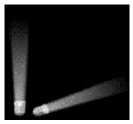
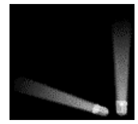
Tournament Trees





Winner trees.

Loser Trees.

Winner Trees

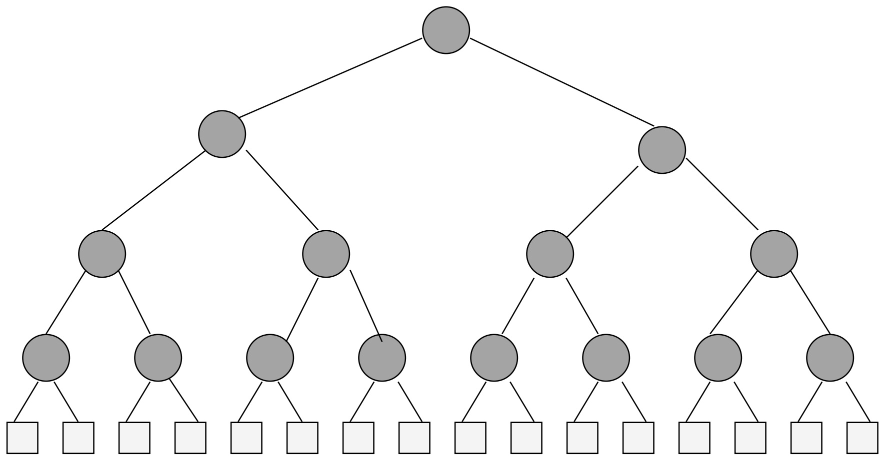
Complete binary tree with n external nodes and n - 1 internal nodes.

External nodes represent tournament players.

Each internal node represents a match played between its two children; the winner of the match is stored at the internal node.

Root has overall winner.

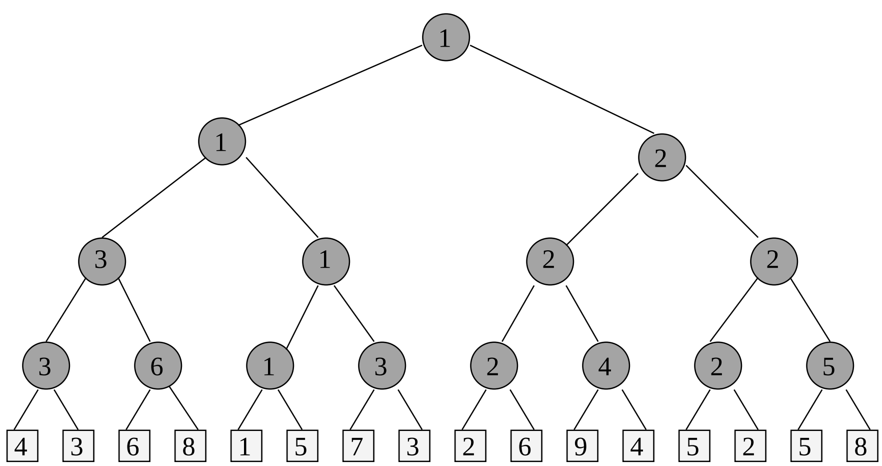
Winner Tree For 16 Players



player

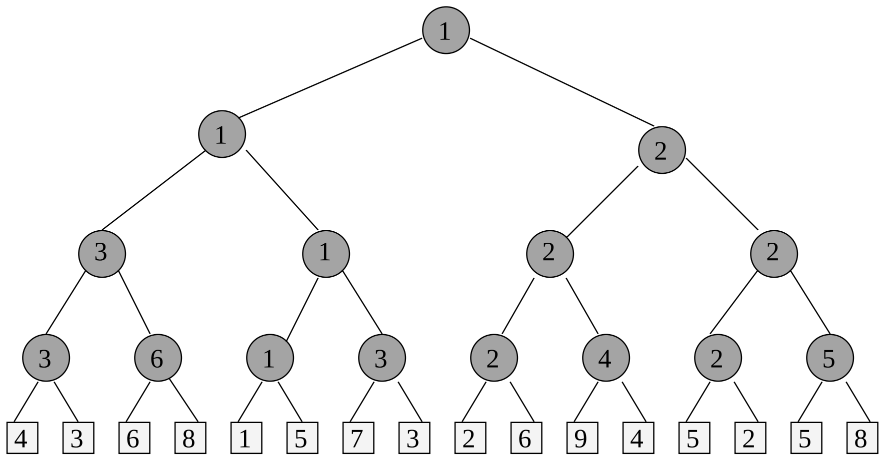
match node

Winner Tree For 16 Players



Smaller element wins => min winner tree.

Winner Tree For 16 Players



height is $\log_2 n$ (excludes player level)

Complexity of Initialize

- O(1) time to play match at each match node.
- n - 1 match nodes.
- O(n) time to initialize n player winner tree.

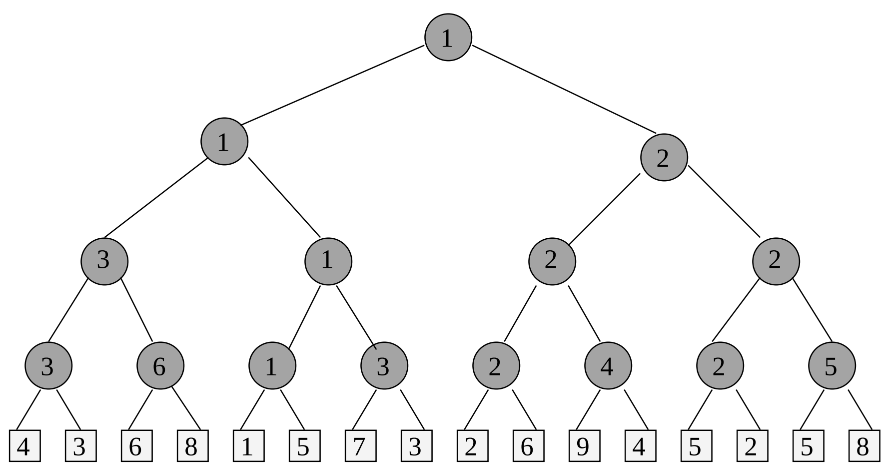
Applications

Sorting.

