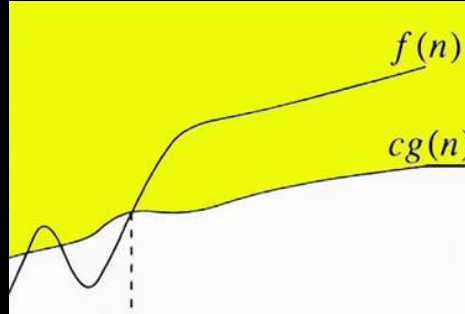
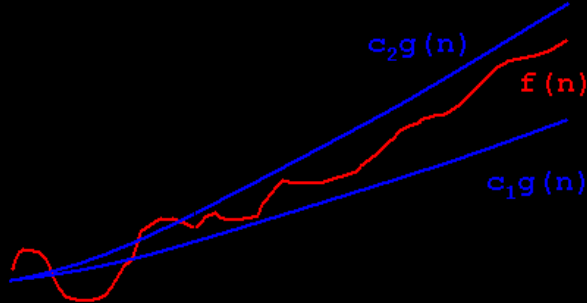
$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c : 0 < c < \infty$$

$$0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq \infty$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$$

$$0 \leq \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$



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

חסם הדוק

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

חסם תחתון

$$f(n) \in O(g(n))$$

חסם עליון

$$\begin{aligned} &\exists c_1 > 0 \\ &\exists c_2 > 0, n_0 \geq 0 : \forall n > n_0 \end{aligned}$$

$$c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n) \geq 0$$

$$\exists c > 0, n_0 \geq 0 : \forall n > n_0$$

$$f(n) \geq c \cdot g(n) \geq 0$$

$$\exists c > 0, n_0 \geq 0 : \forall n > n_0$$

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

$$0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c : 0 < c < \infty$$

$$0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq \infty$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$$

$$0 \leq \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

תרגול שעבר

○ זמני ריצה, אסימפטוטיקה וסימונים Ω, Θ, O

○ **Bubble Sort מיון בועות**

-3	4	88	1	3
----	---	----	---	---

תרגול שעבר

○ זמני ריצה, אסימפטוטיתקה וסימונים Ω, Θ, O

○ Bubble Sort מיון בועות

○ Selection Sort מיון בחירה

5 3 4 1 2

Selection Sort

תרגול שעבר

○ זמני ריצה, אסימפטוטיקה וסימונים Ω, Θ, O

○ Bubble Sort מיון בועות

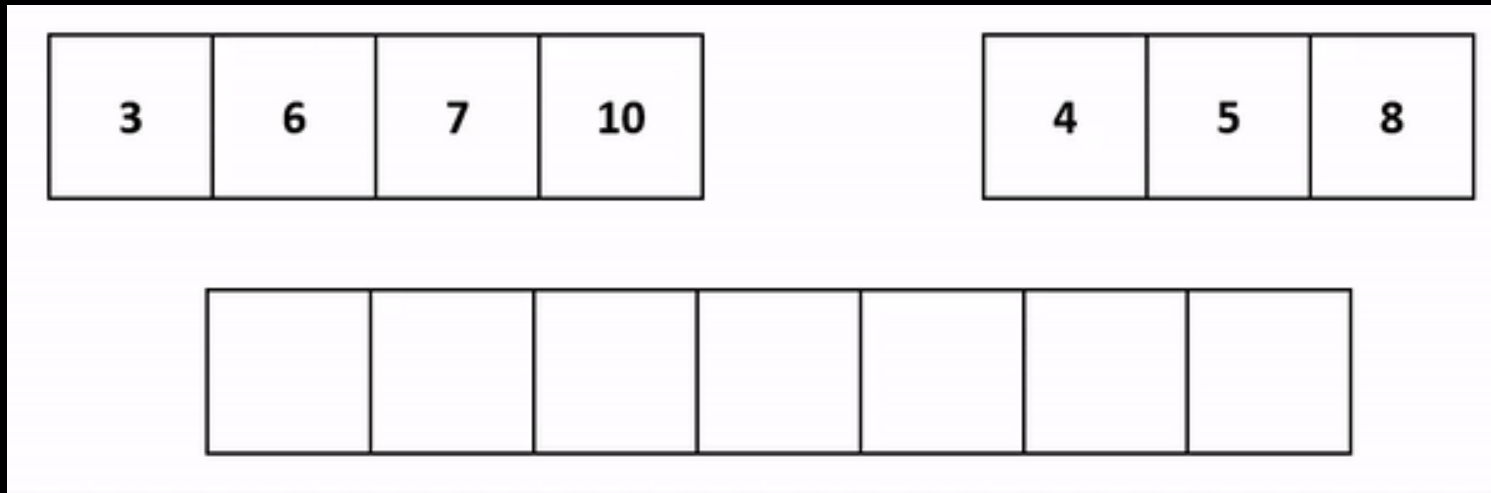
○ Selection Sort מיון בחירה

○ Insertion Sort מיון הכנסה

6 5 3 1 8 7 2 4

תרגול שעבר

○ זמני ריצה, אסימפטוטיקה וסימונים Ω, Θ, O



○ Bubble Sort מיון בועות

○ Selection Sort מיון בחירה

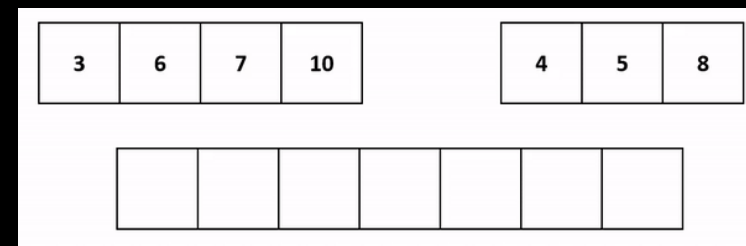
○ Insertion Sort מיון הכנסה

○ Merge

```

private static int[] Merge(int[] sortedA, int[] sortedB) {
    int i = 0;
    int j = 0;
    int k = 0;
    int[] C = new int[sortedA.length + sortedB.length];
    while(i < sortedA.length && j < sortedB.length) {
        if(sortedA[i] < sortedB[j]) C[k++] = sortedA[i++];
        else C[k++] = sortedB[j++];
    }
    while( i < sortedA.length)
        C[k++] = sortedA[i++];
    while(j < sortedB.length)
        C[k++] = sortedB[j++];
    return C;
}

