מבני נתונים

תרגול 2 – המשך סיבוכיות, חסם תחתון ומיון מנייה

zvimints@gmail.com – צבי מינץ

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 O,Θ,Ω

```
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:

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

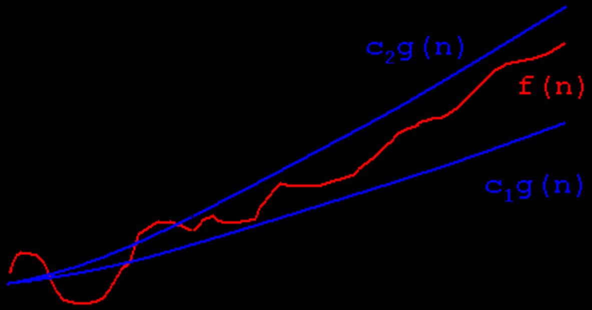
```
int foo(int n) {
   if(n == 1 || n == 2) return 1;
   return foo(n-2) + foo(n-1);
}
```

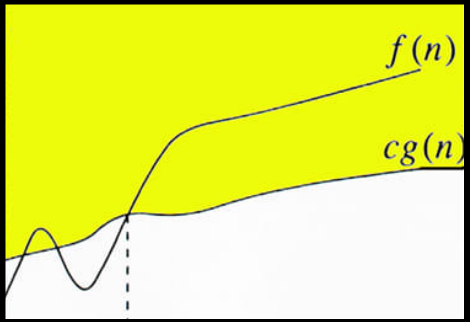
נסמן ב-$T(n)$ את מספר הפעולות ש- $foo(int)$ מבצעת על הקלט n, מכאן $T(2) = T(1) = 1$
בנוסף $T(n) = T(n+1) + T(n-2)$
לבסוף נקבל כי:
$T(n) = c_1 \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n = O(\left(\frac{1+\sqrt{5}}{2}\right)^n)$ **(פתרון נלמד בקורס בדידה)**

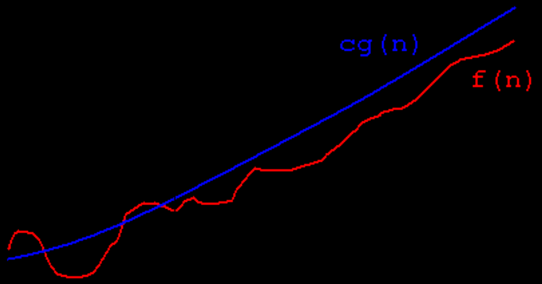
```java
void foo(int n, int a, int b) {
   while(a<n) {
      a=Math.pow(a,b)
      }
}
```

נסמן ב-k את מספר האיטרציות של הלולאה, הלולאה תסתיים כאשר $a^{b^k} > n$ ולכן כמות האיטרציות היא לפחות $k > \log_b \log_a n$

נניח כי n= a^{b^t} ואם לא, נשלם לחזקה הבאה הקרובה
לכן הסיבוכיות היא $O(\log_b \log_a n)$

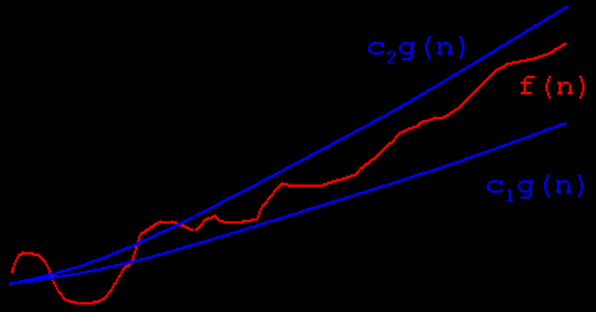


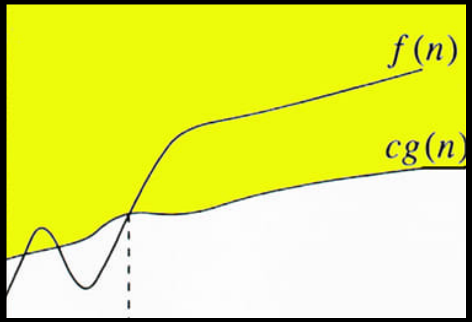


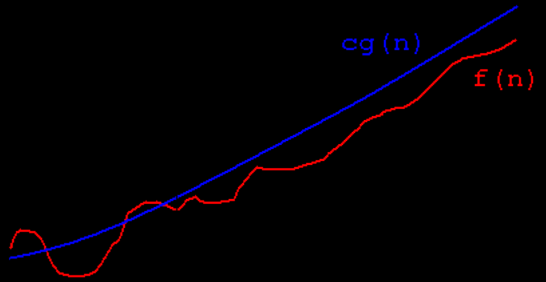


$\boldsymbol{f(n) \in \Theta(g(n))}$ חסם הדוק	$\boldsymbol{f(n) \in \Omega(g(n))}$ חסם תחתון	$\boldsymbol{f(n) \in O(g(n))}$ חסם עליון
$\begin{matrix}\exists c_1 > 0 \\ \exists c_2 > 0\end{matrix}, n_0 \geq 0 : \ \forall n > n_0$ $c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n) \geq 0$	$\exists c > 0, n_0 \geq 0 : \ \forall n > n_0$ $f(n) \geq c \cdot g(n) \geq 0$	$\exists c > 0, n_0 \geq 0 : \ \forall n > n_0$ $f(n) \leq c \cdot g(n)$