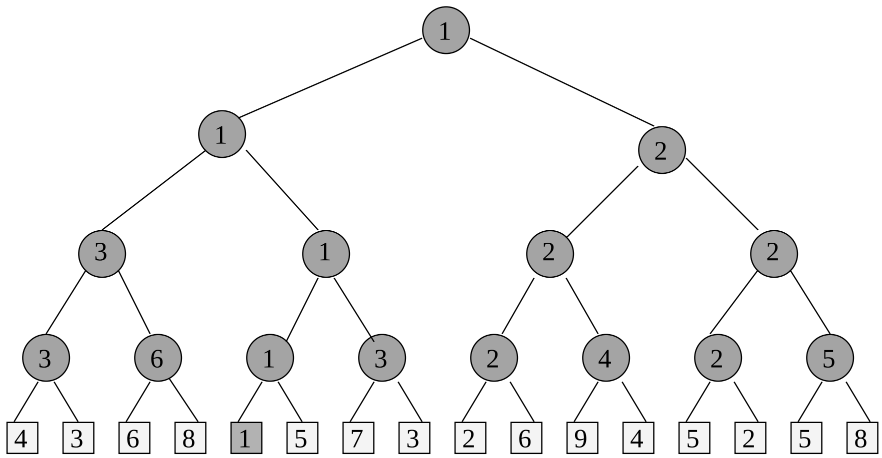
Put elements to be sorted into a winner tree.

Repeatedly extract the winner and replace by a large value.

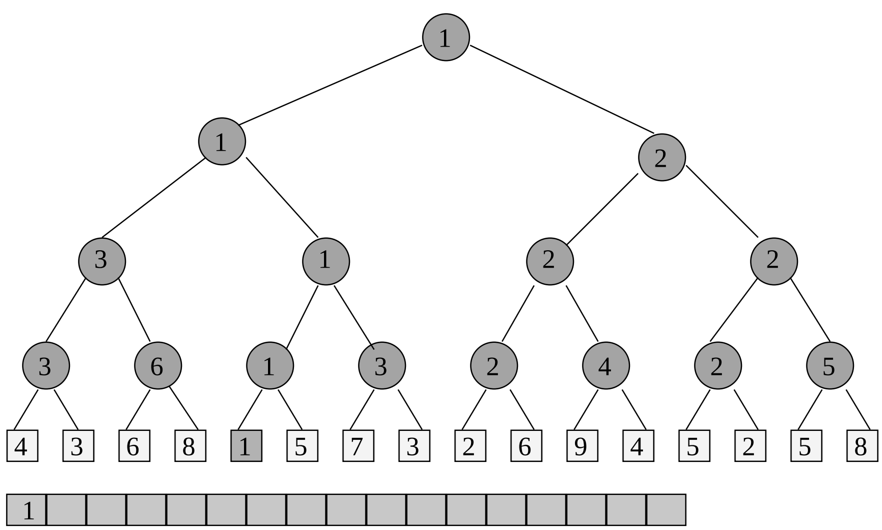
Sort 16 Numbers



Sort 16 Numbers

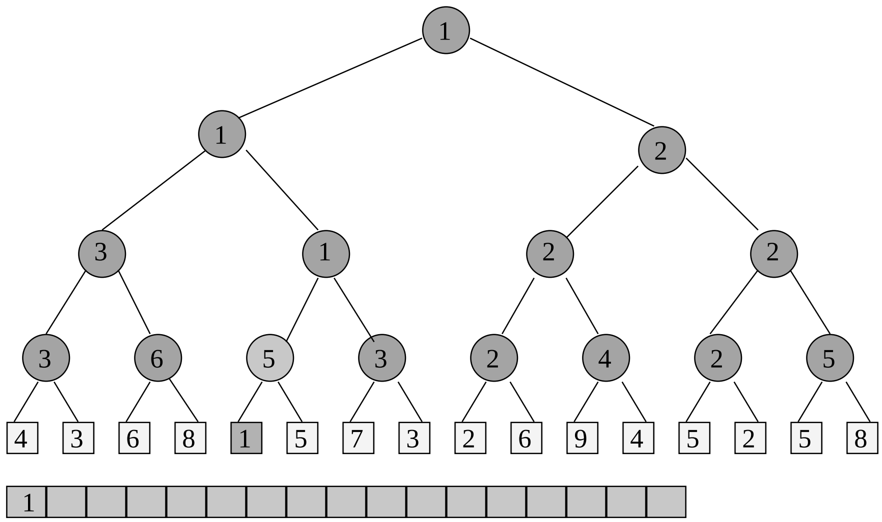


Sort 16 Numbers



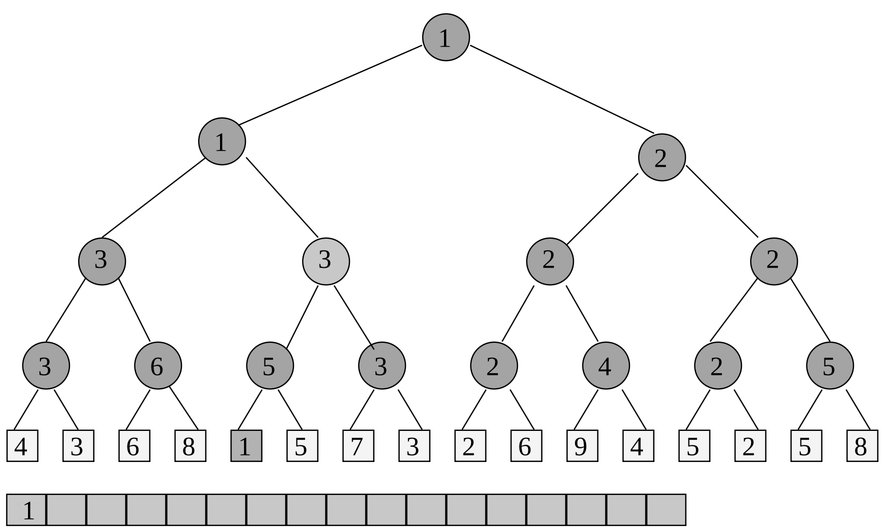
Sorted array.