```



Merge

בהינתן מערך ממויין A בגודל n

ומערך B ממויין בגודל m

ניתן למיין למערך חדש C בגודל $|A| + |B|$ ממויין בסיבוכיות

$$\Theta(n + m)$$

תרגול שעבר

○ זמני ריצה, אסימפטוטיקה וסימונים Ω, Θ, O



○ Bubble Sort מיון בועות

○ Selection Sort מיון בחירה

○ Insertion Sort מיון הכנסה

○ Merge

○ Merge Sort מיון מיזוג

תרגול שעבר

○ זמני ריצה, אסימפטוטיקה וסימונים Ω, Θ, O

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$= 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n = 4T\left(\frac{n}{4}\right) + n + n = 4T\left(\frac{n}{4}\right) + 2n$$

$$= 4\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n = 8T\left(\frac{n}{8}\right) + n + 2n = 8T\left(\frac{n}{8}\right) + 3n$$

○ Bubble Sort מיון בועות

$$= 8\left(2T\left(\frac{n}{16}\right) + \frac{n}{8}\right) + 3n = 16T\left(\frac{n}{16}\right) + n + 3n = 16T\left(\frac{n}{16}\right) + 4n$$

○ Selection Sort מיון בחירה

$$\dots = 2^k \cdot T\left(\frac{n}{2^k}\right) + kn \Rightarrow k = \log_2(n)$$

$$= n \cdot T(1) + \log_2(n) \cdot n = \Theta(n \cdot \log_2(n))$$

○ Insertion Sort מיון הכנסה

○ Merge

○ Merge Sort מיון מיזוג

תרגול שעבר

○ חיפוש לינארי

○ זמני ריצה, אסימפטוטיקה וסימונים Ω, Θ, O

○ Bubble Sort מיון בועות

○ Selection Sort מיון בחירה

○ Insertion Sort מיון הכנסה

○ Merge

○ Merge Sort מיון מיזוג

תרגול שעבר

○ זמני ריצה, אסימפטוטיקה וסימונים Ω, Θ, O ○ חיפוש לינארי

○ Bubble Sort מיון בועות ○ חיפוש בינארי

○ Selection Sort מיון בחירה

○ Insertion Sort מיון הכנסה

○ Merge

○ Merge Sort מיון מיזוג

Quick Sort - מיון מהיר

partition

מטרה: בהינתן מערך A ואיבר x
לשים את x במקום הנכון במערך ממויין
ואת כל האיברים שקטנים מ- x לפני x
ואת כל האיברים שגדולים מ- x אחרי x

כל זה אמור להתבצע בזמן לינארי

Quick Sort - מיון מהיר

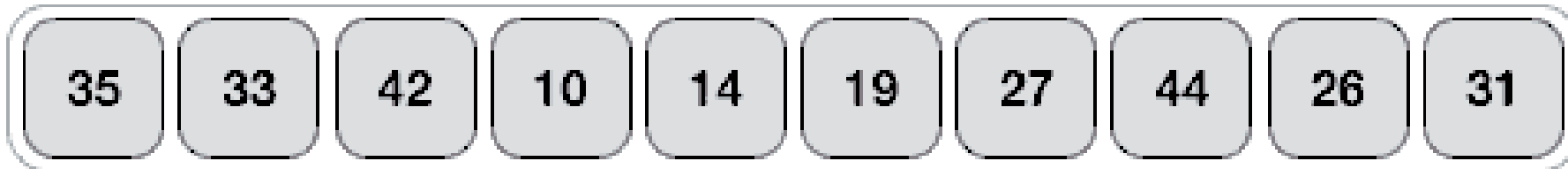
partition

1. Pick a pivot point **How?**
2. Repeat **while** $low \leq high$:
3. $low \leftarrow$ the first element that is \geq than pivot
4. $high \leftarrow$ the first element that is $<$ than pivot
5. Swap the high with low
6. Swap high with pivot

Quick Sort - מיון מהיר

partition

Unsorted Array



Quick Sort - מיון מהיר

```
public static void main(String[] args) {  
    int[] arr = {9,8,7,6,5,4,3,2,1};  
    QuickSort(arr);  
    System.out.println(Arrays.toString(arr));  
}  
  
public static void QuickSort(int[] arr)  
{  
    QuickSort(arr, 0, arr.length-1);  
}
```

```
private static void QuickSort(int[] arr, int start, int end)  
{  
    if(start < end)  
    {  
        int pivot = Partition(arr, start, end);  
        QuickSort(arr, start, pivot-1);  
        QuickSort(arr, pivot+1, end);  
    }  
}
```



```
private static void QuickSort(int[] arr,int start, int end)
{
    if(start < end)
    {
        int pivot = Partition(arr,start,end);
        QuickSort(arr,start,pivot-1);
        QuickSort(arr,pivot+1,end);
    }
}
```

דוגמת ריצה של האלגוריתם



```

private static int Partition(int[] arr,int low, int high)
{
    int pivot = low;
    low++; // [Pivot, Low ... High]

    while(low <= high)
    {
        if(arr[low] < arr[pivot]) low++;
        else if(arr[high] >= arr[pivot]) high--;
        else
            swap(arr,low,high);
    }
    swap(arr,pivot,high); // Pivot = high
    return high;
}

```

Complexity?

