Сортиране

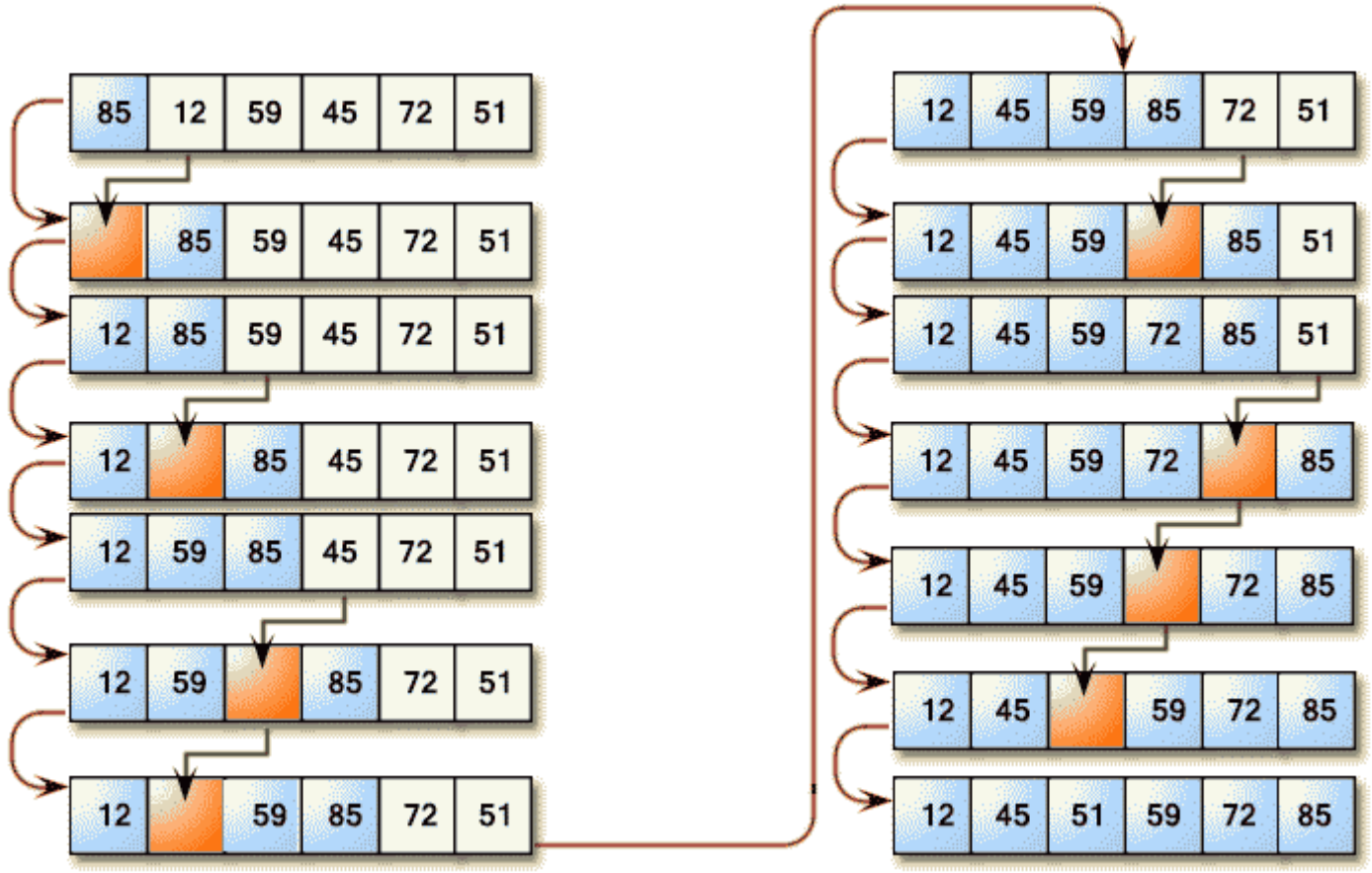
Защо сортирането е важно?

- Визуализация на данни;
- Двоично търсене ($O(n)$ vs. $O(\log_2 n)$);
- Най-близки елементи ($O(n^2)$ vs. $O(n\log_2 n)$);
- Уникалност на елементите ($O(n^2)$ vs. $O(n\log_2 n)$);
- Изчисление на мода и медиана.

Стабилност

- Стабилно сортиране не променя взаимното разположение на елементи с еднакви ключове.

Сортиране чрез вмъкване



Сортиране чрез вмъкване (сравнения)

```
void insertionSort(int[] arr)
{
    int n = arr.Length;
    for (int i = 1; i < n; ++i)
    {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

- $a[1] < a[2] < \cdots < a[n-1]$
- $T_{min}(n) = n - 1$

- $a[1] > a[2] > \cdots > a[n-1]$
- $T_{max}(n) = \frac{n^2+n}{2} - 1$

- $O(n^2)$

Сортиране чрез вмъкване (присвоявания)

```
void insertionSort(int[] arr)
{
    int n = arr.Length;
    for (int i = 1; i < n; ++i)
    {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

- $a[1] < a[2] < \cdots < a[n-1]$
- $T_{min}(n) = 2(n-1)$

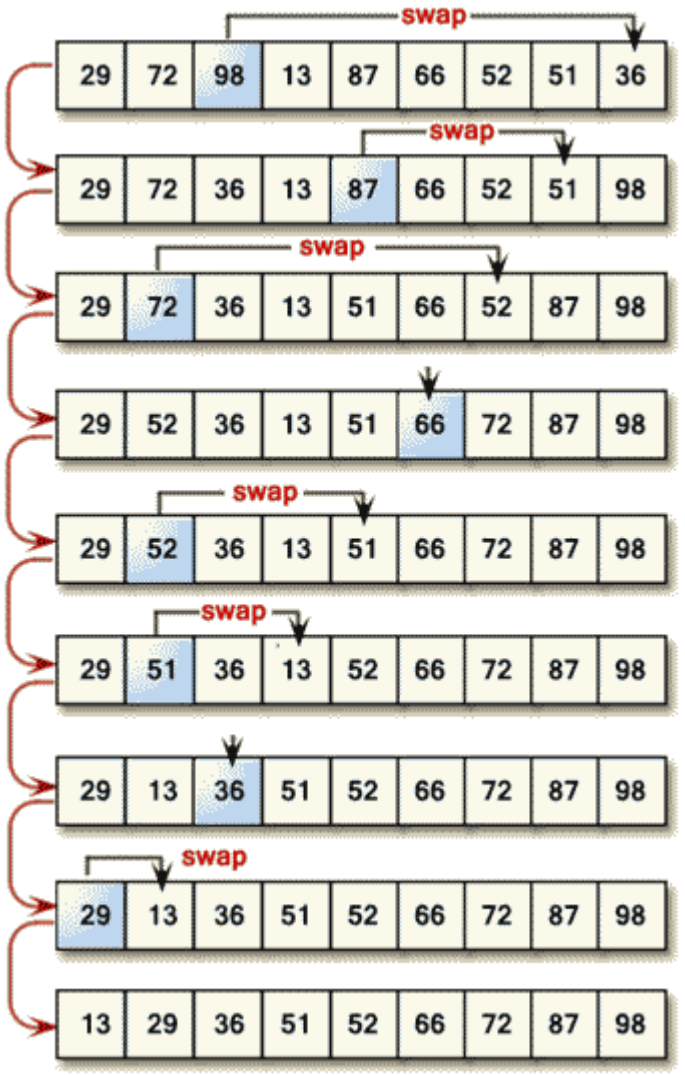
- $a[1] > a[2] > \cdots > a[n-1]$
- $T_{max}(n) = \frac{n^2+3n}{2} - 2$

- $O(n^2)$

Стабилност

- Методът за сортирането чрез вмъкване е стабилен.