Sort 16 Numbers



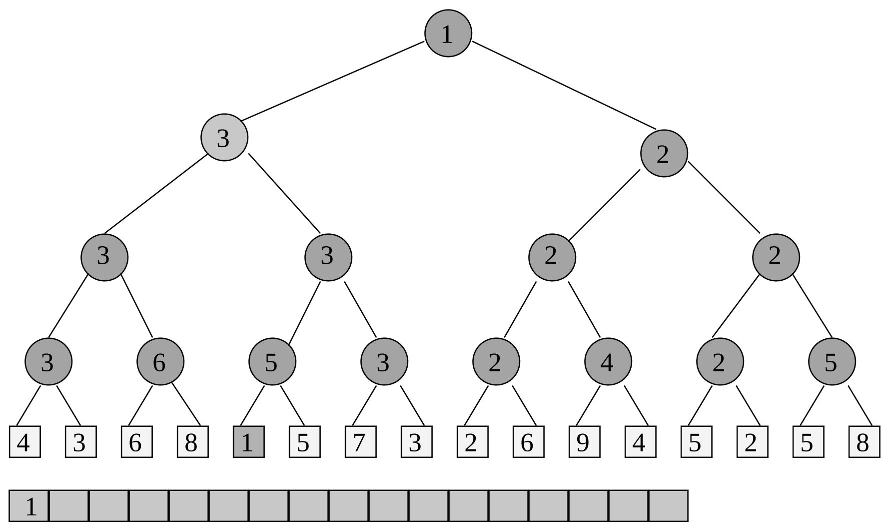
Sorted array.

Sort 16 Numbers



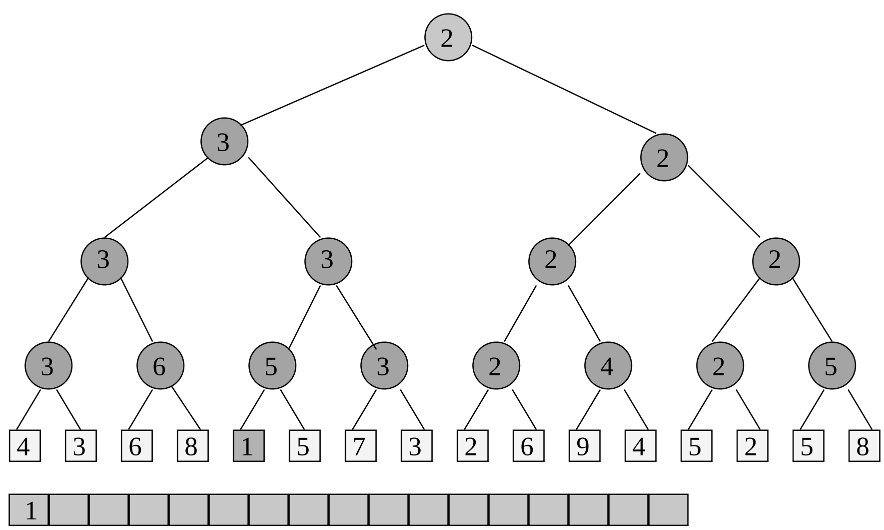
Sorted array.

Sort 16 Numbers



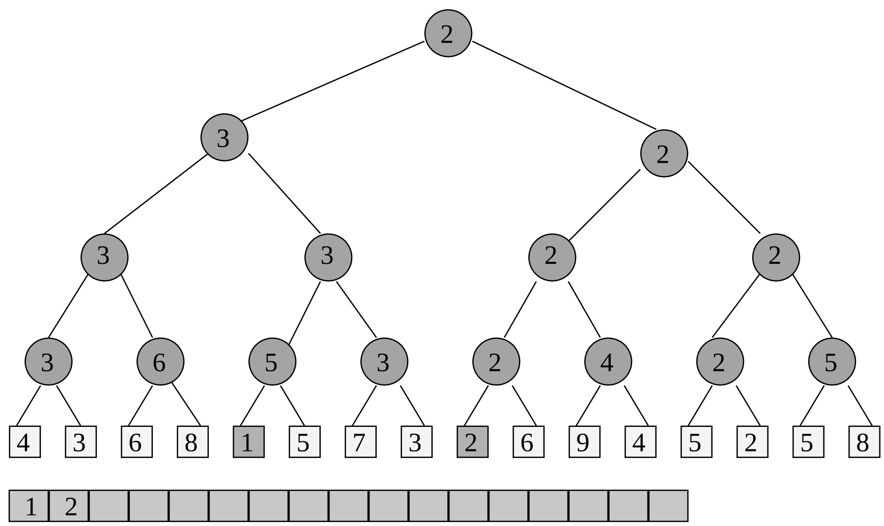
Sorted array.

Sort 16 Numbers



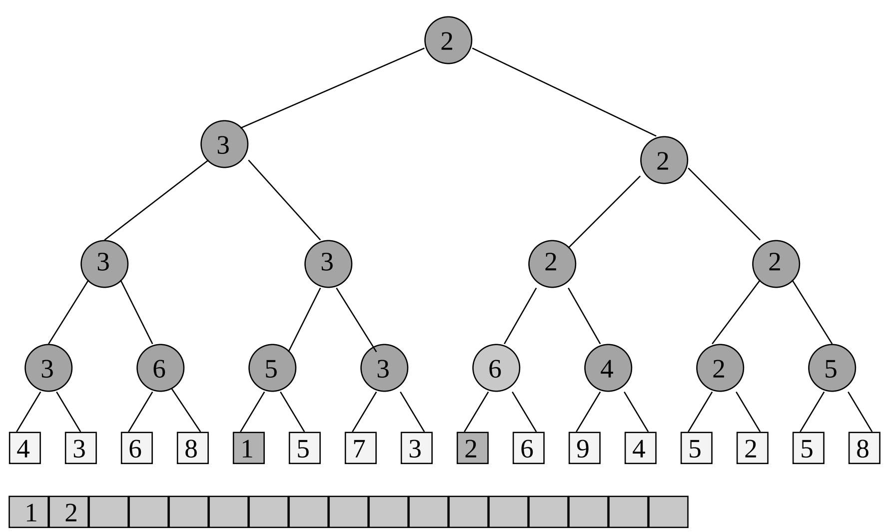
Sorted array.