0	1	2	3	4	5
1	2	3	4	5	6

// Solution to last week question

```

private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}

```

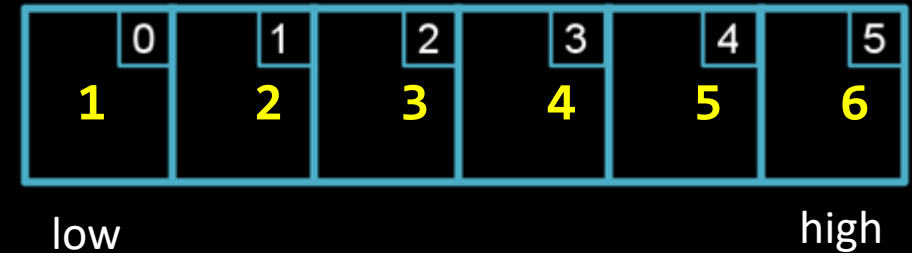
```

private static int Partition(int[] arr,int low, int high)
{
    int pivot = low;
    low++; // [Pivot, Low ... High]

    while(low <= high)
    {
        if(arr[low] < arr[pivot]) low++;
        else if(arr[high] >= arr[pivot]) high--;
        else
            swap(arr,low,high);
    }
    swap(arr,pivot,high); // Pivot = high
    return high;
}

```

Complexity?



// Solution to last week question

```

private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}

```

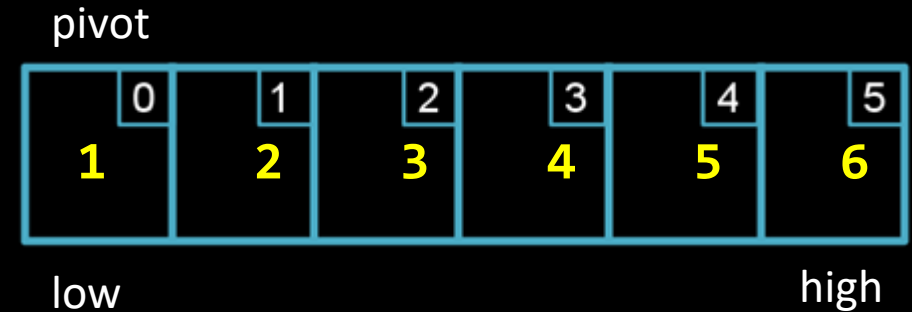
```

private static int Partition(int[] arr,int low, int high)
{
    int pivot = low;
    low++; // [Pivot, Low ... High]

    while(low <= high)
    {
        if(arr[low] < arr[pivot]) low++;
        else if(arr[high] >= arr[pivot]) high--;
        else
            swap(arr,low,high);
    }
    swap(arr,pivot,high); // Pivot = high
    return high;
}

```

Complexity?



// Solution to last week question

```

private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}

```

```
// Solution to last week question
private static void swap(int[] arr, int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}
```

Diagram illustrating the partitioning process in the Quick Sort algorithm. The array is divided into two parts: elements less than the pivot (left) and elements greater than the pivot (right). The pivot is 0.

Index	0	1	2	3	4	5
Value	1	2	3	4	5	6

low: 0 high: 5

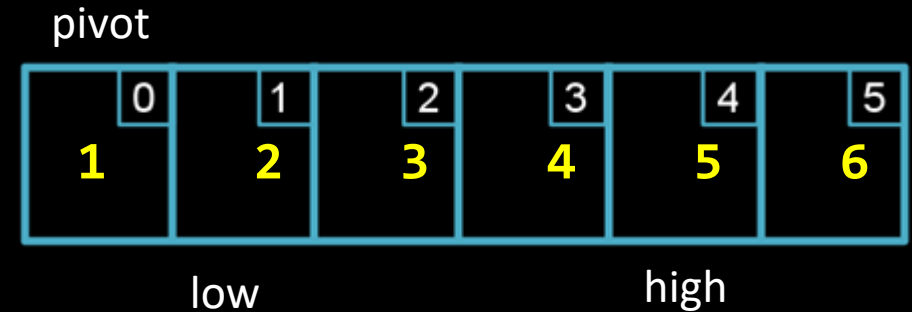

```

private static int Partition(int[] arr,int low, int high)
{
    int pivot = low;
    low++; // [Pivot, Low ... High]

    while(low <= high)
    {
        if(arr[low] < arr[pivot]) low++;
        else if(arr[high] >= arr[pivot]) high--;
        else
            swap(arr,low,high);
    }
    swap(arr,pivot,high); // Pivot = high
    return high;
}

```

Complexity?



// Solution to last week question

```

private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}

```

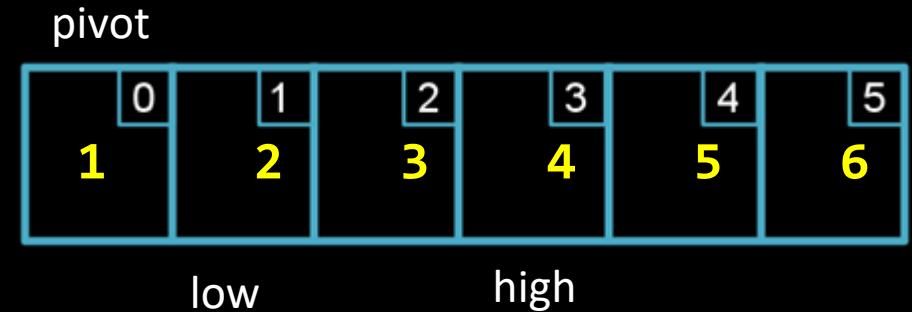
```

private static int Partition(int[] arr,int low, int high)
{
    int pivot = low;
    low++; // [Pivot, Low ... High]

    while(low <= high)
    {
        if(arr[low] < arr[pivot]) low++;
        else if(arr[high] >= arr[pivot]) high--;
        else
            swap(arr,low,high);
    }
    swap(arr,pivot,high); // Pivot = high
    return high;
}

```

Complexity?