$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$ $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \mathrm{c} : 0 < c < \infty$	$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} \leq \infty$ $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$	$0 \leq \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$ $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

צבי מינץ

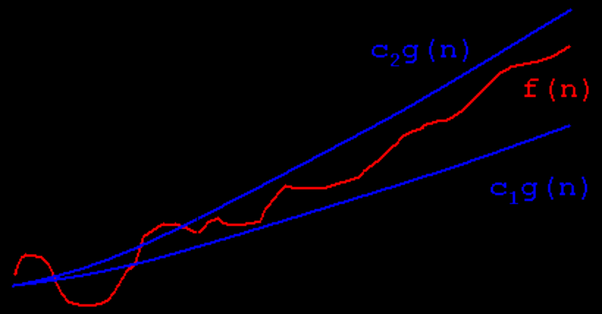


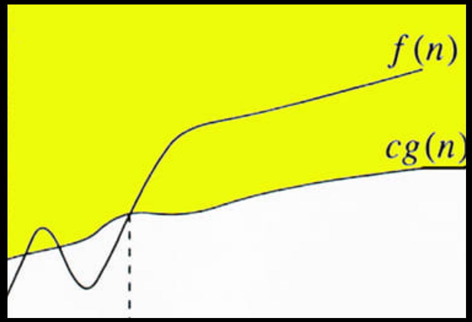


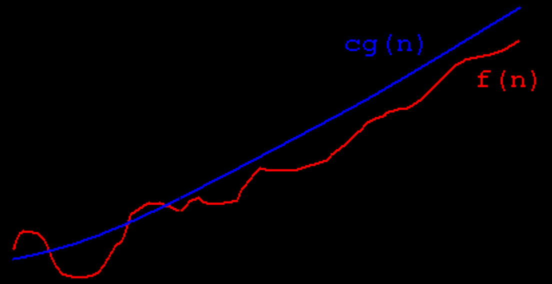


$f(n) \in \Theta(g(n))$ חסם הדוק	$f(n) \in \Omega(g(n))$ חסם תחתון	$f(n) \in O(g(n))$ חסם עליון
$\begin{matrix} \exists c_1 > 0 \\ \exists c_2 > 0 \end{matrix}, n_0 \geq 0 : \ \forall n > n_0$ $c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n) \geq 0$	$\exists c > 0, n_0 \geq 0 : \ \forall n > n_0$ $f(n) \geq c \cdot g(n) \geq 0$	$\exists c > 0, n_0 \geq 0 : \ \forall n > n_0$ $f(n) \leq c \cdot g(n)$
$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$ $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \mathrm{c} : 0 < c < \infty$	$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} \leq \infty$ $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$	$0 \leq \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$ $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

צבי מינץ

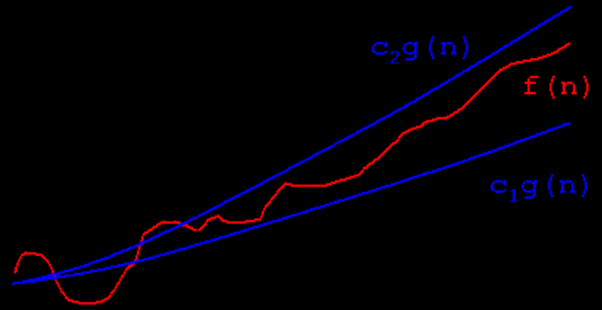


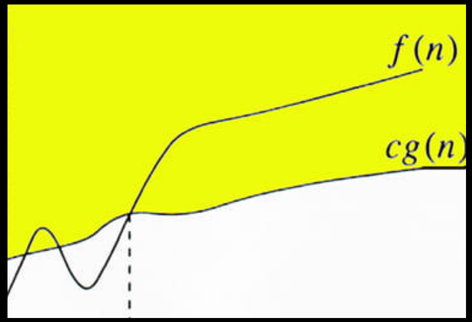


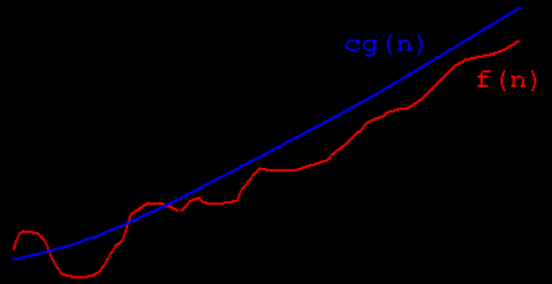


$\boldsymbol{f(n) \in \Theta(g(n))}$ חסם הדוק	$\boldsymbol{f(n) \in \Omega(g(n))}$ חסם תחתון	$\boldsymbol{f(n) \in O(g(n))}$ חסם עליון
$\begin{matrix}\exists c_1 > 0 \\ \exists c_2 > 0\end{matrix}, n_0 \geq 0 : \; \forall n > n_0$ $c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n) \geq 0$	$\exists c > 0, n_0 \geq 0 : \; \forall n > n_0$ $f(n) \geq c \cdot g(n) \geq 0$	$\exists c > 0, n_0 \geq 0 : \; \forall n > n_0$ $f(n) \leq c \cdot g(n)$