Sort 16 Numbers



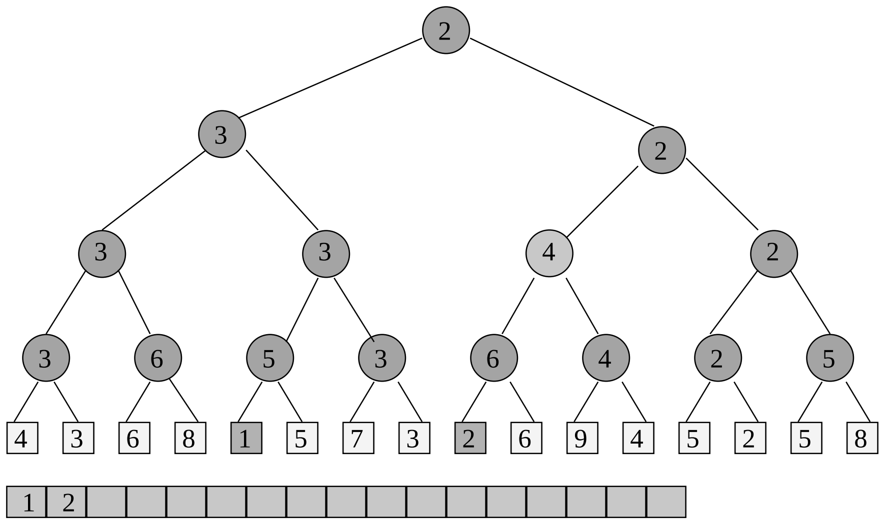
Sorted array.

Sort 16 Numbers



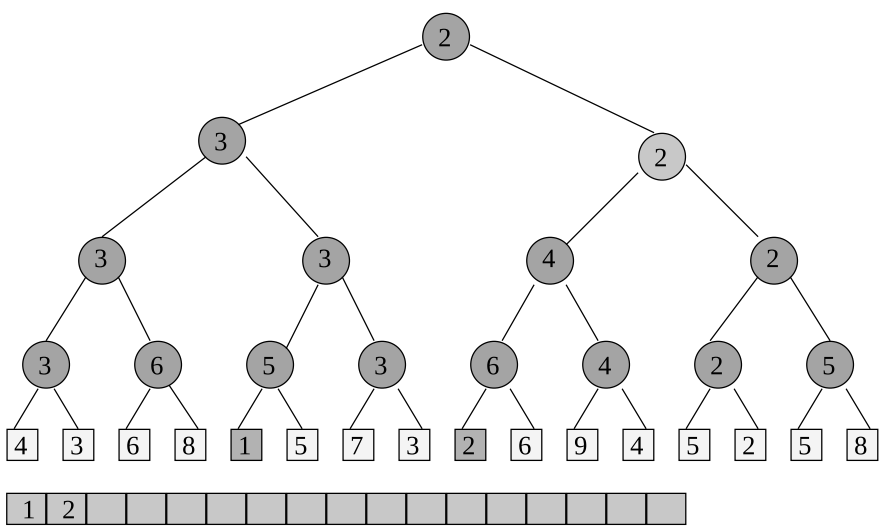
Sorted array.

Sort 16 Numbers



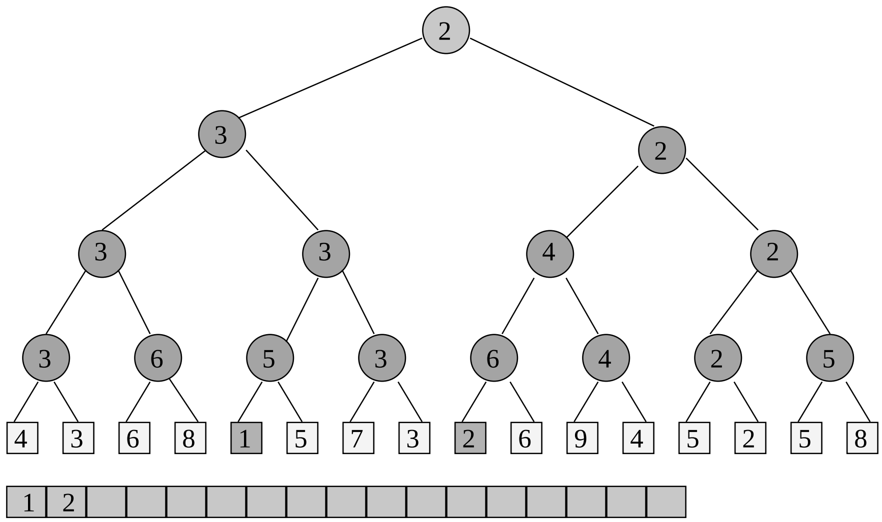
Sorted array.

Sort 16 Numbers



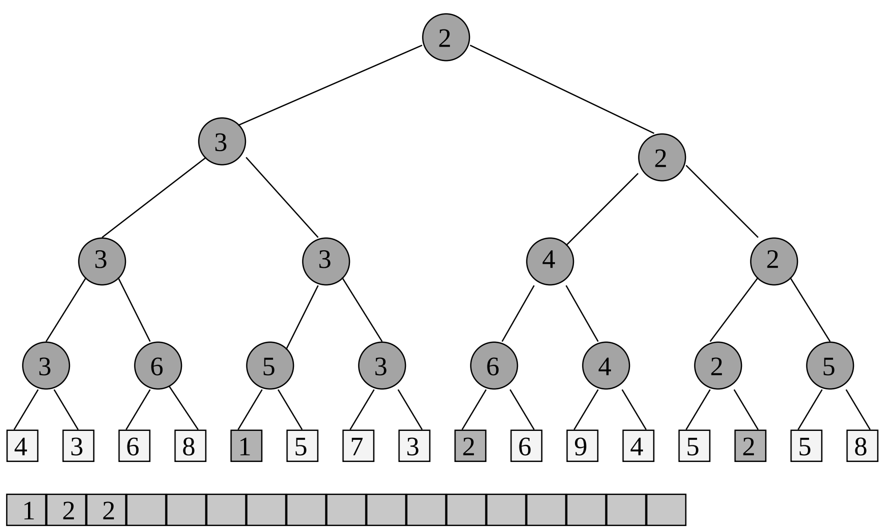
Sorted array.