// Solution to last week question

```

private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}

```



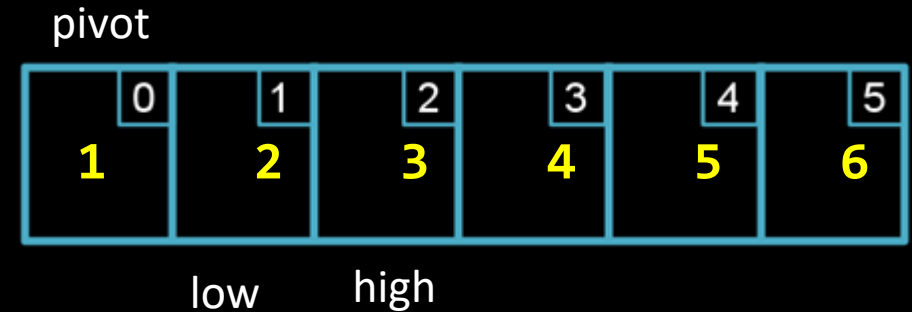
```

private static int Partition(int[] arr,int low, int high)
{
    int pivot = low;
    low++; // [Pivot, Low ... High]

    while(low <= high)
    {
        if(arr[low] < arr[pivot]) low++;
        else if(arr[high] >= arr[pivot]) high--;
        else
            swap(arr,low,high);
    }
    swap(arr,pivot,high); // Pivot = high
    return high;
}

```

Complexity?



// Solution to last week question

```

private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}

```

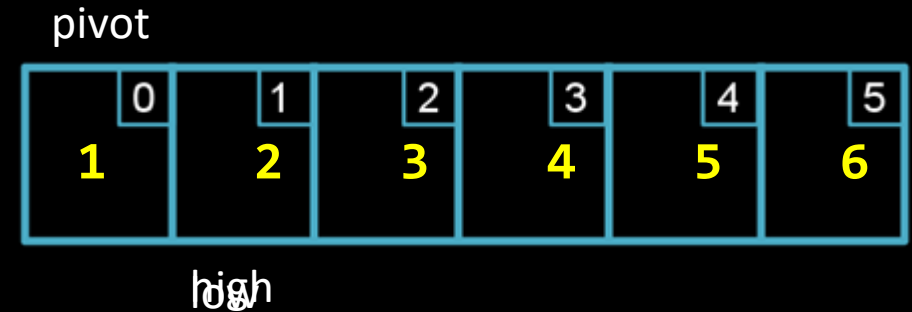
```

private static int Partition(int[] arr,int low, int high)
{
    int pivot = low;
    low++; // [Pivot, Low ... High]

    while(low <= high)
    {
        if(arr[low] < arr[pivot]) low++;
        else if(arr[high] >= arr[pivot]) high--;
        else
            swap(arr,low,high);
    }
    swap(arr,pivot,high); // Pivot = high
    return high;
}

```

Complexity?



// Solution to last week question

```

private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}

```

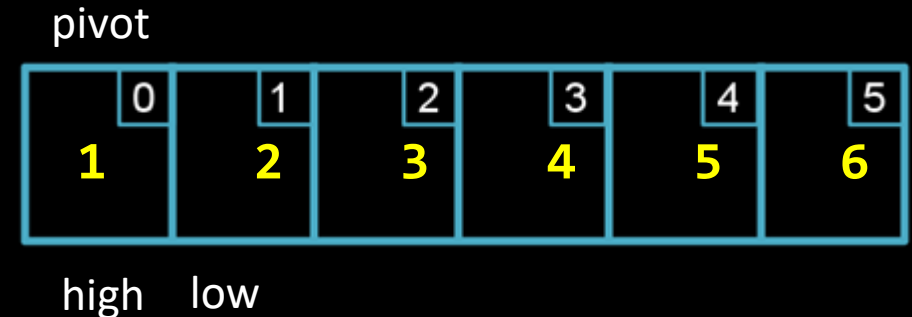
```

private static int Partition(int[] arr,int low, int high)
{
    int pivot = low;
    low++; // [Pivot, Low ... High]

    while(low <= high)
    {
        if(arr[low] < arr[pivot]) low++;
        else if(arr[high] >= arr[pivot]) high--;
        else
            swap(arr,low,high);
    }
    swap(arr,pivot,high); // Pivot = high
    return high;
}

```

Complexity?



$O(n)$

```

// Solution to last week question
private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}

```

Quick Sort - מיון מהיר

pivot

```
int pivot = Partition(arr, start, end);
```

...

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. **Pick a random element as pivot.**
4. Pick median as pivot.

```

private static int Partition(int[] arr,int low, int high)
{
    int pivot = low;
    low++; // [Pivot, Low ... High]
    swap(arr,pivot,pivot + (int) Math.random()*(high-low)); // Swap pivot
    while(low <= high)
    {
        if(arr[low] < arr[pivot]) low++;
        else if(arr[high] >= arr[pivot]) high--;
        else
            swap(arr,low,high);
    }
    swap(arr,pivot,high); // Pivot = high
    return high;
}

```

// Solution to last week question

```

private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}

```

```

private static void QuickSort(int[] arr, int start, int end)
{
    if(start < end)
    {
        int pivot = Partition(arr, start, end);
        QuickSort(arr, start, pivot-1);
        QuickSort(arr, pivot+1, end);
    }
}

```

$$T(n) = T(k) + T(n-k-1) + \Theta(n)$$

Worst Case: $T(n) = T(n-1) + \Theta(n) \Rightarrow \Theta(n^2)$

Best Case: $T(n) = 2T(\frac{n}{2}) + \Theta(n) \Rightarrow \Theta(n \log n)$

```

private static void QuickSort(int[] arr, int start, int end)
{
    if(start < end)
    {
        int pivot = Partition(arr, start, end);
        QuickSort(arr, start, pivot-1);
        QuickSort(arr, pivot+1, end);
    }
}

```

$$T(n) = T(k) + T(n-k-1) + \Theta(n)$$

Worst Case: $T(n) = T(n-1) + \Theta(n) \Rightarrow \Theta(n^2)$

Best Case: $T(n) = 2T(\frac{n}{2}) + \Theta(n) \Rightarrow \Theta(n \log n)$

תרגיל (Partition)

כתוב פונקציה סטטית שמקבלת מערך של מספרים שלמים וממיינת אותו כך שמספרים זוגיים נמצאים בתחילת המערך, ומספרים אי-זוגיים נמצאים בסוף המערך. הסיבוכיות $O(N)$.