Сортиране чрез пряка селекция



Сортиране чрез пряка селекция (сравнения)

```
static void selectionSort(int[] arr)
{
    int n = arr.Length;

    for (int i = n - 1; i > 0; i--)
    {
        int maxIdx = i;
        for (int j = i - 1; j >= 0; j--)
            if (arr[j] > arr[maxIdx])
                maxIdx = j;

        int temp = arr[maxIdx];
        arr[maxIdx] = arr[i];
        arr[i] = temp;
    }
}
```

- $a[1] < a[2] < \cdots < a[n-1]$
- $T_{min}(n) = \frac{n(n-1)}{2}$

- $a[1] > a[2] > \cdots > a[n-1]$
- $T_{max}(n) = \frac{n(n-1)}{2}$

- $O(n^2)$

Сортиране чрез пряка селекция (присвоявания)

```
static void selectionSort(int[] arr)
{
    int n = arr.Length;

    for (int i = n - 1; i > 0; i--)
    {
        int maxIdx = i;
        for (int j = i - 1; j >= 0; j--)
            if (arr[j] > arr[maxIdx])
                maxIdx = j;

        int temp = arr[maxIdx];
        arr[maxIdx] = arr[i];
        arr[i] = temp;
    }
}
```

- $a[1] < a[2] < \cdots < a[n-1]$
- $T_{min}(n) = 3(n-1)$

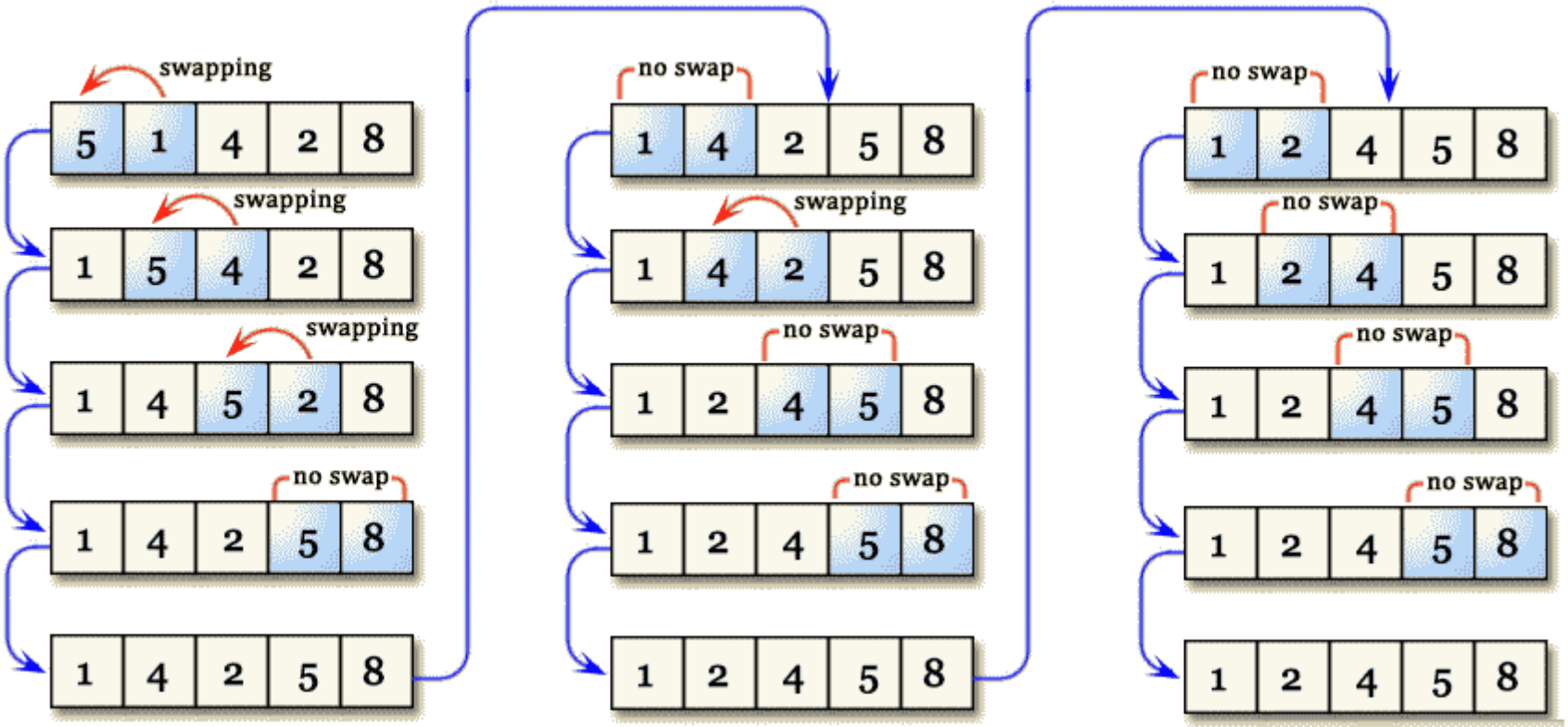
- $a[1] > a[2] > \cdots > a[n-1]$
- $T_{max}(n) = \frac{n^2}{4} + 3(n-1)$

- $O(n^2)$

Стабилност

- Методът за сортирането чрез пряка селекция не е стабилен.
- Пр: 2, 3, 4, 2, 1 -> 1, 3, 4, 2, 2 -> 1, 2, 4, 3, 2 -> 1, 2, 2, 3, 4

Сортиране по метод на мехурчето



Сортиране по метод на мехурчето (сравнения)

```
static void bubbleSort(int[] arr)
{
    int n = arr.Length;
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
}
```

- $a[1] < a[2] < \cdots < a[n-1]$
- $T_{min}(n) = \frac{n(n-1)}{2}$

- $a[1] > a[2] > \cdots > a[n-1]$
- $T_{max}(n) = \frac{n(n-1)}{2}$

- $O(n^2)$

Сортиране по метод на мехурчето (присвоявания)

```
static void bubbleSort(int[] arr)
{
    int n = arr.Length;
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
}
```