Sort 16 Numbers



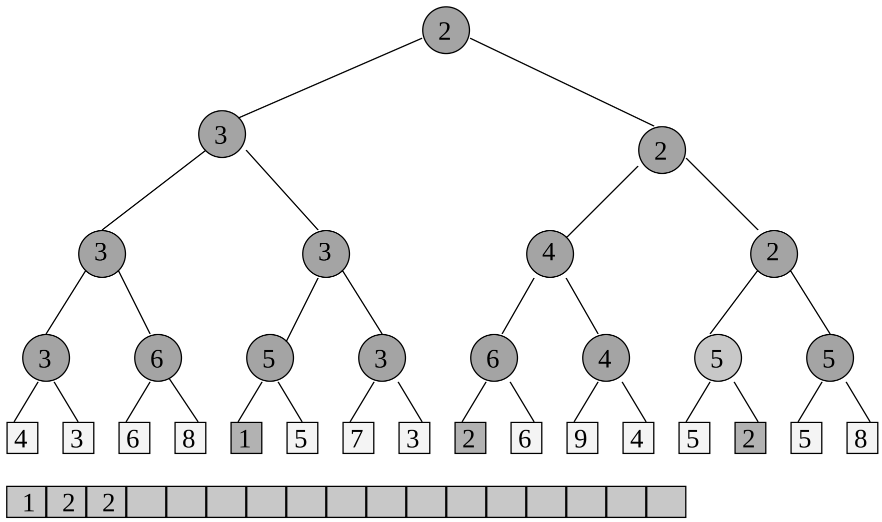
Sorted array.

Sort 16 Numbers



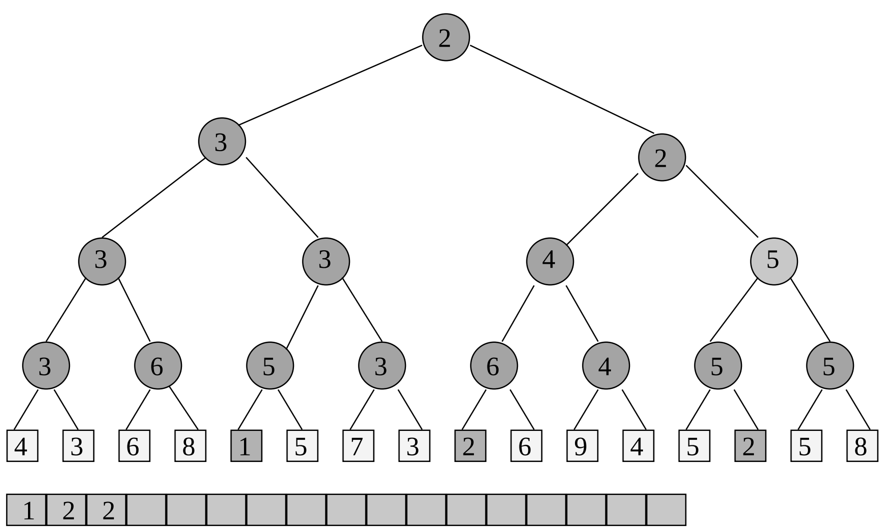
Sorted array.

Sort 16 Numbers



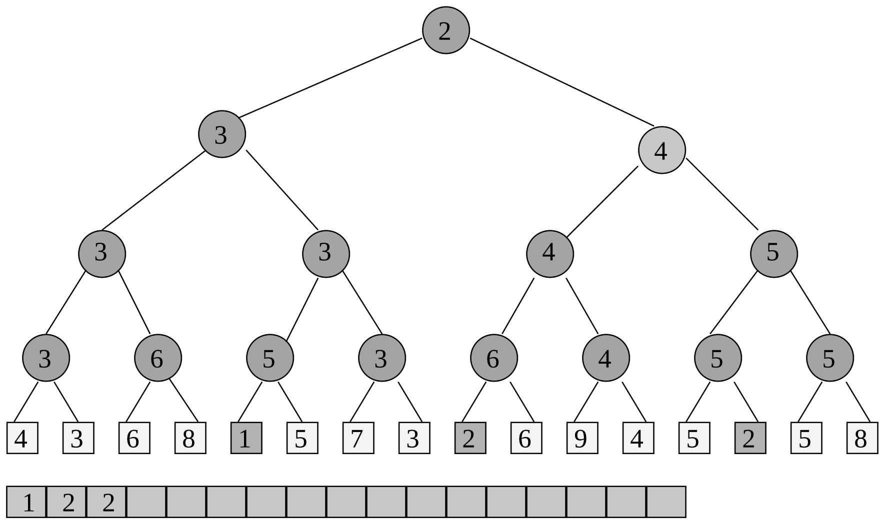
Sorted array.

Sort 16 Numbers



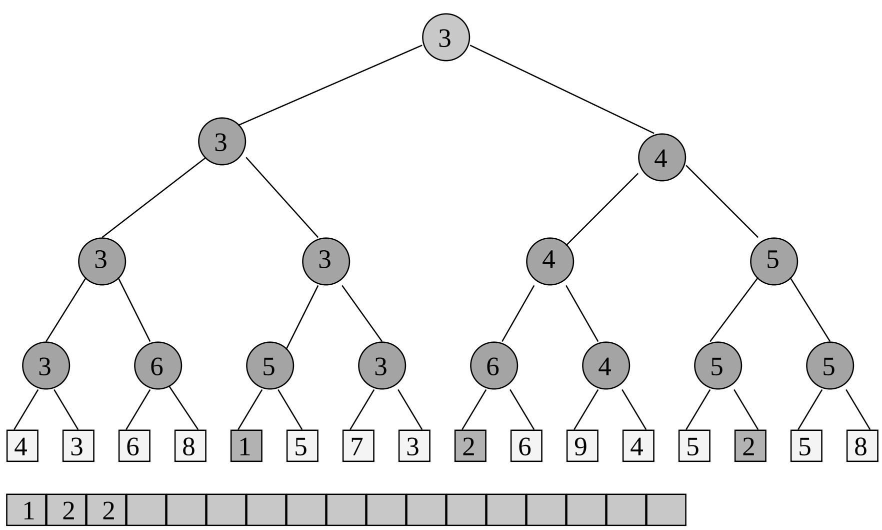
Sorted array.