דוגמה: קלט: {-3,6,12,4,-7,45,-6,-3,-1,2,3,10,1,2,3,4,5}

פלט: {6, 12, 4, 2, 10, -6, 2, -1, -3, 3, 45, 1, -7, 3, -3, 5}

```
// This function is responsible to partition the array into even side and odd side
```

```
// [even.....odd]
```

```
private static void Question6(int[] arr) {
```

```
int low = 0;
```

```
int high = arr.length - 1;
```

```
while(low <= high) {
```

```
    if(arr[low]%2 == 0) low++; // Will Stay at the first that odd
```

```
    else if(arr[high]%2 != 0) high--; // Will stay at the first that even
```

```
    else { // Swapping
```

```
        int temp = arr[low];
```

```
        arr[low] = arr[high];
```

```
        arr[high] = temp;
```

```
        High--; low++;
```

```
    }
```

```
}
```

```
}  
צבי מינץ
```


מיון מבוסס השוואות

Comparison Sort



מיון מבוסס השוואות

Comparison Sort

מיון מבוסס השוואות מסדר אלמנטים במערך ע"י השוואות, בדרך כלל ע"י האופרטור \leq

מיונים לא מבוססי השוואות

Counting Sort ○

Radix Sort ○

...

מיונים מבוססי השוואות

Bubble Sort מיון בועות ○

Selection Sort מיון בחירה ○

Insertion Sort מיון הכנסה ○

Merge Sort מיון מיזוג ○

Quick Sort מיון מהיר ○

...

פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיזון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיזון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיזון המבוססים על פעולת השוואה דורשים לפחות

כל אלגוריתמי המיזון המבוססים על פעולת השוואה דורשים לפחות
 $\Omega(n \cdot \log(n))$
פעולות השוואה במקרה הגרוע על-מנת לבצעם

עץ החלטה

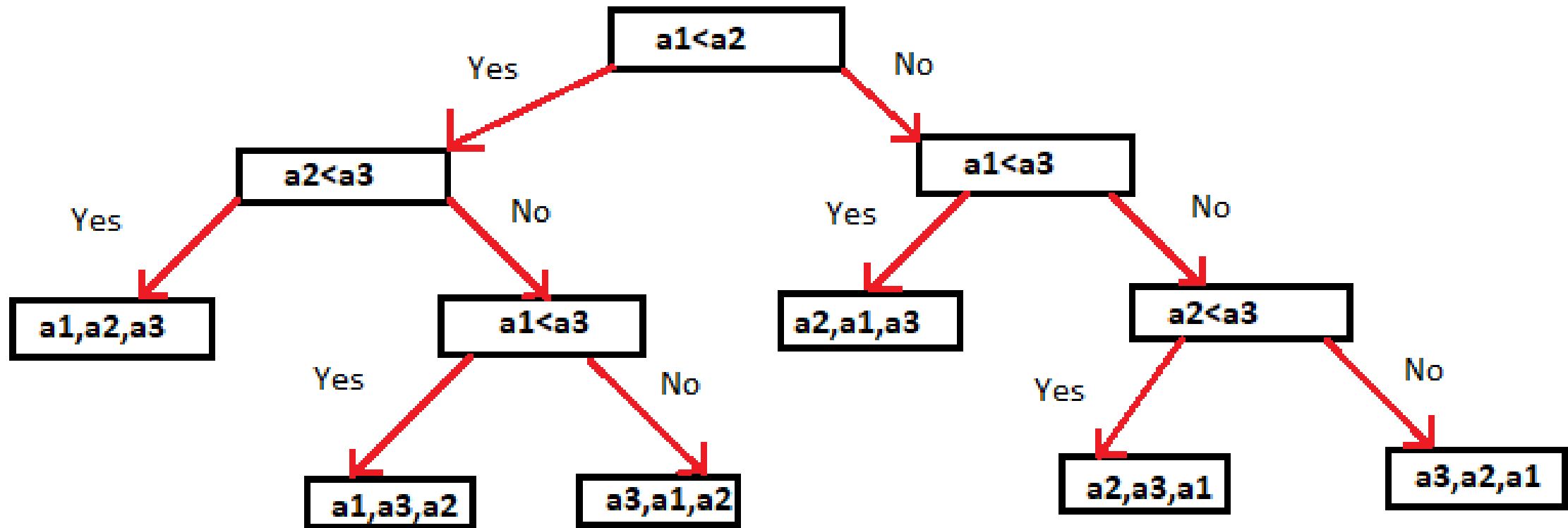
עץ החלטה הוא מודל חיזוי בתחומי הסטטיסטיקה,

עץ החלטה הוא עץ בינארי מלא המורכב מצמתי החלטה שבכל אחד מהם נבדק תנאי מסוים על מאפיין מסוים ועלים המכילים את הערך החזוי עבור התצפית המתאימה למסלול שמוביל אליהם בעץ.