$0 < \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$ $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \mathrm{c} : 0 < c < \infty$	$0 < \lim_{n \to \infty} \frac{f(n)}{g(n)} \leq \infty$ $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$	$0 \leq \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$ $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

צבי מינץ



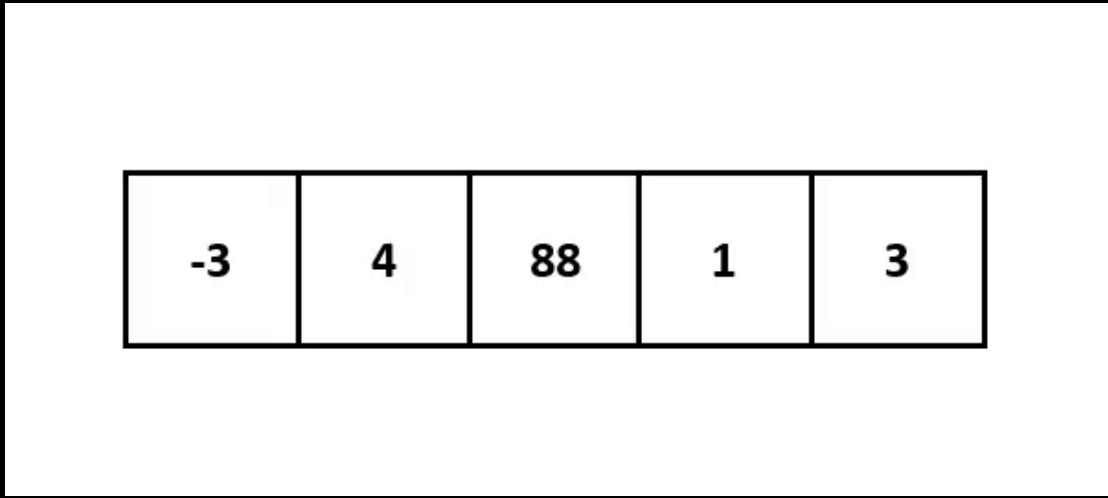




$\boldsymbol{f(n) \in \Theta(g(n))}$ חסם הדוק	$\boldsymbol{f(n) \in \Omega(g(n))}$ חסם תחתון	$\boldsymbol{f(n) \in O(g(n))}$ חסם עליון
$\begin{matrix}\exists c_1 > 0 \\ \exists c_2 > 0\end{matrix}, n_0 \geq 0 : \forall n > n_0$ $c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n) \geq 0$	$\exists c > 0, n_0 \geq 0 : \forall n > n_0$ $f(n) \geq c \cdot g(n) \geq 0$	$\exists c > 0, n_0 \geq 0 : \forall n > n_0$ $f(n) \leq c \cdot g(n)$
$0 < \lim\limits_{n\to\infty} \frac{f(n)}{g(n)} < \infty$ $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \mathrm{c} : 0 < c < \infty$	$0 < \lim\limits_{n\to\infty} \frac{f(n)}{g(n)} \leq \infty$ $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \infty$	$0 \leq \lim\limits_{n\to\infty} \frac{f(n)}{g(n)} < \infty$ $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0$

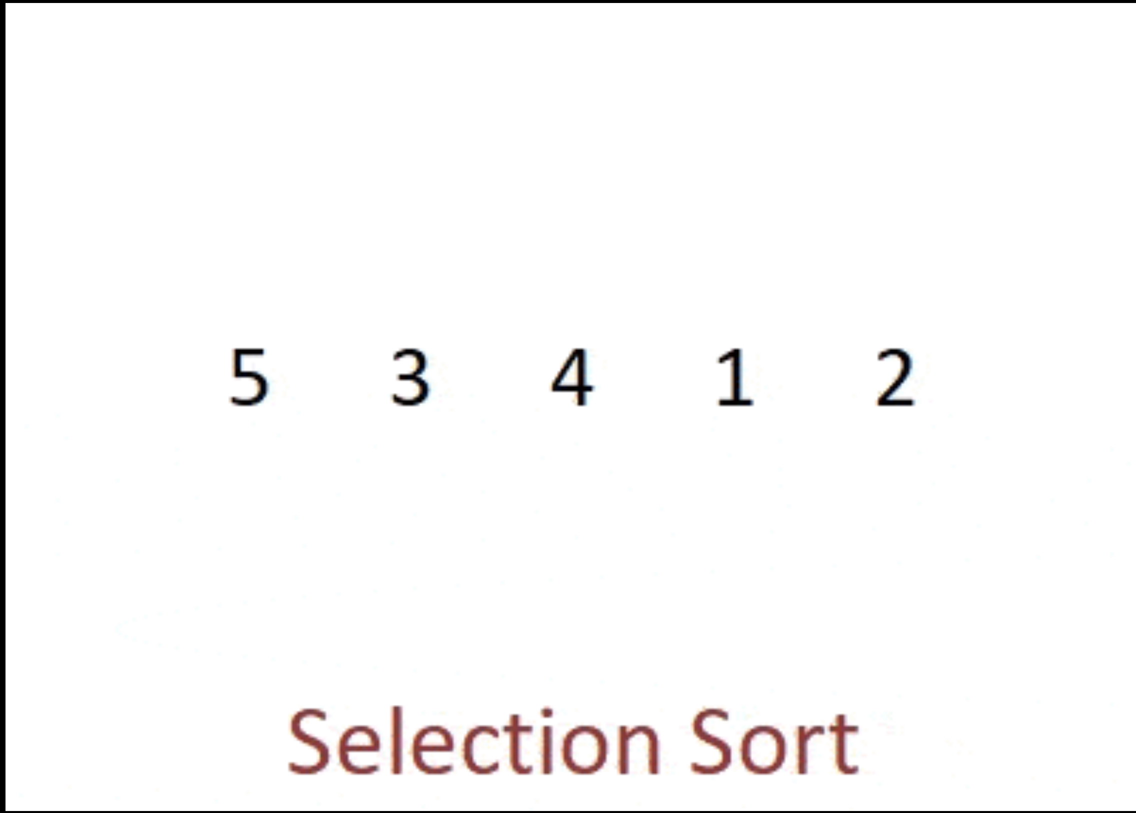
תרגול שעבר

- זמני ריצה, אסימפטוטיקה וסימונים Ω, Θ, O
- Bubble Sort מיון בועות



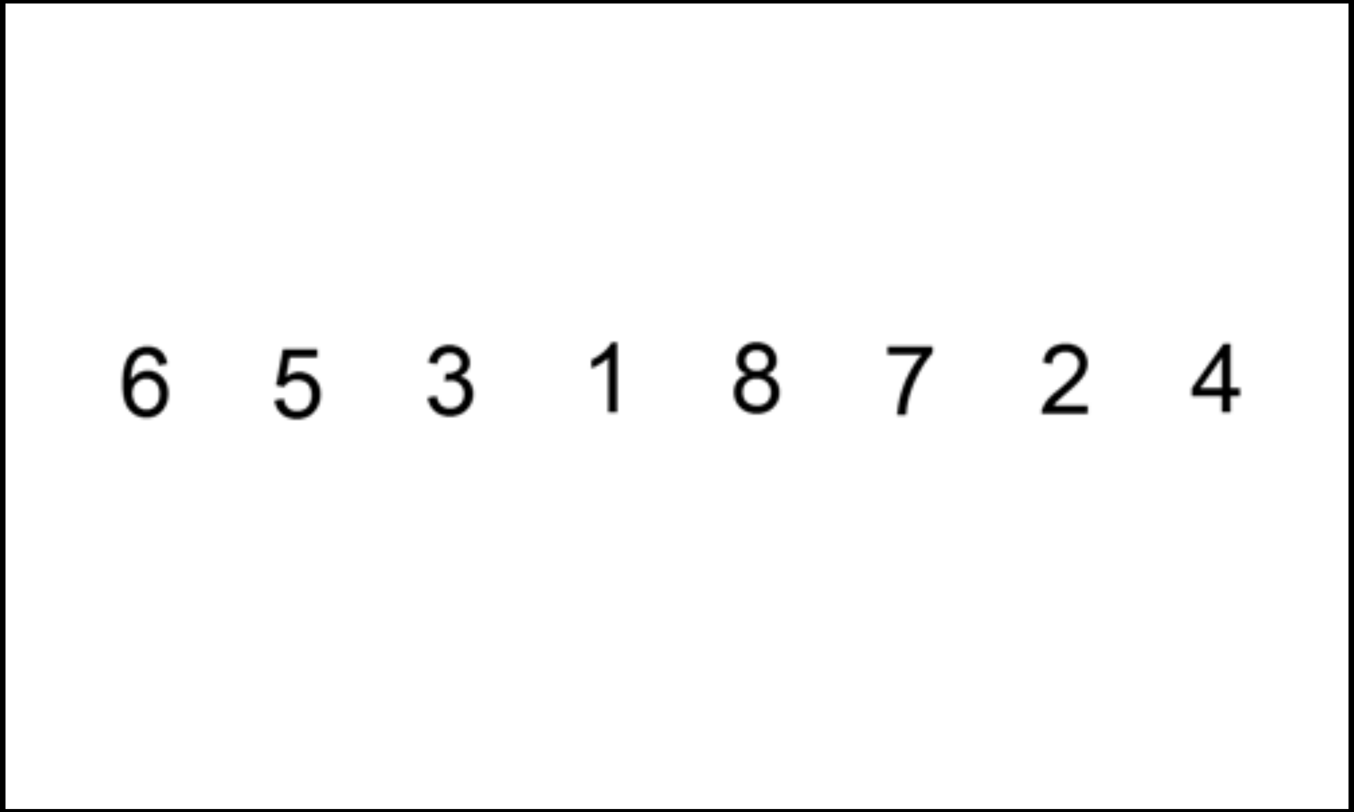
תרגול שעבר

- זמני ריצה, אסימפטוטיקה וסימונים Ω, Θ, O
- Bubble Sort מיון בועות
- Selection Sort מיון בחירה



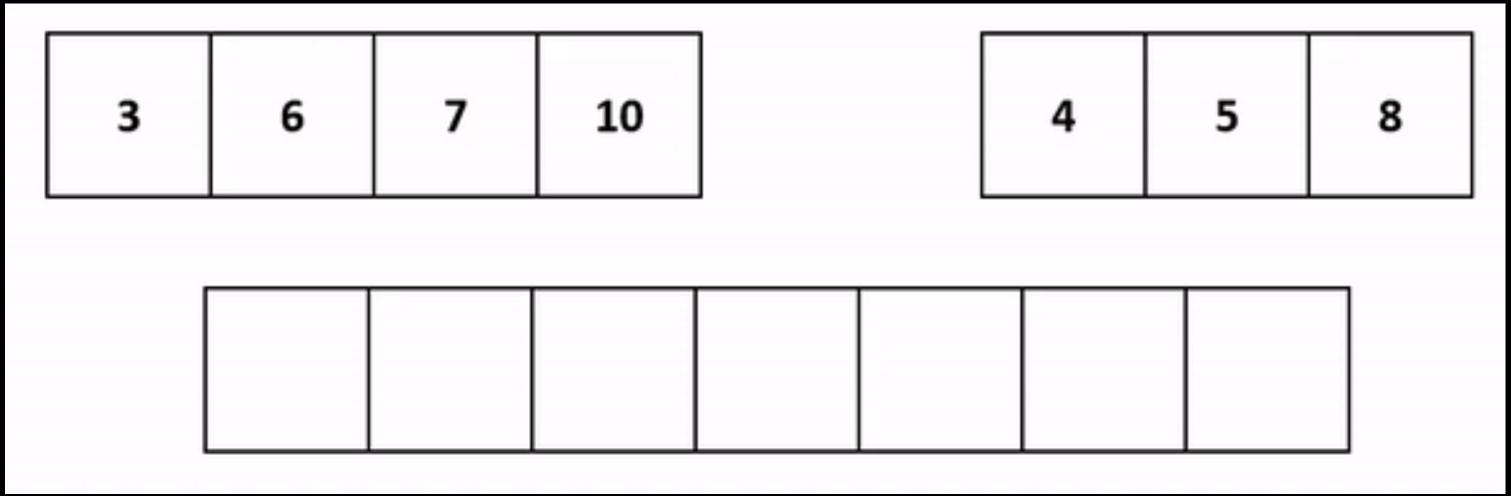
תרגול שעבר

- זמני ריצה, אסימפטוטיקה וסימונים Ω,Θ,O
- Bubble Sort מיון בועות
- Selection Sort מיון בחירה
- Insertion Sort מיון הכנסה

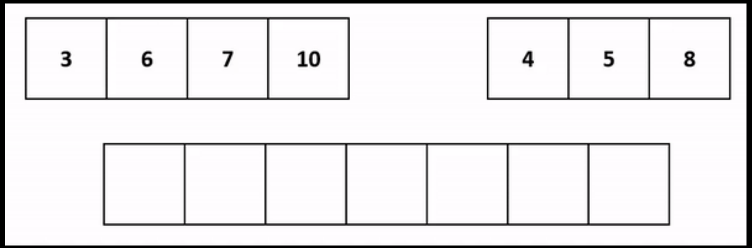


תרגול שעבר

- זמני ריצה, אסימפטוטיקה וסימונים Ω,Θ,O
- מיון בועות Bubble Sort
- מיון בחירה Selection Sort
- מיון הכנסה Insertion Sort
- Merge