Sort 16 Numbers



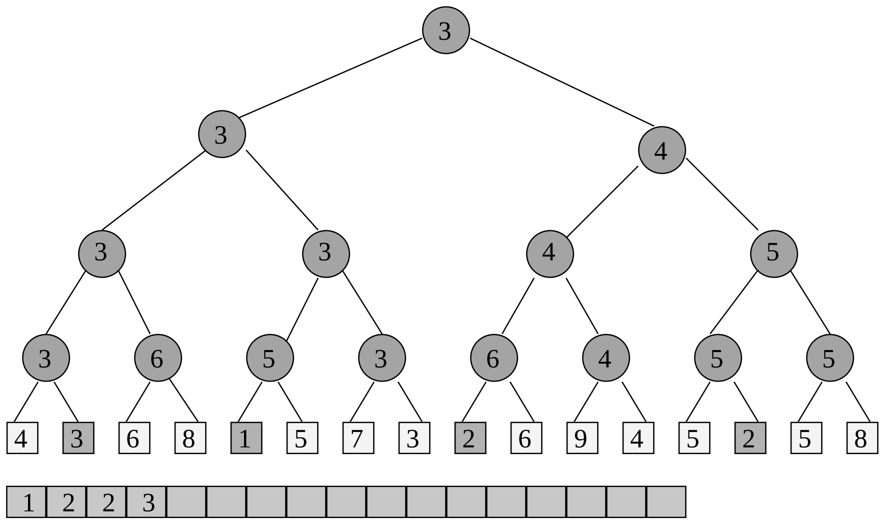
Sorted array.

Sort 16 Numbers



Sorted array.

Sort 16 Numbers



Sorted array.

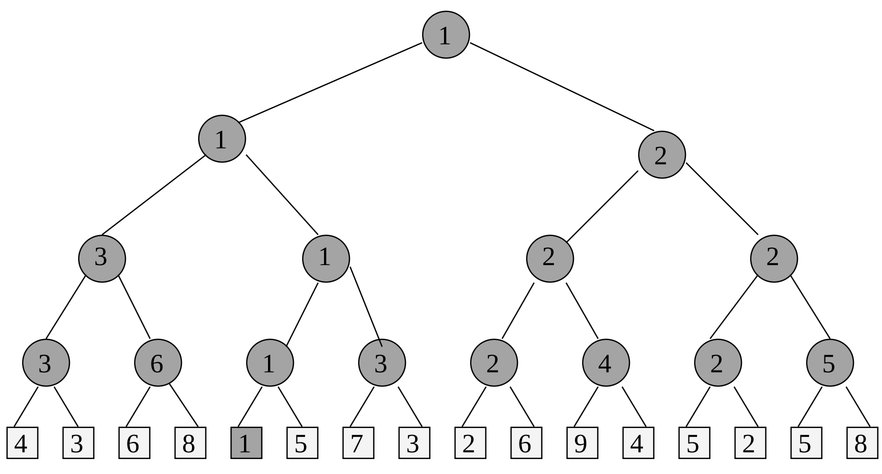
Time To Sort

- Initialize winner tree.
  - O(n) time
- Remove winner and replay.
  - O(log n) time
- Remove winner and replay n times.
  - O(n log n) time
- Total sort time is O(n log n).
- Actually Theta(n log n).

Winner Tree Operations

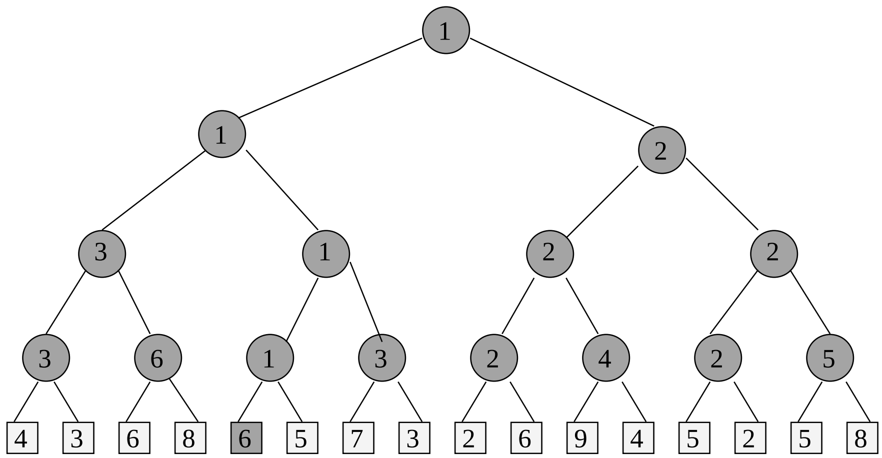
- Initialize
  - O(n) time
- Get winner
  - O(1) time
- Remove winner and replay
  - O(log n) time
  - more precisely Theta(log n)

Replace Winner And Replay



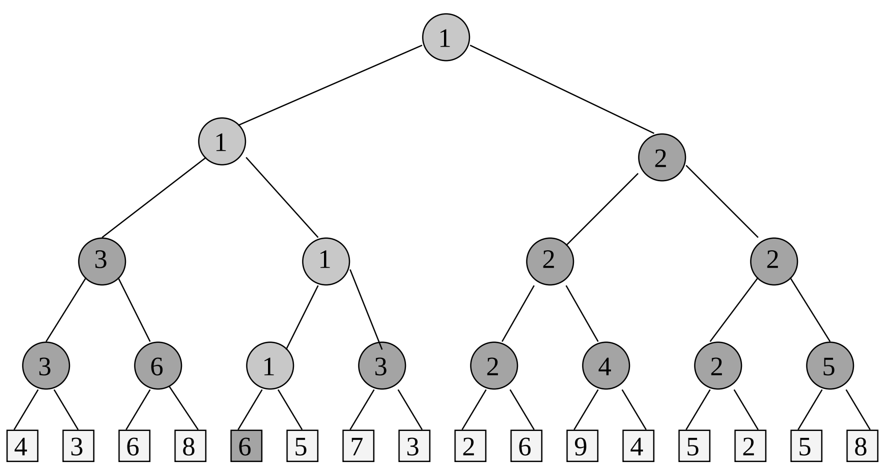
Replace winner with 6.