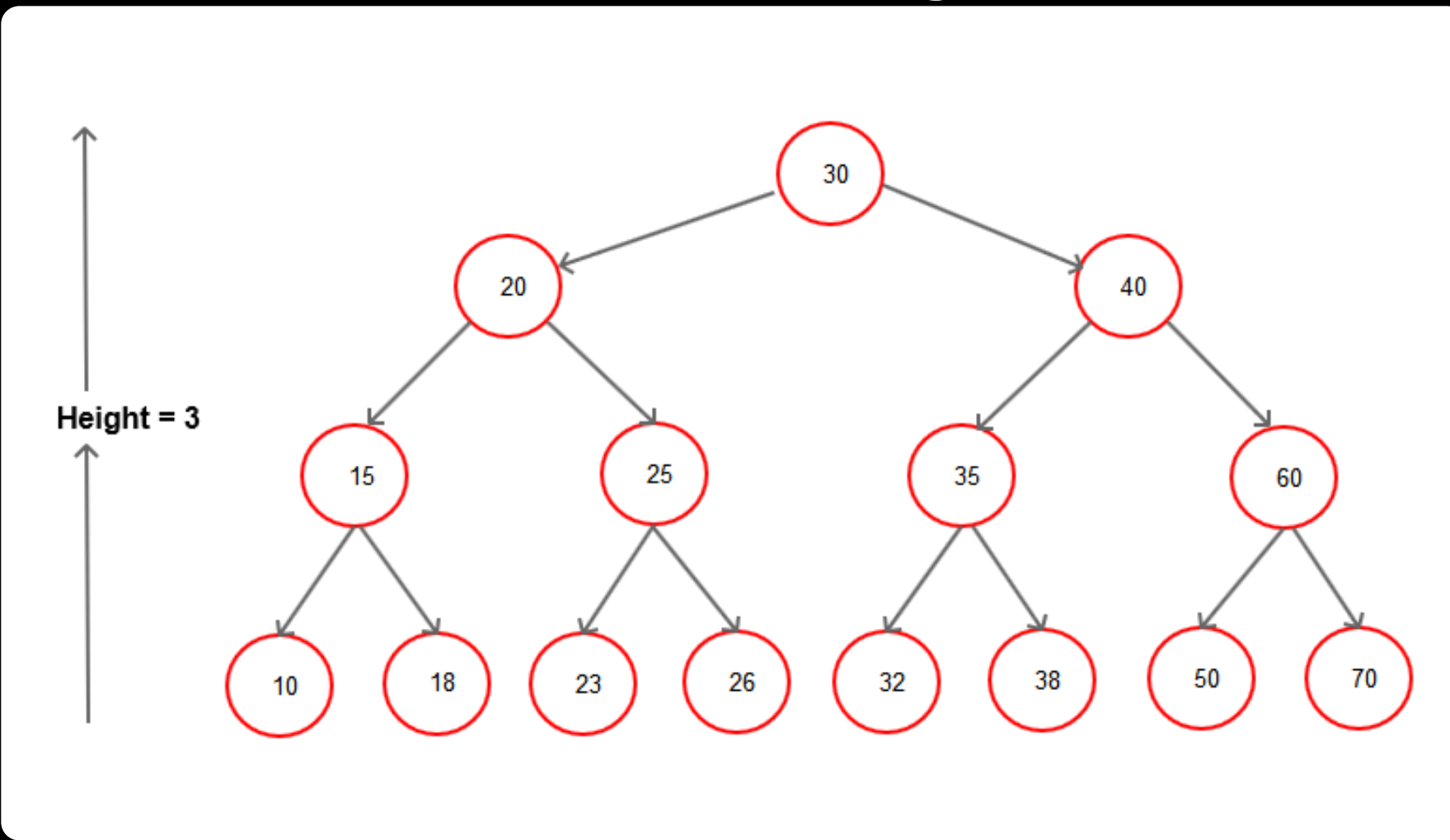
- עץ בינארי כאשר כל קודקוד פנימי מתוויג ע"י תווית $a_i \leq a_j$
- ההפעלה של האלגוריתם תואמת את המסלול שורש-עלה
- כל עלה מחזיק את התוצאה של החישוב

עץ החלטה

נרצה למיין את הקבוצה $\{a_1, a_2, a_3\}$



כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות $\Omega(n \cdot \log(n))$



הוכחה

- תהי קבוצה
- חייב להיות
- בעץ בינארי

(ברורים)

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות $\Omega(n \cdot \log(n))$

פעולות השוואה במקרה הגרוע על-מנת לבצעם

הוכחה

- תהי קבוצה בעלת n איברים שונים ממוספרים מ-1 עד n
- חייב להיות $n!$ עלים (עבור כל אחד מ- $n!$ הפרמוטציות של n האיברים)
- בעץ בינארי בגובה h יש לכל היותר 2^h עלים

$$2^h \overset{\text{כמות העלים בפועל}}{\geq} n! \quad \Rightarrow \quad h \geq \log_2(n!)$$

כמות העלים בעץ בינארי לכל היותר

Stirling's approximation

$$n! \sim \left(\frac{n}{e}\right)^n \cdot \sqrt{2\pi n}$$

$$\begin{aligned}\log(n!) &= \log\left(\left(\frac{n}{e}\right)^n \cdot \sqrt{2\pi n}\right) \\&= \log\left(\left(\frac{n}{e}\right)^n\right) + \log\left((2\pi n)^{\frac{1}{2}}\right) \\&= n \cdot \log\left(\frac{n}{e}\right) + \frac{1}{2} \cdot \log(2\pi n) \\&= n \cdot \log n - n \cdot \log_2 e + \frac{1}{2} \cdot \log(2\pi n) \\&= \Omega(n \cdot \log n)\end{aligned}$$

מיונים לינארים

המחיר ליעילות זו היא אובדן כלליות. יעילות זו מושגת בעזרת הנחות נוקשות למדי לגבי הקלט. שינויים בקלט יכולים לגרום לאלגוריתמים לא לעבוד כלל, או לעבוד בסיבוכיות גבוהה מאד.

Counting Sort - מיון מנייה

אלגוריתם מיון עבור מספרים שלמים המתבסס על העובדה
שהמספרים נמצאים בטווח חסום כדי לבצע את המיון בזמן מהיר
יותר מזה שמסוגלים לו אלגוריתמי המיון הכלליים.