```java
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 A **ממויין** בגודל n

ומערך B **ממויין** בגודל m

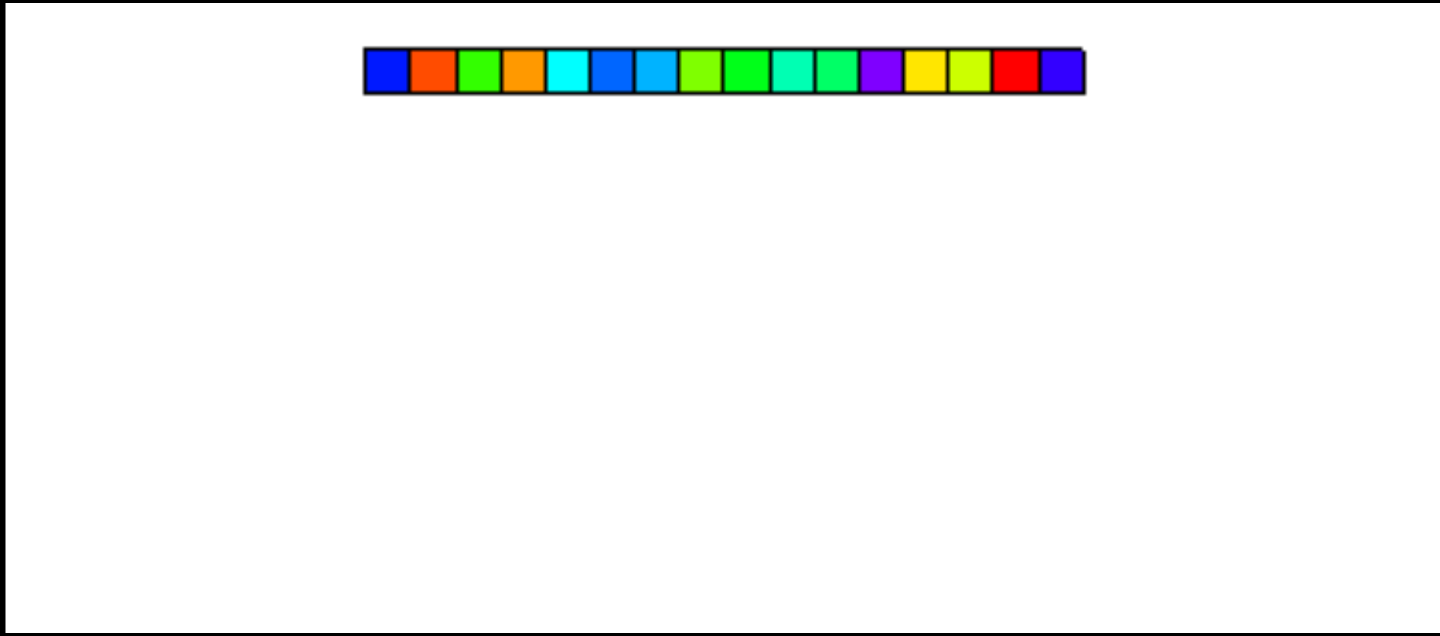
ניתן למיין למערך חדש C בגודל $|A|+|B|$ **ממויין** בסיבוכיות

$$\Theta(n+m)$$

צבי מינץ

תרגול שעבר

- זמני ריצה, אסימפטוטיקה וסימונים O,Θ,Ω



- 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
- מיון בועות Bubble Sort
- מיון בחירה Selection Sort
- מיון הכנסה Insertion Sort
- Merge
- מיון מיזוג Merge Sort

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

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

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

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

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

$$= n \cdot T(1) + \log 2(n) \cdot n = \Theta(n \cdot \log_2(n))$$

תרגול שעבר

- זמני ריצה, אסימפטוטיקה וסימונים Ω, Θ, O
- מיון בועות Bubble Sort
- מיון בחירה Selection Sort
- מיון הכנסה Insertion Sort
- Merge
- מיון מיזוג Merge Sort
- חיפוש לינארי

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
- חיפוש לינארי
- חיפוש בינארי

מיון מהיר - Quick Sort

partition

<u>מטרה:</u> בהינתן מערך A ואיבר x
לשים את x במקום הנכון **<u>במערך ממוין</u>**
ואת כל האיברים שקטנים מ-x לפני x
ואת כל האיברים שגדולים מ-x אחרי x