Replace Winner And Replay



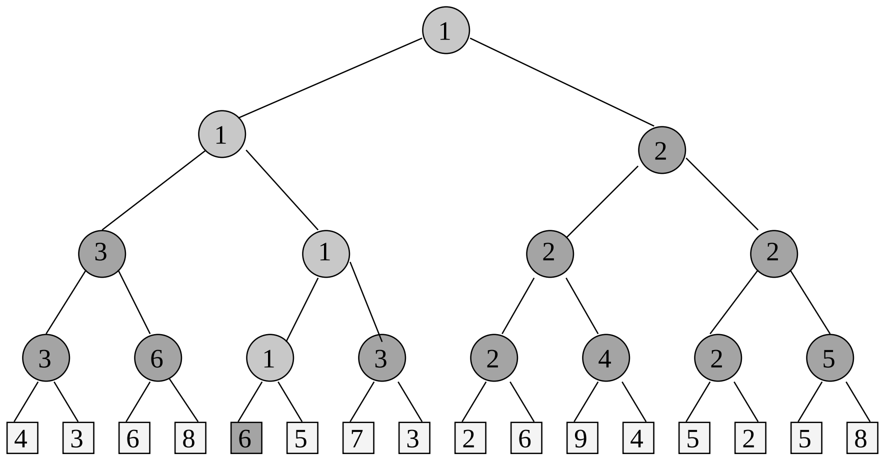
Replay matches on path to root.

Replace Winner And Replay



Replay matches on path to root.

Replace Winner And Replay



Opponent is player who lost last match played at this node.

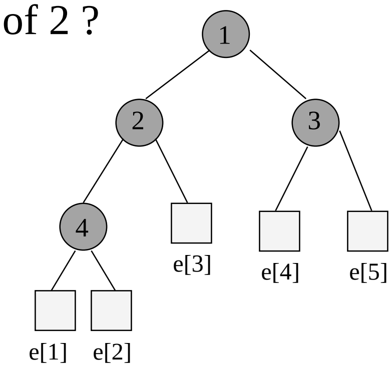
Loser Tree

- Each match node stores the match loser rather than the match winner.
- We will skip Loser Tree.
- Read Section 10.4

What if n is not a power of 2 ?

- n=5 players
- (n-1) internal tree nodes

(Property of Complete Binary Tree)

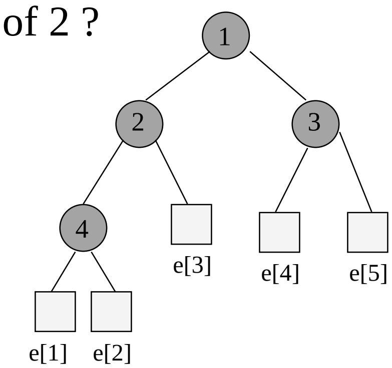


- The leftmost internal node at the lowest level is numbered 2^s where $s = \lfloor \log(n-1) \rfloor$
- # of internal nodes at the lowest level = $(n- 2^s)$
- # of external nodes at the lowest level = LowExt = $2*(n- 2^s)$

What if n is not a power of 2 ?

- n=5 players
- (n-1) internal tree nodes

(Property of Complete Binary Tree)



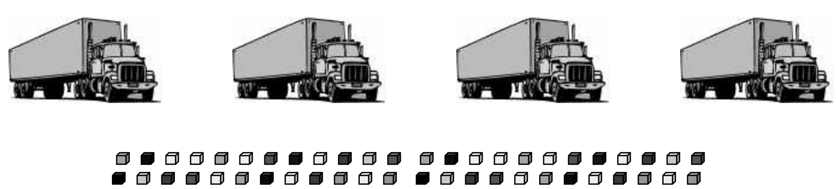
- The leftmost external node at the 2nd lowest level is numbered (LowExt + 1)
- For any external node e[i], its parent index p is given by:

$$p = 2^s + (i-1)/2 \quad \text{if} \quad i \le \text{LowExt}$$
$$p = (i - \text{LowExt} + n - 1)/2 \quad \text{if} \quad i > \text{LowExt}$$

Winner Tree Applications

- Bin Packing Problem
- Truck loading
  - n packages to be loaded into trucks
  - each package has a weight
  - each truck has a capacity of c tons
  - minimize number of trucks

Truck Loading



- n packages to be loaded into trucks
- each package has a weight
- each truck has a capacity of c tons
- minimize number of trucks

Truck Loading

n = 5 packages

weights [2, 5, 6, 3, 4]

truck capacity c = 10

Load packages from left to right. If a package doesn't fit into current truck, start loading a new truck.

Truck Loading

n = 5 packages

weights [2, 5, 6, 3, 4]

truck capacity c = 10

truck1 = [2, 5]

truck2 = [6, 3]

truck3 = [4]

uses 3 trucks when 2 trucks suffice

Truck Loading

n = 5 packages

weights [2, 5, 6, 3, 4]

truck capacity c = 10

truck1 = [2, 5, 3]

truck2 = [6, 4]

Bin Packing

- n items to be packed into bins
- each item has a size
- each bin has a capacity of c tons
- minimize number of bins

Bin Packing

Truck loading is same as bin packing.

- Truck is a bin that is to be packed (loaded).
- Package is an item/element.

Bin packing to minimize number of bins is NP-hard.

Several fast heuristics have been proposed.

Bin Packing Heuristics

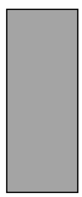
- First Fit.
  - Bins are arranged in left to right order.
  - Items are packed one at a time in given order.
  - Current item is packed into leftmost bin into which it fits.
  - If there is no bin into which current item fits, start a new bin.

First Fit

n = 4

weights = [4, 7, 3, 6]

capacity = 10



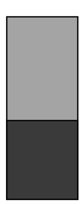
Pack red item into first bin.

First Fit

n = 4

weights = [4, 7, 3, 6]

capacity = 10



Pack blue item next.

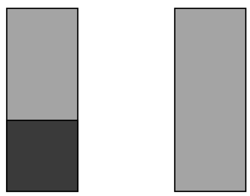
Doesn’t fit, so start a new bin.

First Fit

n = 4

weights = [4, 7, 3, 6]

capacity = 10

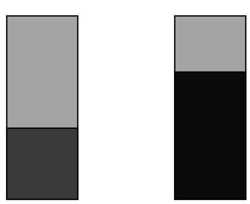


First Fit

n = 4

weights = [4, 7, 3, 6]

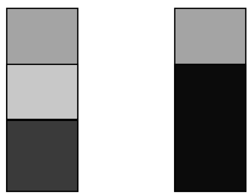
capacity = 10



Pack yellow item into first bin.