קלט: מערך חד-מימדי

פלט: מערך חד-מימדי ממויין

Counting Sort - מיון מנייה

A מערך קלט בגודל n

B פלט האלגוריתם – מערך ממויין בגודל n

C – מערך עזר בגודל k כאשר $k = \max_A - \min_A + 1$

Counting-Sort (A, B, k)

$\max \leftarrow \max \text{ value in } A$

$\min \leftarrow \min \text{ value in } A$

Initialize $C[\max - \min + 1]$ s.t $C[i] = 0$ for all i

for $i \leftarrow 0$ to n

$C[A[i] - \min] \leftarrow C[A[i] - \min] + 1$ // כעת $C[i]$ מחזיק בתוכו את כמות החזרות של i ב A

for $i \leftarrow 1$ to k

$C[i] \leftarrow C[i] + C[i-1]$ // $C[i]$ מחזיק בתוכו את כמות האיברים אשר קטנים שווים ל- i ב A

for $j \leftarrow n$ downto 1

$B[C[A[j] - \min]] \leftarrow A[j]$

$C[A[j] - \min] \leftarrow C[A[j] - \min] - 1$

return B

Counting Sort - מיון מנייה

A מערך קלט בגודל n

B פלט האלגוריתם – מערך ממויין בגודל n

C – מערך עזר בגודל k כאשר $k = \max_A - \min_A + 1$

Counting-Sort (A, B, k)

$\max \leftarrow \max \text{ value in } A$

$\min \leftarrow \min \text{ value in } A$

Initialize $C[\max - \min + 1]$ s.t $C[i] = 0$ for all i

for $i \leftarrow 0$ to n

$C[A[i] - \min] \leftarrow C[A[i] - \min] + 1$ // כעת $C[i]$ מחזיק בתוכו את כמות החזרות של i ב A

for $i \leftarrow 1$ to k

$C[i] \leftarrow C[i] + C[i-1]$ // $C[i]$ מחזיק בתוכו את כמות האיברים אשר קטנים שווים ל- i ב A

for $j \leftarrow n$ downto 1

איבר x במערך המקורי

$B[C[A[j] - \min]] \leftarrow A[j]$

$C[A[j] - \min] \leftarrow C[A[j] - \min] - 1$

return B

Counting Sort - מיון מנייה

A מערך קלט בגודל n

B פלט האלגוריתם – מערך ממויין בגודל n

C – מערך עזר בגודל k כאשר $k = \max_A - \min_A + 1$

Counting-Sort (A, B, k)

$\max \leftarrow \max \text{ value in } A$

$\min \leftarrow \min \text{ value in } A$

Initialize $C[\max - \min + 1]$ s.t $C[i] = 0$ for all i

for $i \leftarrow 0$ to n

$C[A[i] - \min] \leftarrow C[A[i] - \min] + 1$ // כעת $C[i]$ מחזיק בתוכו את כמות החזרות של i ב A

for $i \leftarrow 1$ to k

$C[i] \leftarrow C[i] + C[i-1]$ // $C[i]$ כעת מחזיק בתוכו את כמות האיברים אשר קטנים שווים ל- i ב A

for $j \leftarrow n$ downto 1

$B[C[A[j] - \min]] \leftarrow A[j]$

$C[A[j] - \min] \leftarrow C[A[j] - \min] - 1$

return B

כמות האיברים אשר קטנים שווים ל- x

Counting Sort - מיון מנייה

A מערך קלט בגודל n

B פלט האלגוריתם – מערך ממויין בגודל n

C – מערך עזר בגודל k כאשר $k = \max_A - \min_A + 1$

Counting-Sort (A, B, k)

$\max \leftarrow \max \text{ value in } A$

$\min \leftarrow \min \text{ value in } A$

Initialize $C[\max - \min + 1]$ s.t $C[i] = 0$ for all i

for $i \leftarrow 0$ to n

$C[A[i] - \min] \leftarrow C[A[i] - \min] + 1$ // כעת $C[i]$ מחזיק בתוכו את כמות החזרות של i ב A

for $i \leftarrow 1$ to k

$C[i] \leftarrow C[i] + C[i-1]$ // $C[i]$ כעת מחזיק בתוכו את כמות האיברים אשר קטנים שווים ל- i ב A

for $j \leftarrow n$ downto 1

$B[C[A[j] - \min]] \leftarrow A[j]$

$C[A[j] - \min] \leftarrow C[A[j] - \min] - 1$

return B

צבי מינץ

התא של

כמות האיברים אשר קטנים שווים ל- x

תהיה שווה ל- x

Counting Sort - מיון מנייה

A מערך קלט בגודל n

B פלט האלגוריתם – מערך ממויין בגודל n

C – מערך עזר בגודל k כאשר $k = \max_A - \min_A + 1$

Counting-Sort (A, B, k)

$\max \leftarrow \text{max value in } A$

$\min \leftarrow \text{min value in } A$

Initialize $C[\max - \min + 1]$ s.t $C[i] = 0$ for all i

for $i \leftarrow 0$ to n

$C[A[i] - \min] \leftarrow C[A[i] - \min] + 1$ // כעת $C[i]$ מחזיק בתוכו את כמות החזרות של i ב A

for $i \leftarrow 1$ to k

$C[i] \leftarrow C[i] + C[i-1]$ // $C[i]$ מחזיק בתוכו את כמות האיברים אשר קטנים שווים ל- i ב A

for $j \leftarrow n$ downto 1

$B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

return B

צבי מינץ

$O(n + k)$

Counting Sort - מיון מנייה

A מערך קלט בגודל n

B פלט האלגוריתם – מערך ממויין בגודל n

C – מערך עזר בגודל k כאשר $k = \max_A - \min_A + 1$

Counting-Sort (A, B, k)