כל זה אמור להתבצע בזמן לינארי

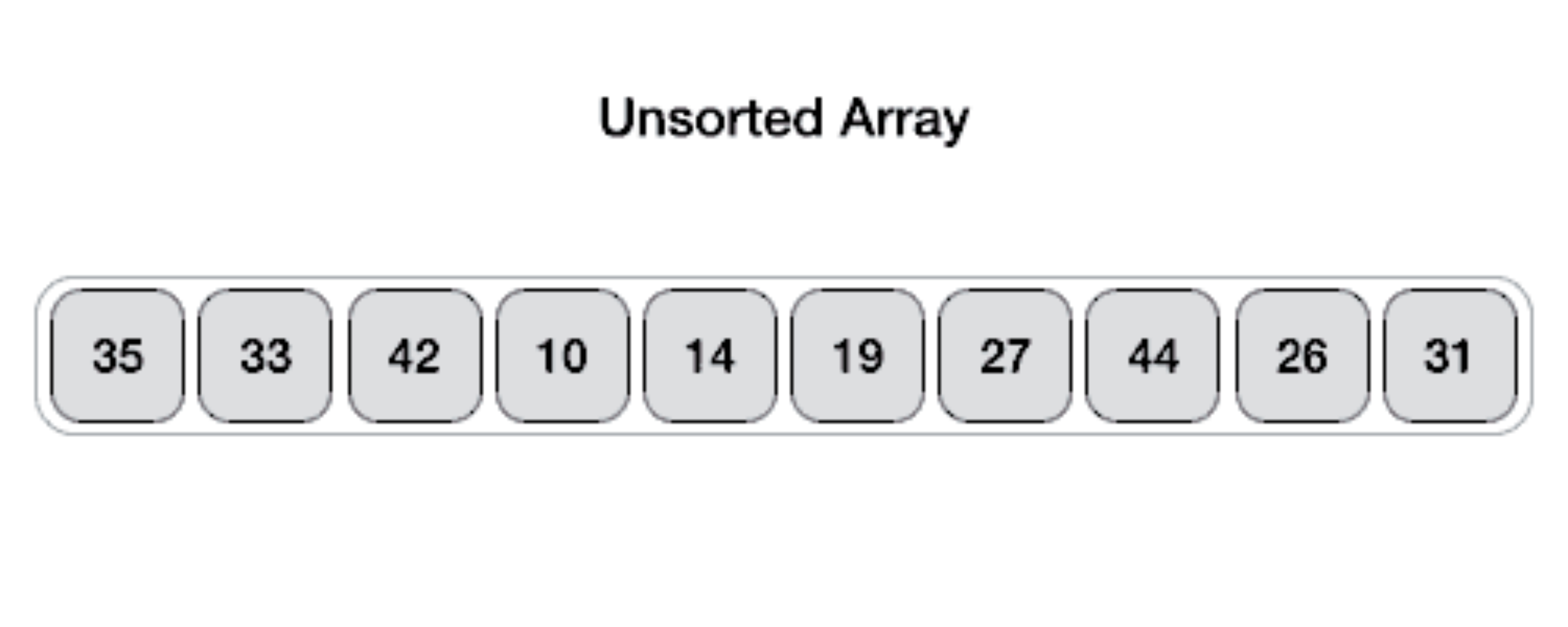
מיון מהיר - Quick Sort

partition

1. Pick a pivot point How?
2. Repeat **while** low ≤ high:
3. low ← the <u>**first**</u> element that is ≥ than pivot
4. high ← the <u>**first**</u> element that is < than pivot
5. Swap the high with low
6. Swap high with pivot

מיון מהיר - Quick Sort

partition



מיון מהיר - Quick Sort

```java
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));
}
```

```java
public static void QuickSort(int[] arr)
{
    QuickSort(arr,0,arr.length-1);
}
```

```java
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);
    }
}
```

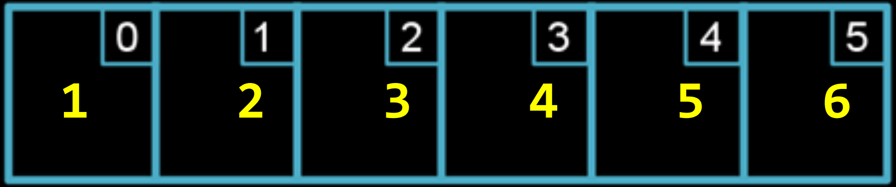
דוגמת ריצה של האלגוריתם

https://www.youtube.com/watch?v=SLauY6PpjW4

```
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?

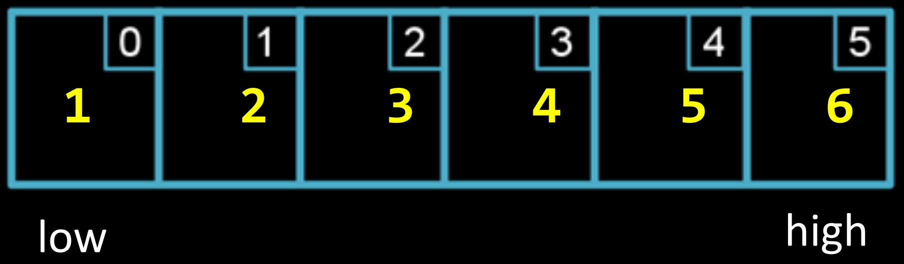



```
// Solution to last week question
private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}
```

```java
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?

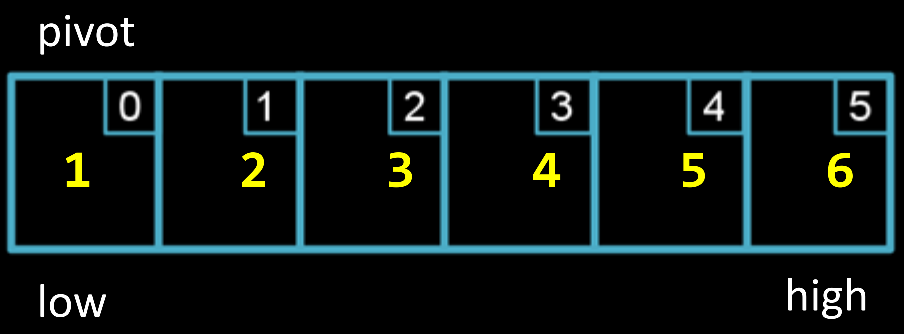



```java
// Solution to last week question
private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}
```

```
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?

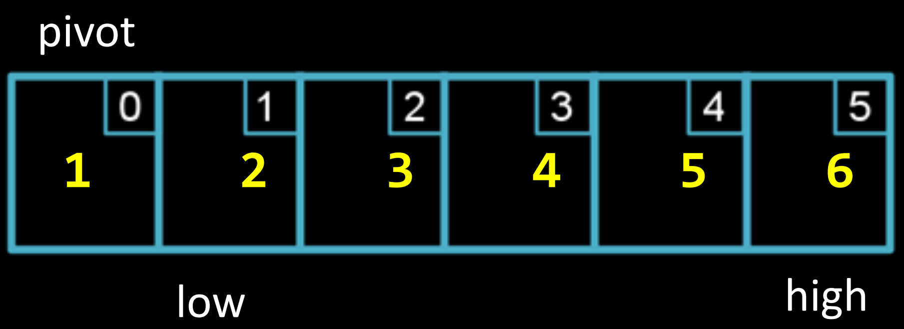



```
// Solution to last week question
private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}
```

```java
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?