- $a[1] < a[2] < \cdots < a[n-1]$
- $T_{min}(n) = 0$

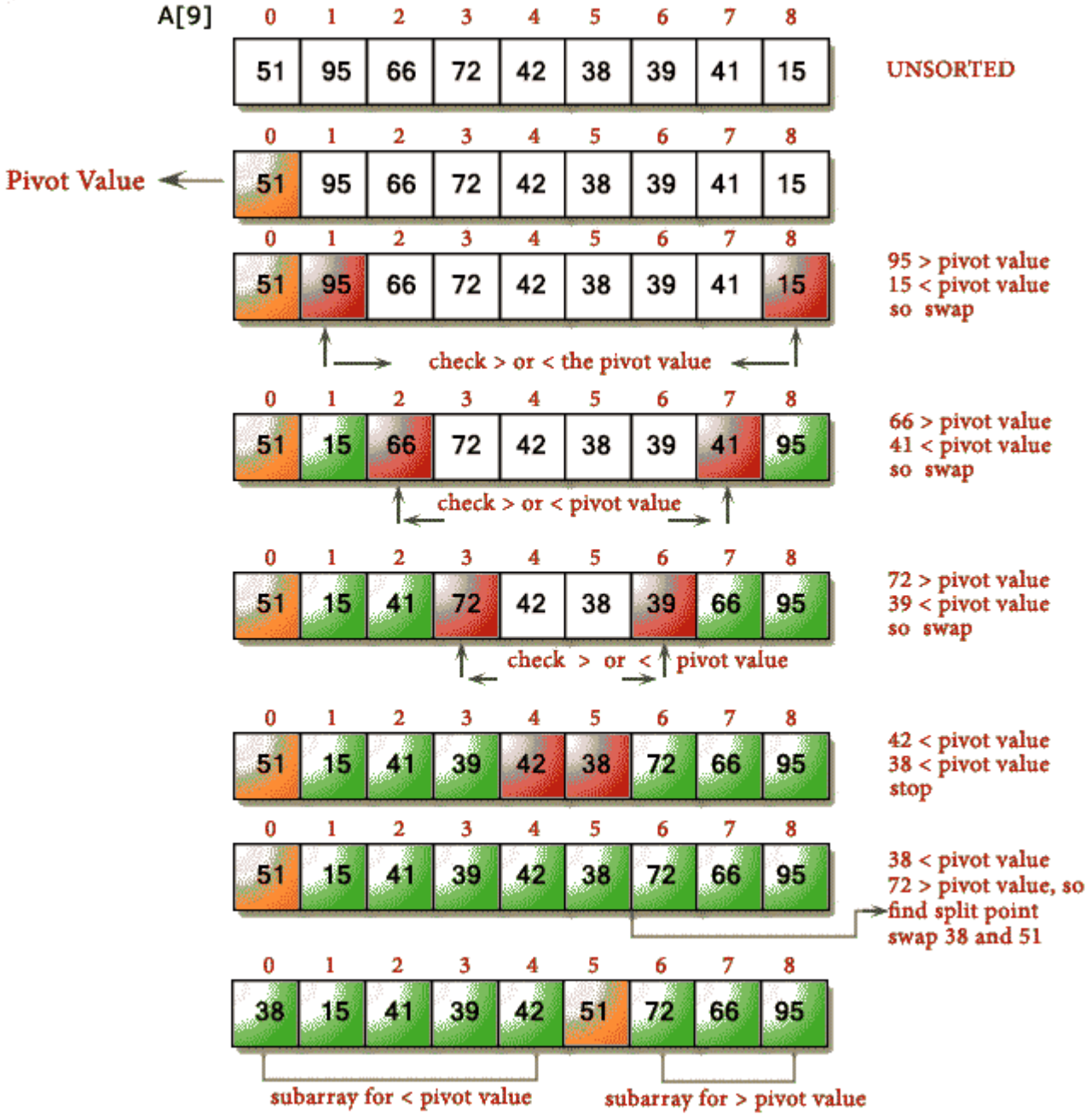
- $a[1] > a[2] > \cdots > a[n-1]$
- $T_{max}(n) = \frac{3(n^2-n)}{2}$