$\max \leftarrow \text{max value in } A$

$\min \leftarrow \text{min value in } A$

Initialize $C[\max - \min + 1]$ s.t $C[i] = 0$ for all i

for $i \leftarrow 0$ to n

$C[A[i] - \min] \leftarrow C[A[i] - \min] + 1$ // כעת $C[i]$ מחזיק בתוכו את כמות החזרות של i ב A

for $i \leftarrow 1$ to k

$C[i] \leftarrow C[i] + C[i-1]$ // $C[i]$ מחזיק בתוכו את כמות האיברים אשר קטנים שווים ל- i ב A

for $j \leftarrow n$ downto 1

$B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

return B

צבי מינץ

$$O(n + k)$$

Counting Sort - מיון מנייה

אבחנה:

הסדר בין איברים שווים נשמר

זאת משום שתא ה- $C[i]$ נשמר המופע האחרון המיועד של האיבר ה- i במערך B ומשום שהמעבר על המערך A הוא מהסוף להתחלה

מה שגורם את Counting Sort למיון יציב (Stable)

מיון יציב Stable Sort

מיון נקרא **מיון יציב** אם הוא שומר על הסדר של הנתונים לאחר המיון גם כשיש שני נתונים זהים.

דוגמה להבחנה בין מיון יציב ללא-יציב:

אם רוצים למיין רשימת שמות על פי שם משפחה, אך אם שמות המשפחה זהים, למיין על פי השם הפרטי, אפשר למיין את המערך על פי שם פרטי ואחר כך למיין שוב על פי שם המשפחה. דבר זה יתאפשר רק אם המיון הוא יציב, אך אם הוא אינו יציב, אזי יכול להיות שהסדר הפנימי של השמות הפרטיים ייהרס.

פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

לסיכום

	מבוסס השוואות					לא מבוסס השוואות	
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Radix Sort	Counting Sort
O	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n^2)$	תרגול הבא	$O(n + k)$
Ω	$\Omega(n^2)$ ניתן גם כן ב- $\Omega(n)$	$\Omega(n^2)$	$\Omega(n)$	$\Omega(n \cdot \log n)$	$\Omega(n \cdot \log n)$	תרגול הבא	$\Omega(n + k)$
Θ	$\theta(n^2)$	$\Theta(n^2)$		$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$ בהסתברות גבוהה	תרגול הבא	$\Theta(n + k)$

פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

לסיכום

	מבוסס השוואות					לא מבוסס השוואות	
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Radix Sort	Counting Sort
O	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n^2)$	תרגול הבא	$O(n + k)$
Ω	$\Omega(n^2)$ ניתן גם כן ב- $\Omega(n)$	$\Omega(n^2)$	$\Omega(n)$	$\Omega(n \cdot \log n)$	$\Omega(n \cdot \log n)$	תרגול הבא	$\Omega(n + k)$
Θ	$\Theta(n^2)$	$\Theta(n^2)$		$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$ בהסתברות גבוהה	תרגול הבא	$\Theta(n + k)$

פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

לסיכום

	מבוסס השוואות					לא מבוסס השוואות	
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Radix Sort	Counting Sort
O	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n^2)$	תרגול הבא	$O(n + k)$
Ω	$\Omega(n^2)$ ניתן גם כן ב- $\Omega(n)$	$\Omega(n^2)$	$\Omega(n)$	$\Omega(n \cdot \log n)$	$\Omega(n \cdot \log n)$	תרגול הבא	$\Omega(n + k)$
Θ	$\theta(n^2)$	$\Theta(n^2)$		$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$ בהסתברות גבוהה	תרגול הבא	$\Theta(n + k)$

פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

לסיכום

	מבוסס השוואות					לא מבוסס השוואות	
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Radix Sort	Counting Sort
O	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n^2)$	תרגול הבא	$O(n + k)$
Ω	$\Omega(n^2)$ ניתן גם כן ב- $\Omega(n)$	$\Omega(n^2)$	$\Omega(n)$	$\Omega(n \cdot \log n)$	$\Omega(n \cdot \log n)$	תרגול הבא	$\Omega(n + k)$
Θ	$\theta(n^2)$	$\theta(n^2)$		$\theta(n \cdot \log n)$	$\theta(n \cdot \log n)$ בהסתברות גבוהה	תרגול הבא	$\theta(n + k)$

פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

לסיכום

	מבוסס השוואות					לא מבוסס השוואות	
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Radix Sort	Counting Sort
O	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n^2)$	תרגול הבא	$O(n + k)$
Ω	$\Omega(n^2)$ ניתן גם כן ב- $\Omega(n)$	$\Omega(n^2)$	$\Omega(n)$	$\Omega(n \cdot \log n)$	$\Omega(n \cdot \log n)$	תרגול הבא	$\Omega(n + k)$
Θ	$\theta(n^2)$	$\Theta(n^2)$		$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$ בהסתברות גבוהה	תרגול הבא	$\Theta(n + k)$

פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

לסיכום

	מבוסס השוואות					לא מבוסס השוואות	
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Radix Sort	Counting Sort
O	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n^2)$	תרגול הבא	$O(n + k)$
Ω	$\Omega(n^2)$ ניתן גם כן ב- $\Omega(n)$	$\Omega(n^2)$	$\Omega(n)$	$\Omega(n \cdot \log n)$	$\Omega(n \cdot \log n)$	תרגול הבא	$\Omega(n + k)$
Θ	$\theta(n^2)$	$\Theta(n^2)$		$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$ בהסתברות גבוהה	תרגול הבא	$\Theta(n + k)$

פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות

לסיכום

	מבוסס השוואות					לא מבוסס השוואות	
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Radix Sort	Counting Sort
O	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n^2)$	תרגול הבא	$O(n + k)$
Ω	$\Omega(n^2)$ ניתן גם כן ב- $\Omega(n)$	$\Omega(n^2)$	$\Omega(n)$	$\Omega(n \cdot \log n)$	$\Omega(n \cdot \log n)$	תרגול הבא	$\Omega(n + k)$
Θ	$\theta(n^2)$	$\Theta(n^2)$		$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$ בהסתברות גבוהה	תרגול הבא	$\Theta(n + k)$

עבודה עצמית