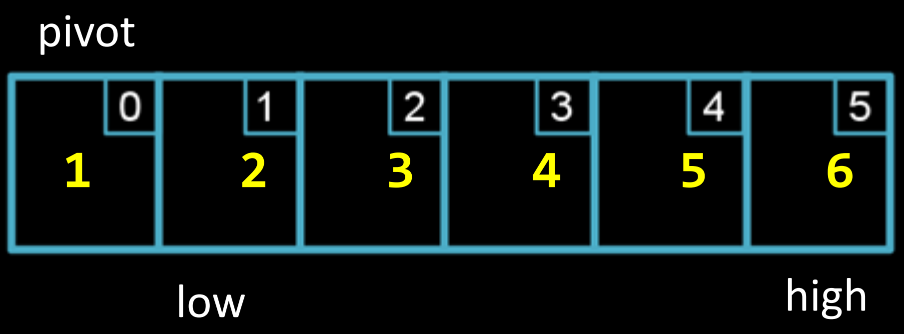

```java
// Solution to last week question
private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}
```

צבי מינץ

```java
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?

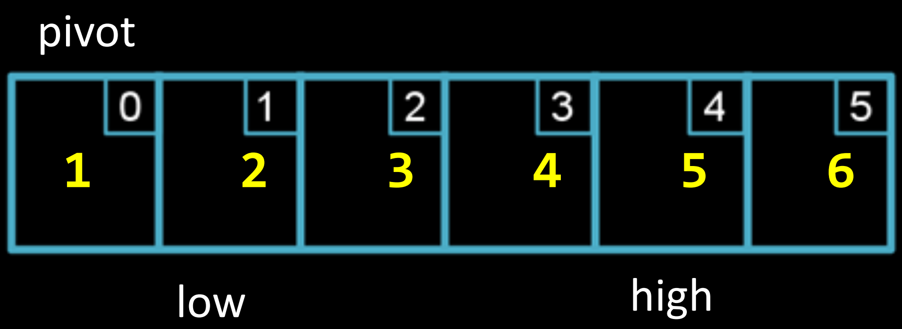



```java
// Solution to last week question
private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}
```

```
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];
 }
```

```java
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?

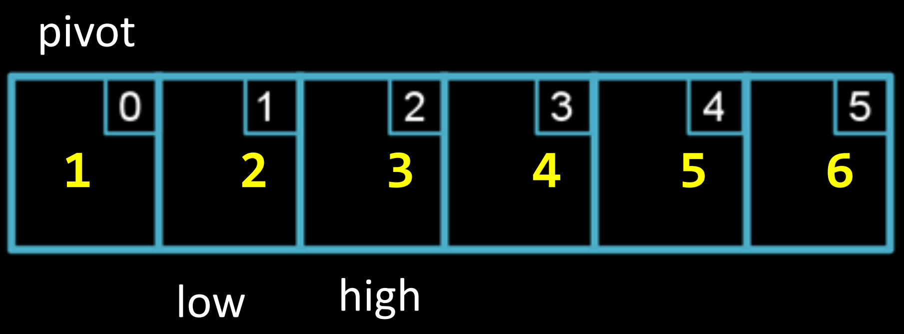

```java
// Solution to last week question
private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}
```

צבי מינץ

```java
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?

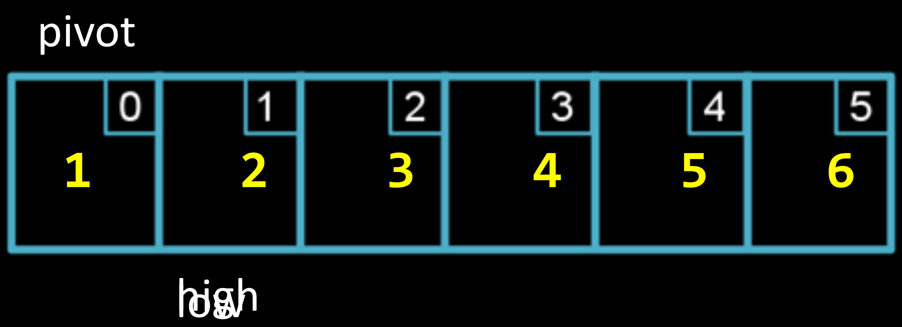



```java
// Solution to last week question
private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}
```

```
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?

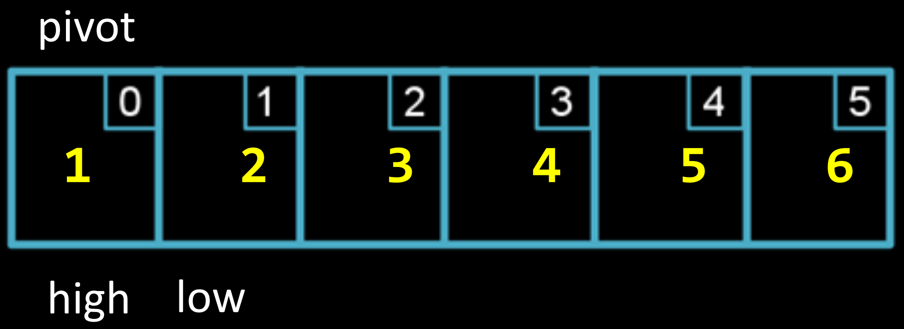



```
// Solution to last week question
private static void swap(int[] arr,int i, int j)
{
    if(i == j) return;
    arr[i] = arr[i] + arr[j];
    arr[j] = arr[i] - arr[j];
    arr[i] = arr[i] - arr[j];
}
```

```
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?



$\Theta(n)$

```
// 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];
}
```

צבי מינץ

מיון מהיר - Quick Sort

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.

```java
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;
}
```

```java
// 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];
}
```

```java
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: $T(n) = 2T(\frac{n}{2}) + \Theta(n) \Rightarrow \Theta(nlogn)$

```
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);
    }
}
```

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

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

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

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).