- $O(n^2)$

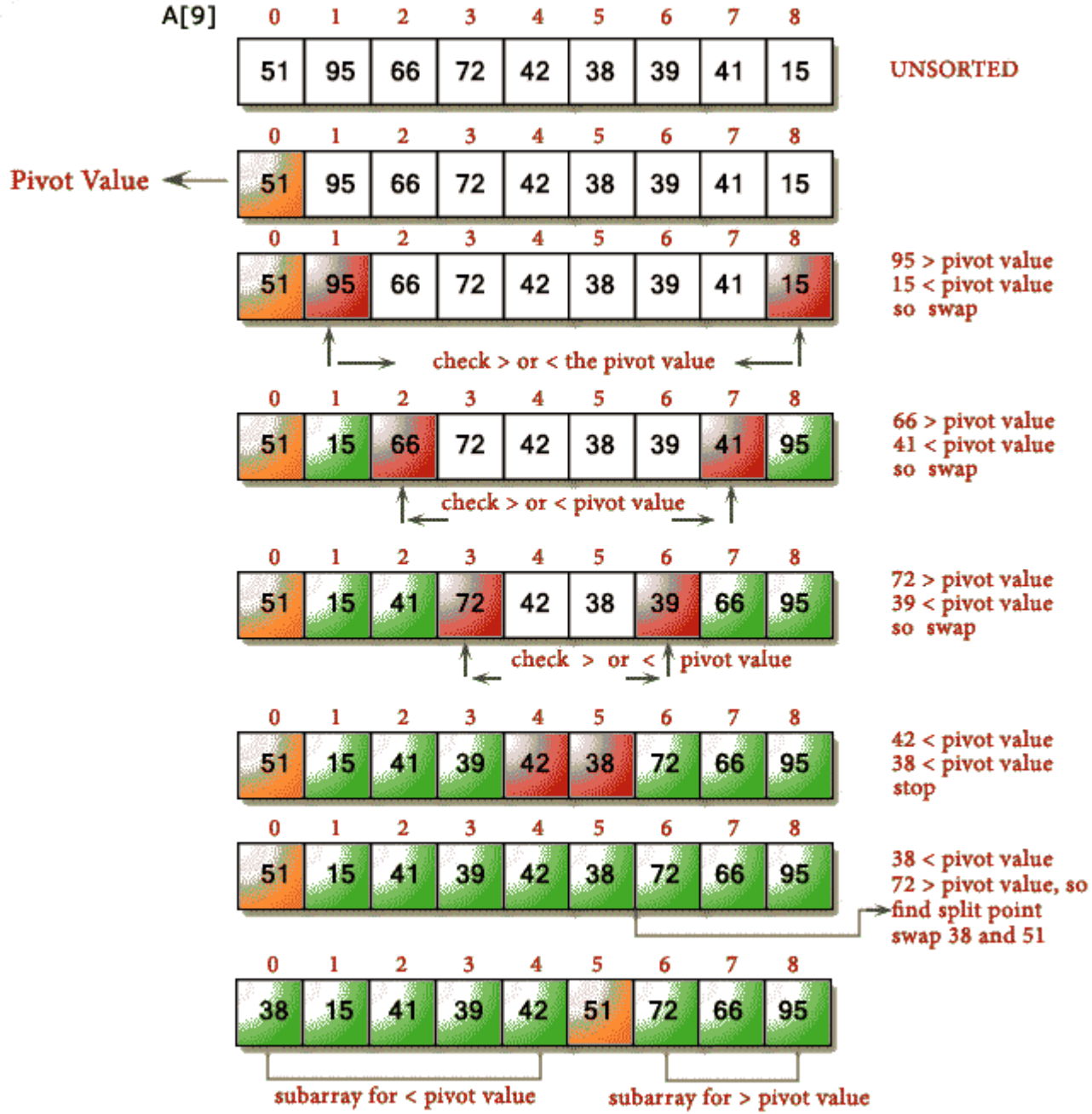
Стабилност

- Сортирането по метод на мехурчето е стабилен.

Бързо сортиране



```
static int Partition(
    int[] arr,
    int l,
    int r)
{
    int temp;
    int pivotIdx = l++;
    int lastIdx = arr.Length - 1;

    while (true)
    {
        while (arr[l] < arr[pivotIdx] && l < lastIdx)
            l++;

        while (arr[r] > arr[pivotIdx] && r > 0)
            r--;

        if (l >= r)
            break;

        if (arr[l] == arr[r])
        {
            l++;
            r--;
        }
        else
        {
            temp = arr[l];
            arr[l] = arr[r];
            arr[r] = temp;
        }
    }

    temp = arr[r];
    arr[r] = arr[pivotIdx];
    arr[pivotIdx] = temp;

    return r;
}
```



Бързо сортиране

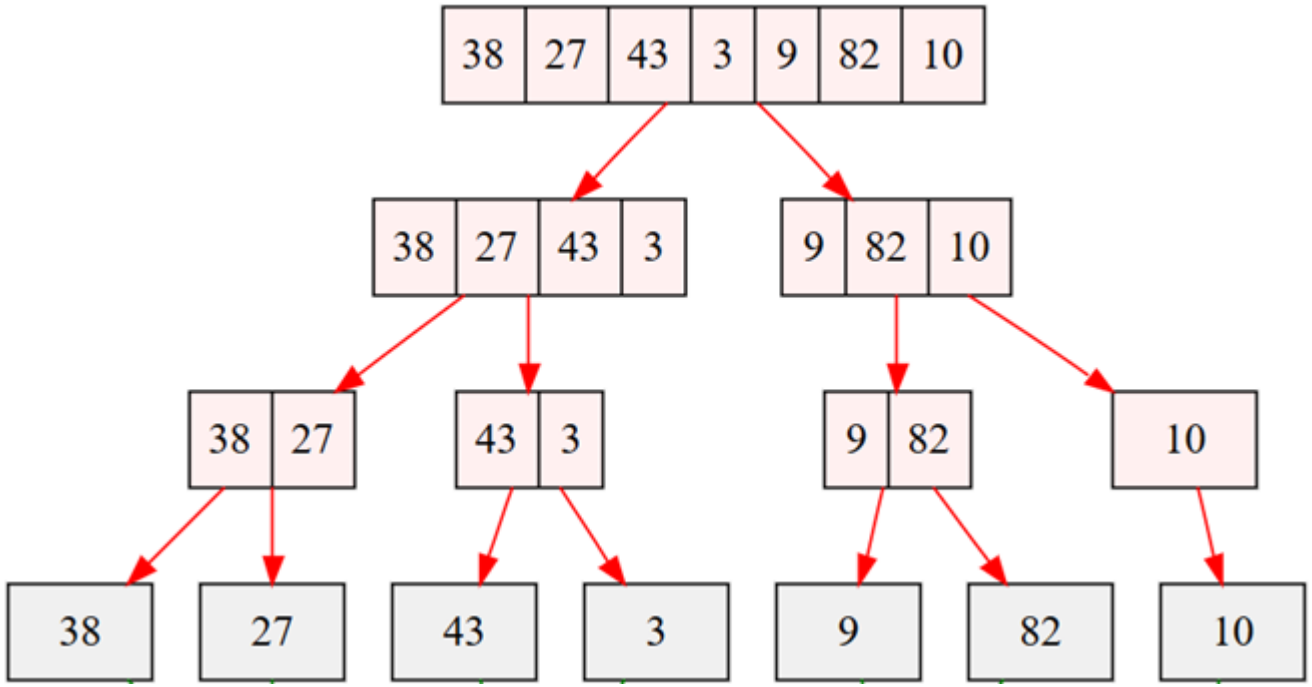
```
static void QuickSort(int[] arr, int l, int r)
{
    if (l < r)
    {
        int pivot = Partition(arr, l, r);

        if (pivot > 1)
            QuickSort(arr, l, pivot - 1);

        if (pivot + 1 < r)
            QuickSort(arr, pivot + 1, r);
    }
}
```

- $O(n^2)$

Стабилност

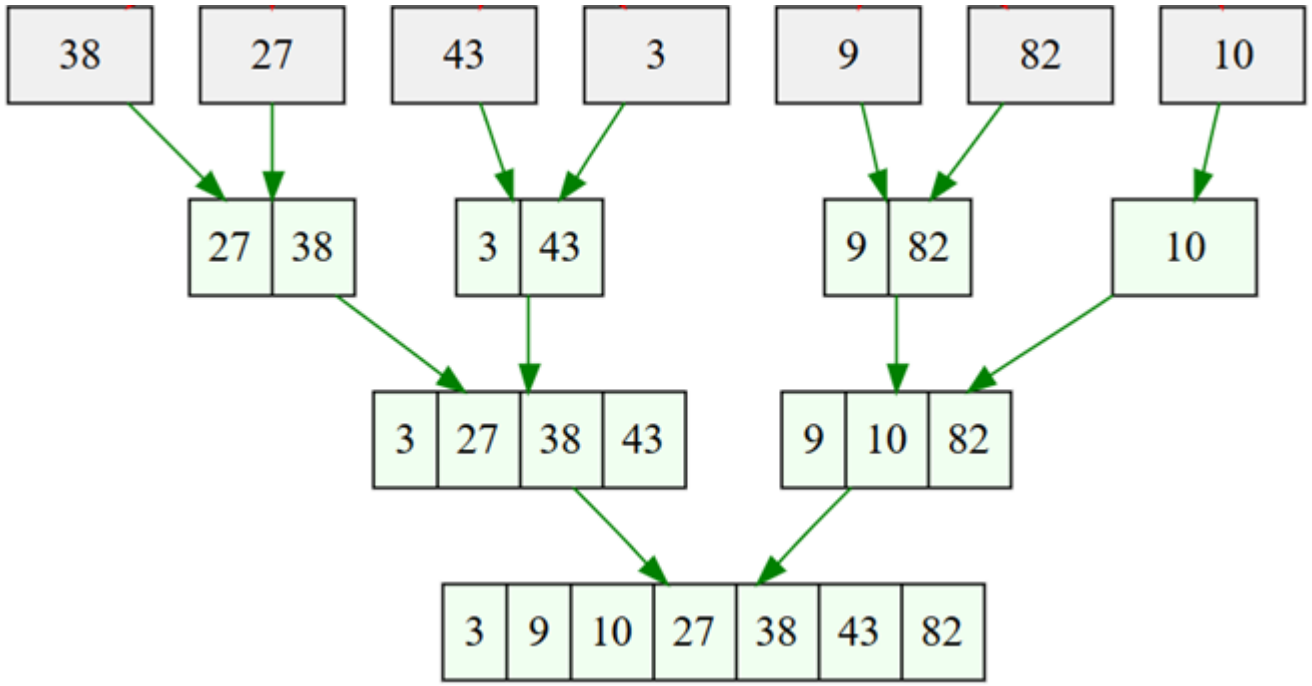
- Методът за бързо сортиране не е стабилен.

