Recursive Sorting algorithms

1. **Divide and conquer methods**

- **MergeSort**: Mergesort is a divide and conquer algorithm which is divided in two parts, **divide** and **combine**.
  It uses recursive calls to itself, therefore the complexity is calculated with recursive equations.

$W_{MS}(N) \leq 2 \cdot W_{MS}\left(\frac{N}{2}\right) + N - 1;$

$W_{MS}(N) \leq N \cdot \log(N) + O(N)$

$B_{MS}(N) \geq 2 \cdot B_{MS}\left(\frac{N}{2}\right) + \frac{N}{2}$

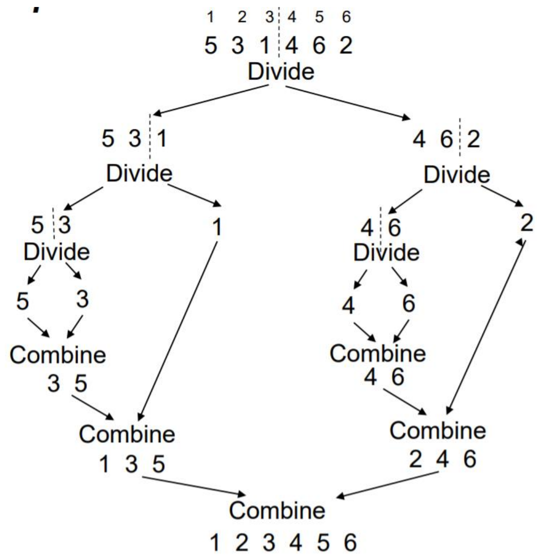
$B_{MS}(N) \geq \frac{1}{2} \cdot N \cdot log(N)$

$$A_{MS}(N) = \Theta(N \cdot \log(N))$$

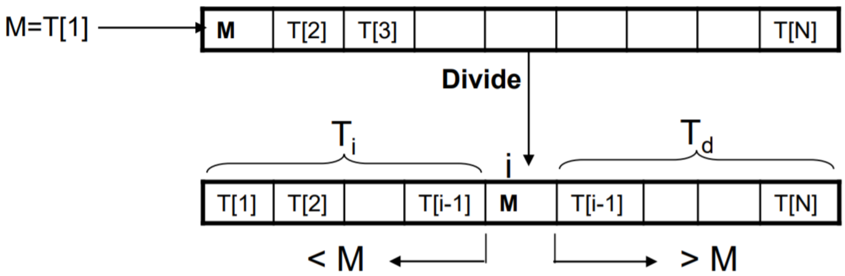
The comparisons made by combine in mergesort are:

$$\lfloor N/2 \rfloor \leq n_{combine}(\sigma, \sigma_l, \sigma_r) \leq N - 1$$

The runtime of the algorithm is good, but it uses dynamic memory and being recursive makes it less efficient regarding the stack.



- **Quicksort**: Quicksort is also a divide and conquer algorithm which is divided in two parts, **divide** and **combine**.
  Divide takes an element of the array and then the array is sorted based in that element.



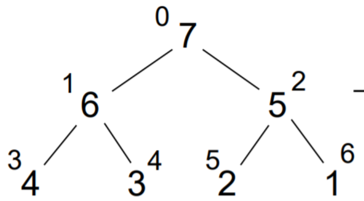
The **pivot** (M) can be selected in a lot of ways, not just the first element.

$$W_{QS}(N) \leq \frac{N^2}{2} - \frac{N}{2}$$
$$A_{QS}(N) = 2 \cdot N \cdot \log(N) + O(N)$$
$$B_{QS}(N) \leq O(N \cdot \log(N))$$

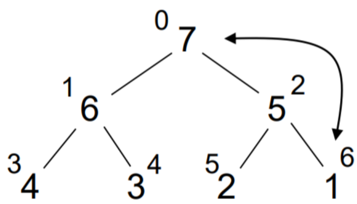
Mergesort uses another table and quicksort has a worse worst case.

- **HeapSort**: A heap is an almost complete binary tree. A **Max-heap** is a heap such that for all subtrees, the children are smaller than the parent.

Example:



0	1	2	3	4	5	6
7	6	5	4	3	2	1



0	1	2	3	4	5	6
7	6	5	4	3	2	1

swap(T[0],T[6])

0	1	2	3	4	5	6
1	6	5	4	3	2	7

In order to get the index of a child given the index of a parent we just apply this formula:
Parent: J Left child: 2·J+1 Right child: 2·J+2
If we have the child index we apply this:
Child: J Parent: $\lfloor (J-1)/2 \rfloor$

Heapsort is divided in 2 parts, first we **create the Max-heap** with the function **heapify** and finally you **sort the Max-heap**.

The height of the **Max-heap** depends on the number of nodes. **Height(T)=$\lfloor \log(N) \rfloor$**
Abstract runtime of heapsort:

$$\boldsymbol{W_{HS}(N) = \Theta\big(N \cdot log(N)\big) = B_{HS}(N) = A_{HS}(N)}$$

2. Decision trees for sorting algorithms

There is not any sorting algorithm which is better than heapsort($\boldsymbol{\Theta\big(N \cdot log(N)\big)}$), in order to prove this, we have decision trees.

A decision tree $\boldsymbol{T_A}^{\boldsymbol{N}}$ is composed by a **comparison-based sorting algorithm A** and a **size N**.

The **height of a node** in a tree is the number of edges on the longest path from a node to a leaf.
The **depth of a node** in a tree is the number of edges from the node to the root node.
The **height of a tree** = **Depth of a tree** is the height of the root node.

The **number of leaves** in $\boldsymbol{T_A}^{\boldsymbol{N}}$ is **N!**, and the **height of a decision tree** with N! leaves is $\lceil \log(N!) \rceil$
The **n$_A$(σ) = number of key comparisons** is $N_A(\sigma) = depth_{T_A{}^N}(L_\sigma)$
The worst case of an algorithm is $W_A(N) = max_{\sigma \epsilon \Sigma_N} \; n_A(\sigma) = \max depth_{\sigma \epsilon \Sigma_N} \; n_A(\sigma)$

The **minimum height/worst case of an algorithm**, regarding comparisons, with N! leaves is $\lceil \log(N!) \rceil$.
The **minimum average height/average case of an algorithm**, regarding comparisons, with N! leaves is $A_A(N) = \Omega(N \cdot \log(N))$

The **external path length (EPL)** are the sum of lengths of all paths from the root to a leaf
$EPL_{\min}(K) = K \cdot \lceil \log(K) \rceil + K - 2^{\lceil \log(K) \rceil}$