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

<u>פלט</u>: 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++;
        }
    }
}
```

מיון מבוסס השוואות

Comparison Sort

מיון מבוסס השוואות

Comparison Sort

מיון מבוסס השוואות מסדר אלמנטים במערך ע"י השוואות, <u>בדרך כלל</u> ע"י האופרטור $\leq$

מיונים מבוססי השוואות

- Bubble Sort מיון בועות
- Selection Sort מיון בחירה
- Insertion Sort מיון הכנסה
- Merge Sort מיון מיזוג
- Quick Sort מיון מהיר

...

מיונים לא מבוססי השוואות

- Counting Sort
- Radix Sort

...

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות פעולות השוואה במקרה הגרוע על-מנת לבצעם.

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

$$\Omega(n \cdot \log(n))$$

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

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות פעולות השוואה במקרה הגרוע על-מנת לבצעם.

עץ החלטה

עץ החלטה הוא **<u>מודל</u>** חיזוי בתחומי הסטטיסטיקה,

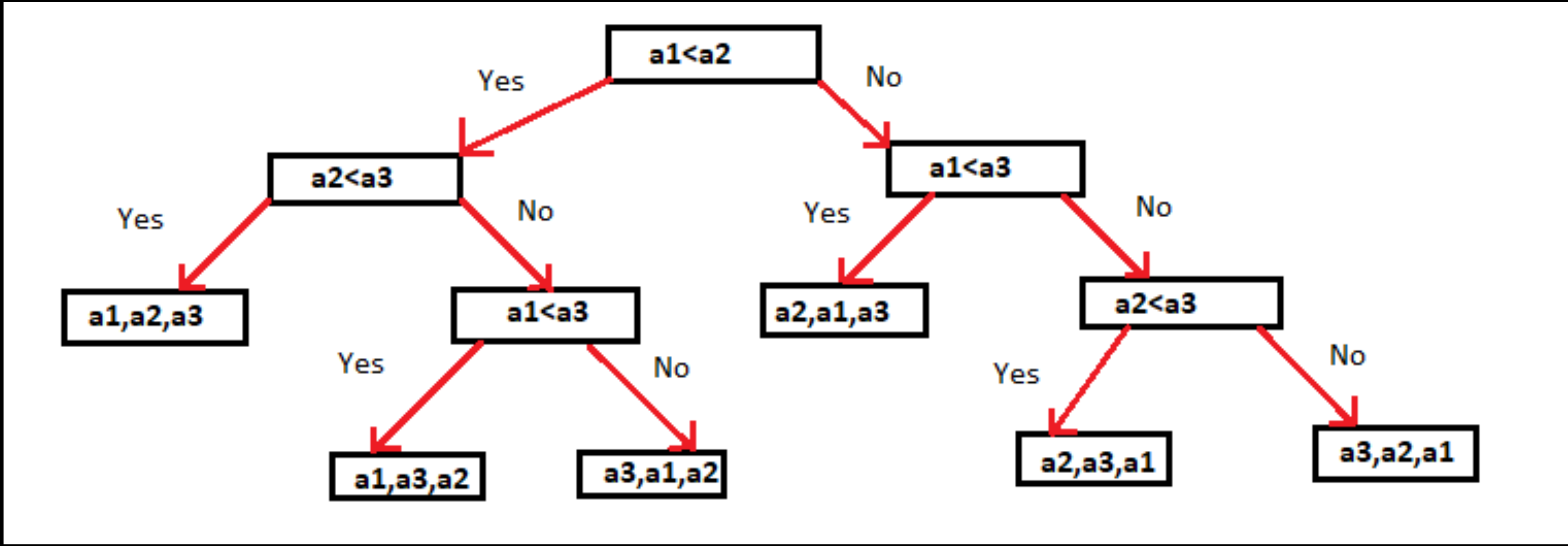
עץ החלטה הוא עץ בינארי מלא המורכב מצמתי החלטה שבכל אחד מהם נבדק תנאי מסוים על מאפיין מסוים ועלים המכילים את **הערך החזוי** עבור התצפית המתאימה למסלול שמוביל אליהם בעץ.

- **<u>עץ בינארי</u>** כאשר כל קודקוד פנימי מתוייג ע"י תווית $a_i \leq a_j$
- ההפעלה של האלגוריתם תואמת את המסלול שורש-עלה
- כל עלה מחזיק את התוצאה של החישוב

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות פעולות השוואה במקרה הגרוע על-מנת לבצעם.

עץ החלטה

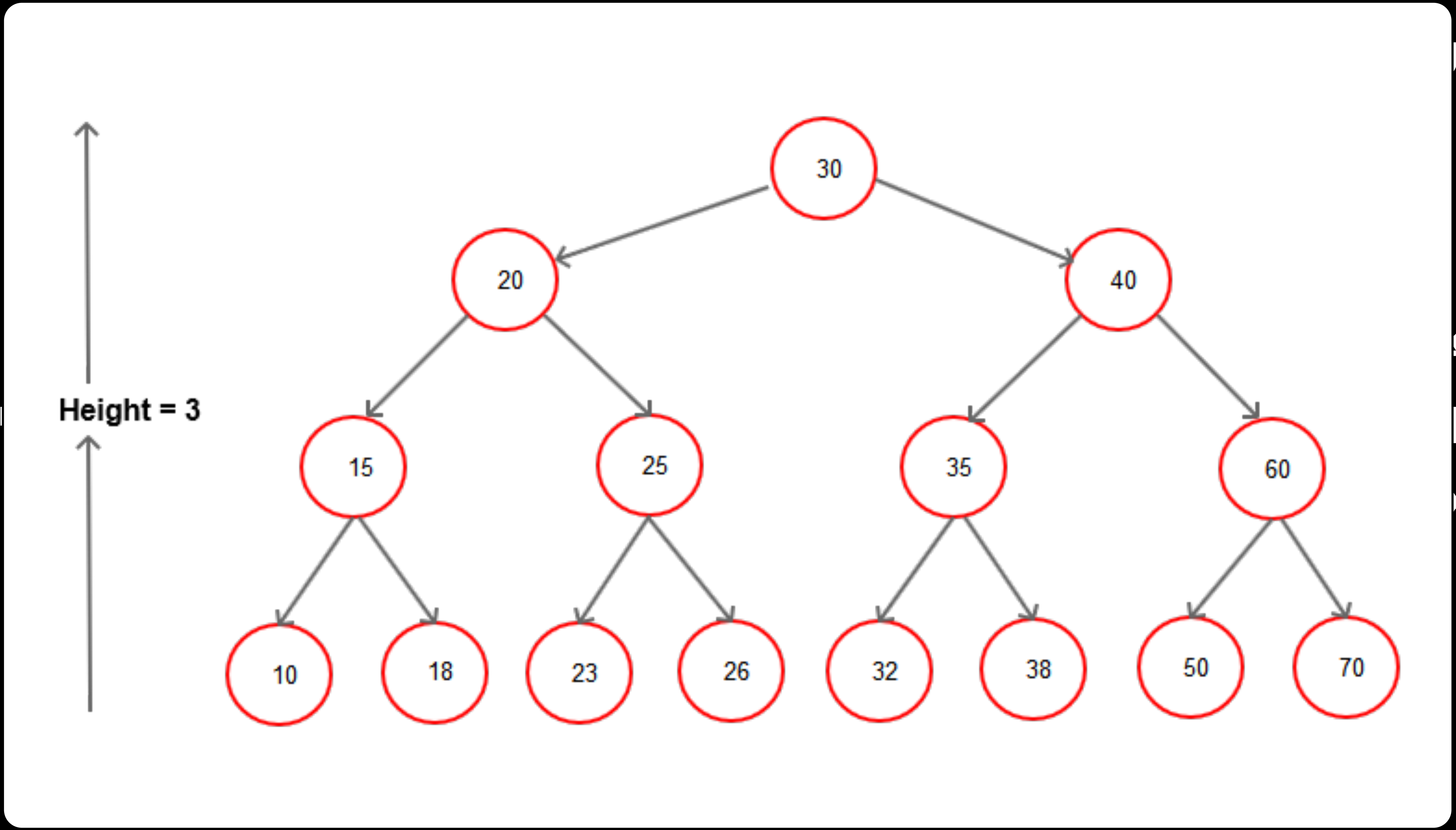
נרצה למיין את הקבוצה $\{a_1,a_2,a_3\}$



כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון <u>**המבוססים על פעולת השוואה**</u> דורשים לפחות $\Omega(n \cdot \log(n))$

פ



הוכחה

- תהי קבוצ
- חייב להיו ... ברים)
- בעע בינא

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

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

כמות העלים בפועל

$$2^h \ge n! \quad \Rightarrow \quad h \ge \log_2(n!)$$

כמות העלים בעץ בינארי לכל היותר

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

Stirling's approximation

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

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

$$= \log\left(\left(\frac{n}{e}\right)^n\right) + \log\left((2\pi n)^{\frac{1}{2}}\right)$$

$$= n \cdot \log\left(\frac{n}{e}\right) + \frac{1}{2} \cdot \log(2\pi n)$$

$$= n \cdot \log n - n \cdot \log_2 e + \frac{1}{2} \cdot \log(2\pi n)$$

$$= \Omega\,(\,n \cdot \log n\,)$$

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.

מיון מנייה - Counting Sort

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

מיון מנייה - Counting Sort

A מערך קלט בגודל n

B פלט האלגוריתם – מערך ממויין בגודל n

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

<u>Counting-Sort (A, B, k)</u>

max ← max value in A

min ← max value in A

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

for i ← 0 to n

    C[A[i] - min] ← C[A[i] - min] + 1

כעת C[i] מחזיק בתוכו את כמות החזרות של A[i] עבור min=0

for i ← 1 to k

    C[i] ← C[i] + C[i-1] // C[i]

כעת C[i] מחזיק בתוכו את כמות האיברים אשר קטנים שווים ל- A[i] עבור min=0

for j ← n downto 1

    B[C[A[j] - min]] ← A[j]

    C[A[j] - min] ← C[A[j] - min] - 1

return B

מיון מנייה - Counting Sort

A מערך קלט בגודל n

B פלט האלגוריתם – מערך ממויין בגודל n

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

<u>Counting-Sort (A, B, k)</u>

max ← max value in A

min ← max value in A

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

for i ← 0 to n

C[A[i] - min] ← C[A[i] - min] + 1

כעת C[i] מחזיק בתוכו את כמות החזרות של A[i] עבור min=0

for i ← 1 to k

C[i] ← C[i] + C[i-1] // C[i]

כעת C[i] מחזיק בתוכו את כמות האיברים אשר קטנים שווים ל- A[i] עבור min=0

for j ← n downto 1

איבר x במערך המקורי

B[C[A[j] - min]] ← A[j]

C[A[j] - min] ← C[A[j] - min] - 1

return B

מיון מנייה - Counting Sort

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 ← max value in A
min ← max value in A
Initialize C[max – min + 1] s.t C[i] = 0 for all i

for i ← 0 to n
    C[A[i] - min] ← C[A[i] - min] + 1

כעת C[i] מחזיק בתוכו את כמות החזרות של A[i] עבור min=0

for i ← 1 to k
    C[i] ← C[i] + C[i-1] // C[i]

כעת C[i] מחזיק בתוכו את כמות האיברים אשר קטנים שווים ל- A[i] עבור min=0

כמות האיברים אשר קטנים **<u>שווים</u>** ל- **x**

for j ← n downto 1
    B[C[A[j] - min]] ← A[j]
    C[A[j] - min] ← C[A[j] - min] - 1

return B

מיון מנייה - Counting Sort

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 ← max value in A
min ← max value in A
Initialize C[ max – min + 1 ] s.t C[i] = 0 for all i

for i ← 0 to n
      C[ A[i] - min ] ← C[A[i] - min] + 1

for i ← 1 to k
      C[i] ← C[i] + C[i-1] // C[i]

for j ← n downto 1
      B[C[A[j] - min]] ← A[j]
      C[A[j] - min] ← C[A[j] - min] - 1

return B
```