Сортиране чрез сливане




```
static void MergeSort(int[] arr, int l, int r)
{
    if (l < r)
    {
        int m = l+(r-l)/2;

        MergeSort(arr, l, m);
        MergeSort(arr, m+1, r);

        Merge(arr, l, m, r);
    }
}
```

Сортиране чрез сливане




```
static void Merge(int[] arr, int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    var L = new int[n1];
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];

    var R = new int[n2];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
            arr[k] = L[i++];
        else
            arr[k] = R[j++];
        k++;
    }

    while (i < n1)
        arr[k++] = L[i++];

    while (j < n2)
        arr[k++] = R[j++];
}
```

Стабилност

- Методът за сортиране чрез сливане е стабилен.

Сортиране чрез сливане

Предимства:

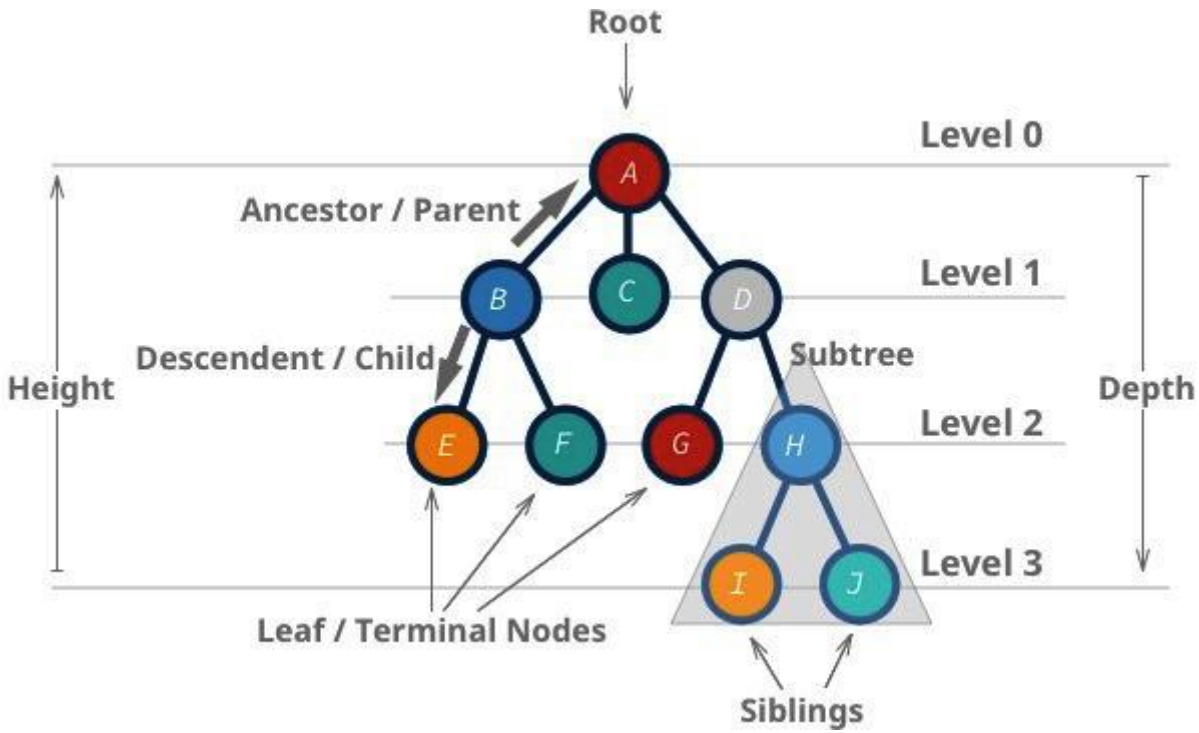
- Удобен за свързан списък
- Външно сортиране

Недостатъци:

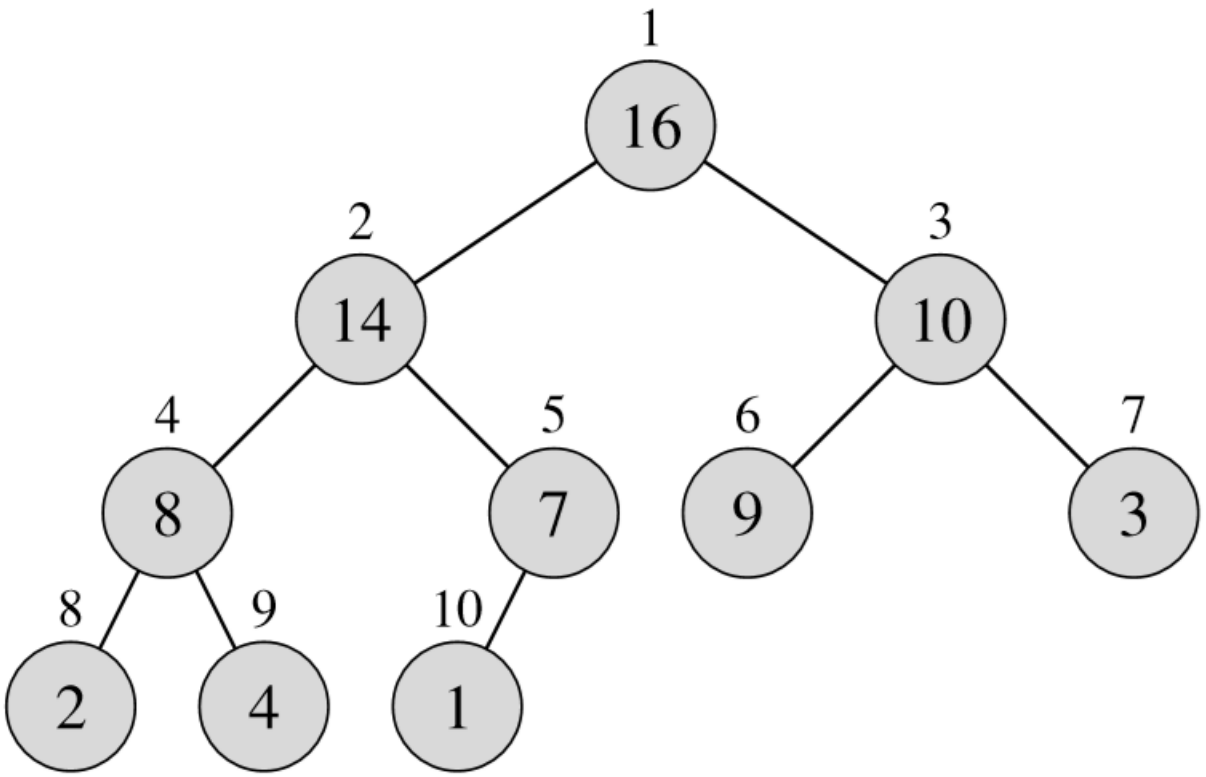
- Рекурсивен
- Изисква допълнителна памет