First Fit

n = 4
weights = [4, 7, 3, 6]
capacity = 10

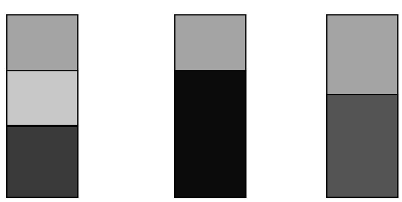


Pack green item.

Need a new bin.

First Fit

n = 4
weights = [4, 7, 3, 6]
capacity = 10



Not optimal.

2 bins suffice.

Bin Packing Heuristics

- First Fit Decreasing.
  - Items are sorted into decreasing order.
  - Then first fit is applied.

(Can this one find the optimal solution for the previous example?)

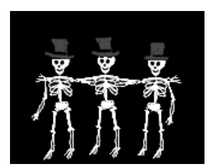
Bin Packing Heuristics

- Best Fit.
  - Items are packed one at a time in given order.
  - To determine the bin for an item, first determine set S of bins into which the item fits.
  - If S is empty, then start a new bin and put item into this new bin.
  - Otherwise, pack into bin of S that has least available capacity.

Bin Packing Heuristics

- Best Fit Decreasing.
  - Items are sorted into decreasing order.
  - Then best fit is applied.

Performance



- For first fit and best fit:

  Heuristic Bins ≤ (17/10)(Minimum Bins) + 2

- For first fit decreasing and best fit decreasing:

  Heuristic Bins ≤ (11/9)(Minimum Bins) + 4

[The proofs of these facts are laborious and can be found in two papers published in *J. of Combinatorial Theory* (1976) and *SIAM J. on Computing* (1974)].

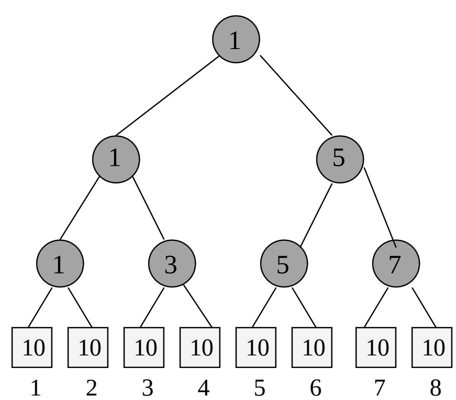
Complexity of First Fit

Use a max tournament tree in which the players are n bins and the value of a player is the available capacity in the bin.

O(n log n), where n is the number of items.

First Fit

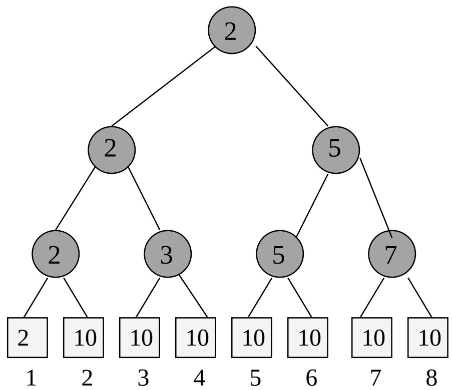
Example: n=8, c=10, s[ ]={8, 6, 5, 3, 6, 4, 2, 7}



Initial

First Fit

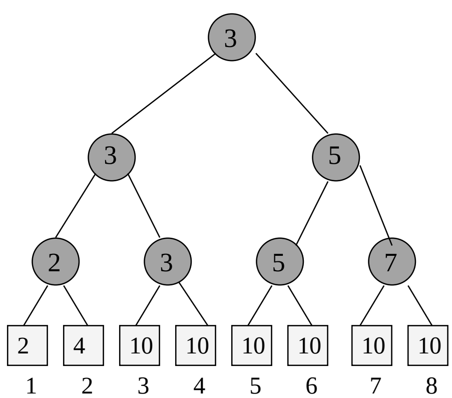
Example: n=8, c=10, s[ ]={8, 6, 5, 3, 6, 4, 2, 7}



After s[1]=8 packed

First Fit

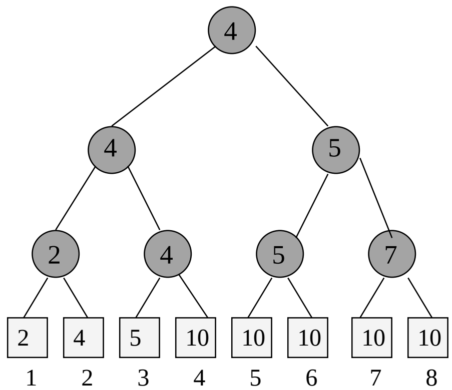
Example: n=8, c=10, s[ ]={8, 6, 5, 3, 6, 4, 2, 7}



After s[2]=6 packed

First Fit

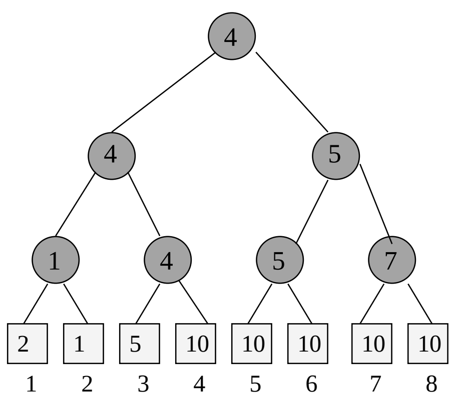
Example: n=8, c=10, s[ ]={8, 6, 5, 3, 6, 4, 2, 7}



After s[3]=5 packed

First Fit

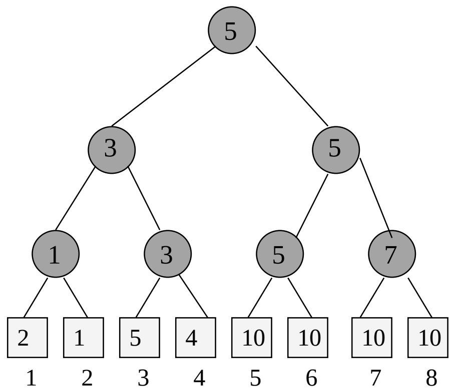
Example: n=8, c=10, s[ ]={8, 6, 5, 3, 6, 4, 2, 7}



After s[4]=3 packed

First Fit

Example: n=8, c=10, s[ ]={8, 6, 5, 3, 6, 4, 2, 7}



After s[5]=6 packed

Total no. of bins used = 4 **TIME COMPLEXITY = O(nlogn)**