כעת C[i] מחזיק בתוכו את כמות החזרות של A[i] עבור min=0

כעת C[i] מחזיק בתוכו את כמות האיברים אשר קטנים שווים ל- A[i] עבור min=0

התא של
כמות האיברים אשר קטנים **שווים** ל- x
תהיה שווה ל-x

צבי מינץ

מיון מנייה - Counting Sort

A מערך קלט בגודל n

B פלט האלגוריתם – מערך ממויין בגודל n

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

<u>Counting-Sort (A, B, k)</u>

max ← max value in A
min ← max value in A
Initialize C[max – min + 1] s.t C[i] = 0 for all i

for i ← 0 to n
    C[A[i] - min] ← C[A[i] - min] + 1

כעת C[i] מחזיק בתוכו את כמות החזרות של A[i] עבור min=0

for i ← 1 to k
    C[i] ← C[i] + C[i-1] // C[i]

כעת C[i] מחזיק בתוכו את כמות האיברים אשר קטנים שווים ל- A[i] עבור min=0

for j ← n downto 1
    B[C[A[j]]] ← A[j]
    C[A[j]] ← C[A[j]] - 1

return B

$$O(n + k)$$

מיון מנייה - Counting Sort

A מערך קלט בגודל n

B פלט האלגוריתם – מערך ממויין בגודל n

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

<u>Counting-Sort (A, B, k)</u>

max ← max value in A

min ← max value in A

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

for i ← 0 to n

C[A[i] - min] ← C[A[i] - min] + 1

כעת C[i] מחזיק בתוכו את כמות החזרות של A[i] עבור min=0

for i ← 1 to k

C[i] ← C[i] + C[i-1] // C[i]

כעת C[i] מחזיק בתוכו את כמות האיברים אשר קטנים שווים ל- A[i] עבור min=0

for j ← n downto 1

B[C[A[j]]] ← A[j]

C[A[j]] ← C[A[j]] - 1

return B

$$O(n+k)$$

צבי מינץ

מיון מנייה - Counting Sort

<u>אבחנה:</u>

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.
(הסדר היחסי בין איברים זהים בערכם נשמר לאחר המיון)

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.

או לחלופין ...Selection Sort, QuickSort,
דוגמא {4,2,3,4,1}

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

לסיכום

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

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות פעולות השוואה במקרה הגרוע על-מנת לבצעם.

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות פעולות השוואה במקרה הגרוע על-מנת לבצעם.

לסיכום

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

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות פעולות השוואה במקרה הגרוע על-מנת לבצעם.

לסיכום

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

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצע.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצע.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצע.

לסיכום

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

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

לסיכום

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

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

לסיכום

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

כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.
כל אלגוריתמי המיון המבוססים על פעולת השוואה דורשים לפחות
פעולות השוואה במקרה הגרוע על-מנת לבצעם.

לסיכום

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

אוניברסיטת אריאל בשומרון

עבודה עצמית

PDF

במודל

צבי מינץ