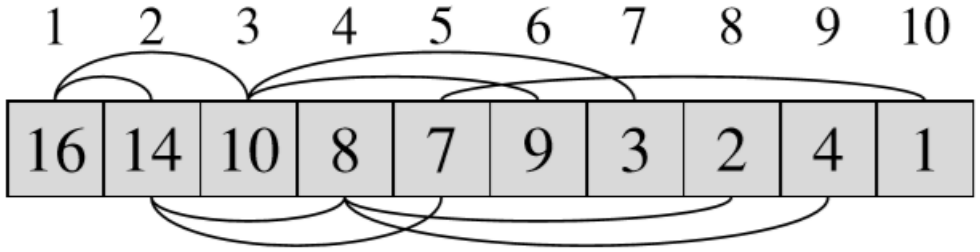
- $O(n \log_2 n)$

Дървета – основни понятия

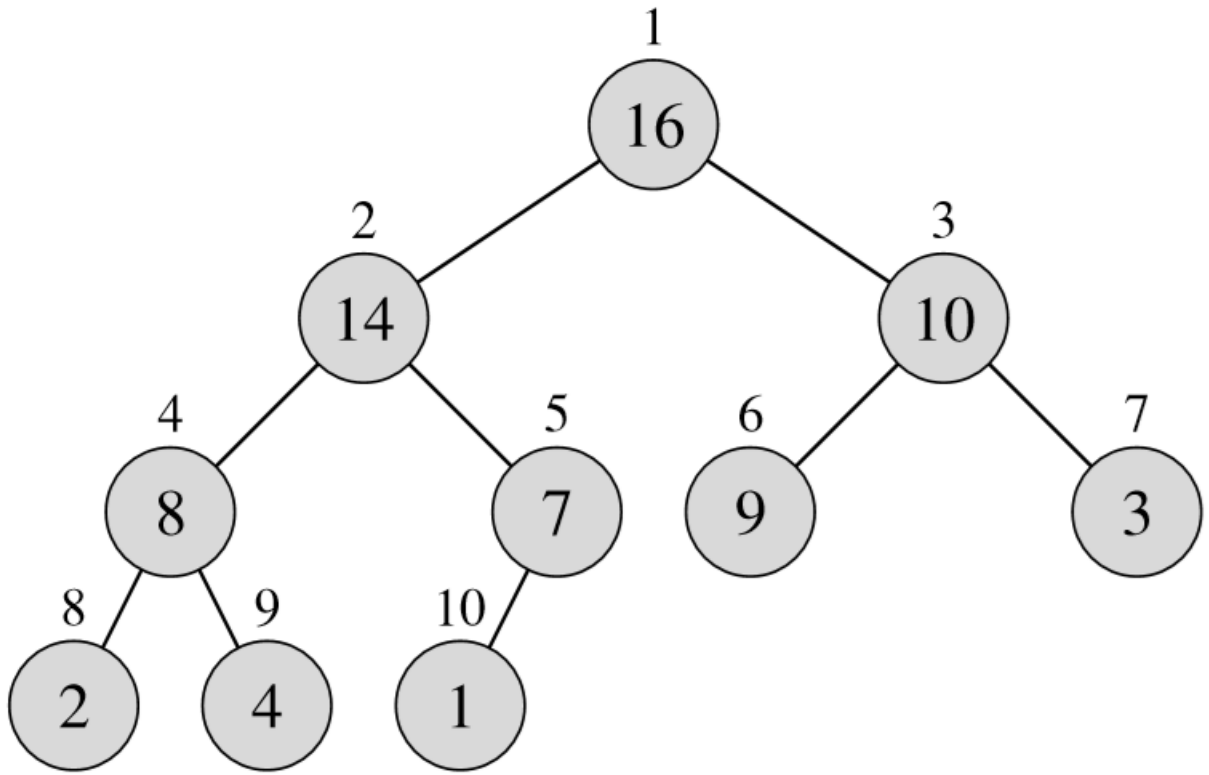


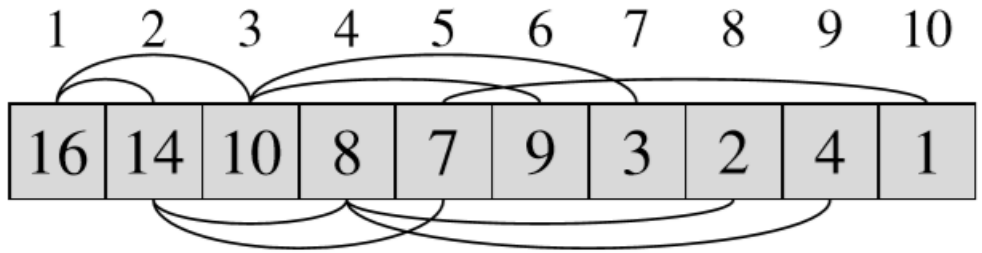
Двоично дърво





Двоично дърво





PARENT(i)
    return $\lfloor i/2 \rfloor$

LEFT(i)
    return $2i$

RIGHT(i)
    return $2i + 1$

Видове двоични дървета

- Низходящо

$$A[\text{PARENT}(i)] \geq A[i]$$

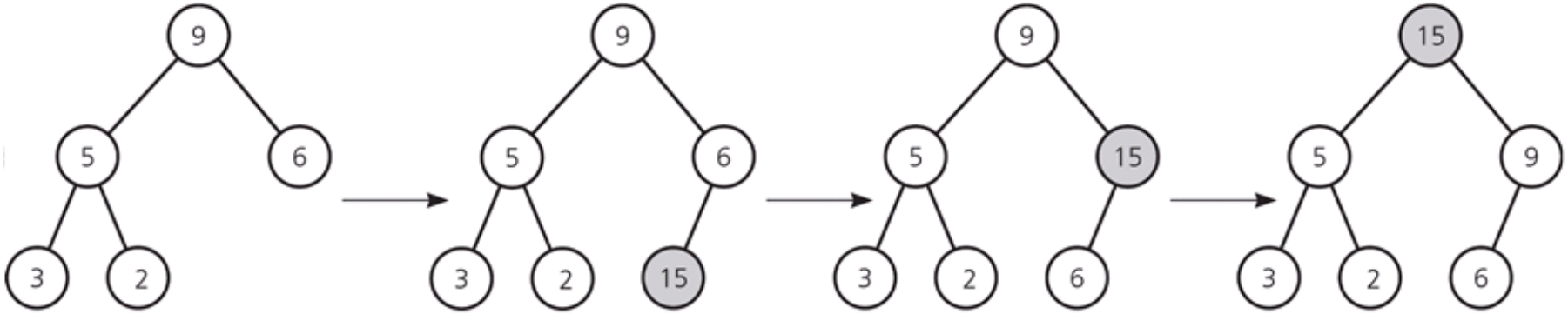
- Възходящо

$$A[\text{PARENT}(i)] \leq A[i]$$

Двоично дърво

- Височина на дървото: $\log_2 n$;
- Пълно двоично дърво

Добавяне на елемент



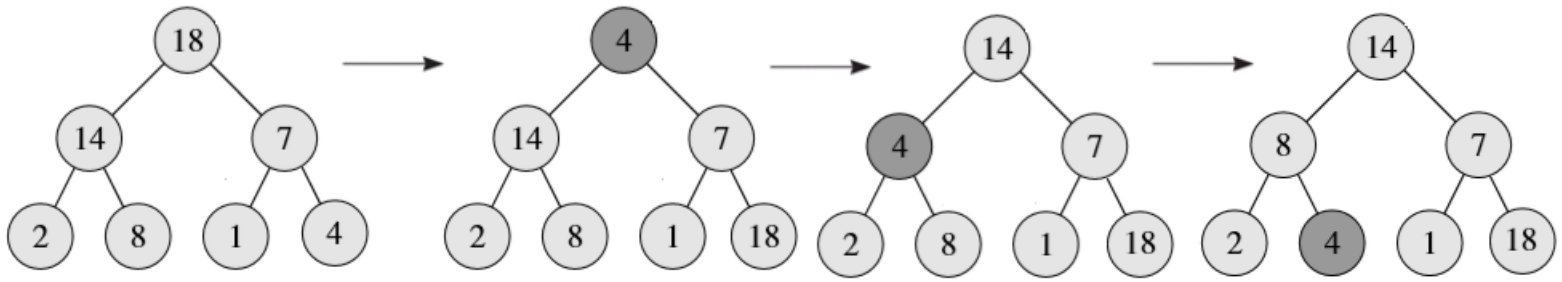
Добавяне на елемент

```
static void Insert(int[] arr, int n, int val)
{
    n = n + 1;
    arr[n - 1] = val;
    InsertRestore(arr, n, n - 1);
}
```

```
static void InsertRestore(int[] arr, int n, int i)
{
    int parent = (i - 1) / 2;

    if (arr[parent] > 0)
    {
        if (arr[i] > arr[parent])
        {
            var temp = arr[i];
            arr[i] = arr[parent];
            arr[parent] = temp;

            InsertRestore(arr, n, parent);
        }
    }
}
```

- $O(\log_2 n)$

Изтриване на елемент



Изтриване на елемент

```
static void Delete(int[] arr, int n)
{
    var root = arr[0];
    var lastElement = arr[n - 1];

    arr[0] = lastElement;
    DeleteRestore(arr, n, 0);
    arr[n - 1] = root;
}
```

- $O(\log_2 n)$

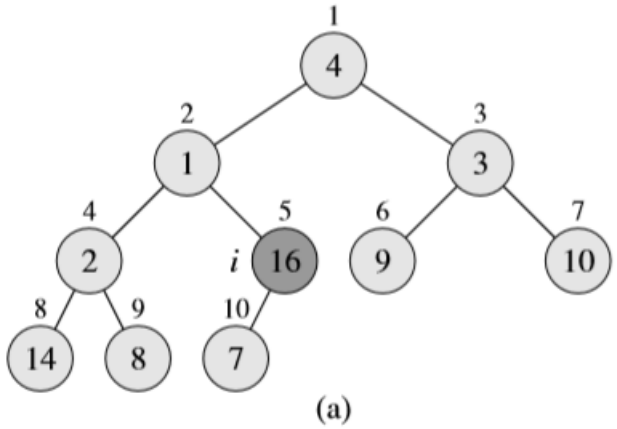
```
static void DeleteRestore(int[] arr, int n, int i)
{
    var largest = i;
    var l = 2 * i + 1;
    var r = 2 * i + 2;

    if (l < n && arr[l] > arr[largest])
        largest = l;

    if (r < n && arr[r] > arr[largest])
        largest = r;

    if (largest != i)
    {
        var temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;

        DeleteRestore(arr, n, largest);
    }
}
```

Създаване

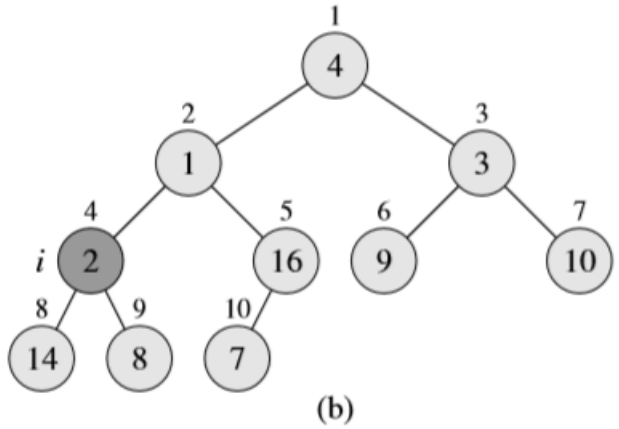
A	4	1	3	2	16	9	10	14	8	7

- За всеки родител от долу

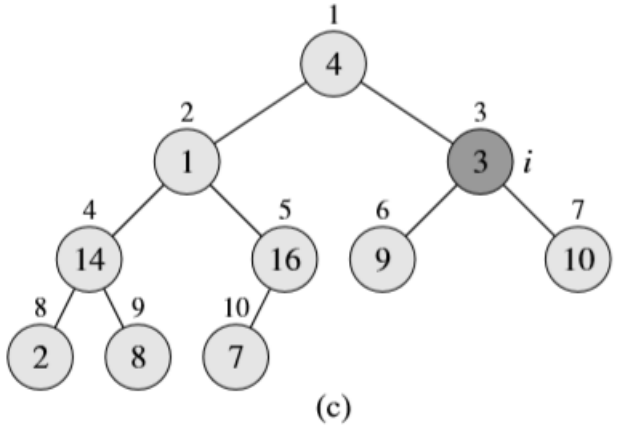
- $O(n)$



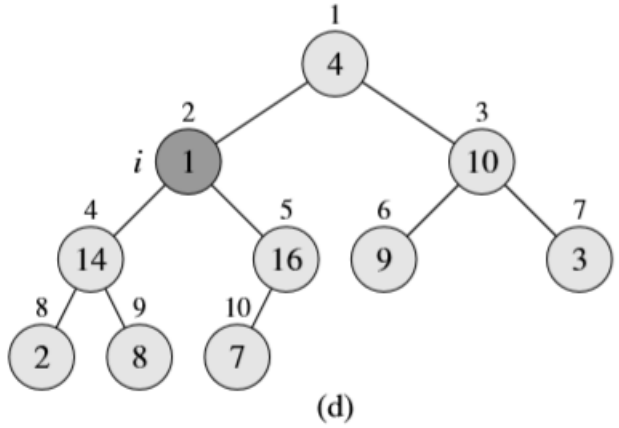
(a)



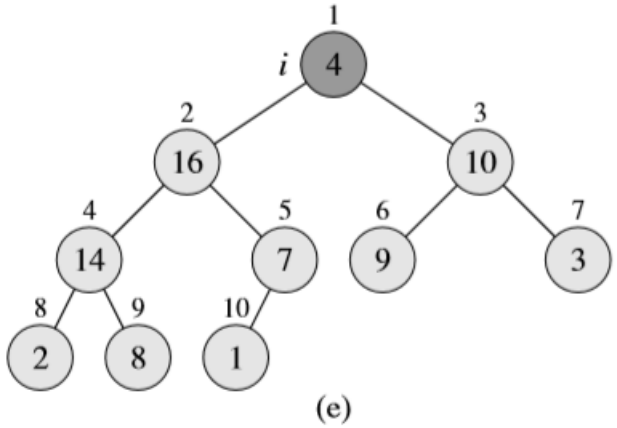
(b)



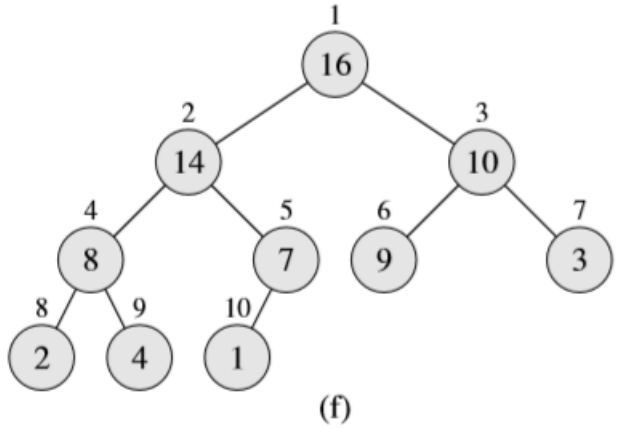
(c)



(d)



(e)

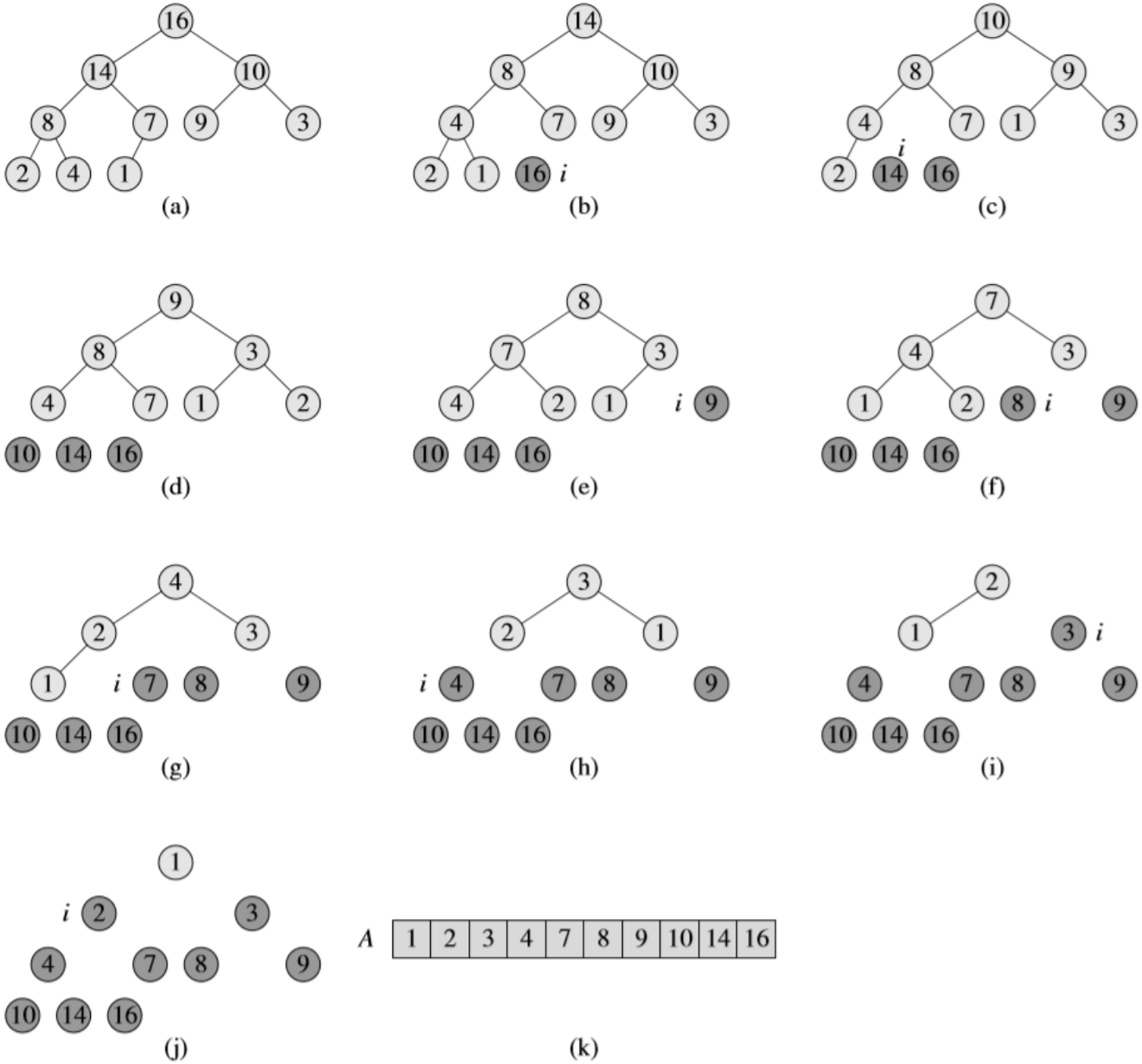


(f)

Сортиране

- За всеки възел без последния: Delete

- $O(n \log_2 n)$



Стабилност

- Методът за пирамидално сортиране не е стабилен.