

# Data Structures and Algorithms– Lab 1 1

Iulia–Cristina Stanica



# Roadmap

## Heap

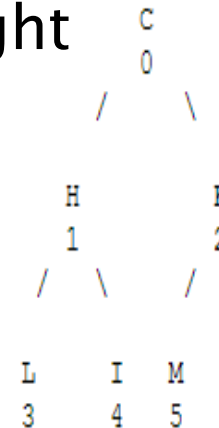
- ▶ Heap
- ▶ Heap sort
- ▶ C++ implementation

# Ex 1

- ▶ Add in a min-heap the following elements:  
25, 17, 36, 2, 3, 100, 1, 19, 17
- ▶ Delete 3 elements from the heap.
- ▶ (do this exercise on paper)

# Implementation of Heap

- ▶ as a binary tree
- ▶ as an array
  - Define the numbers of the nodes based on the levels, from top to bottom, left to right
  - Save the values in an array
  - If CI is the current index:
    - $\text{Parent}(\text{CI}) = (\text{CI} - 1) / 2$
    - $\text{RightChild}(\text{CI}) = 2 * (\text{CI} + 1)$
    - $\text{LeftChild}(\text{CI}) = 2 * \text{CI} + 1$



C	H	K	L	I	M
0	1	2	3	4	5

CI pour H =1=> RightChild pour H (CI)=2\*(1+1)=4 => I

## Ex. 2

- ▶ Test the Heap implementation with the values from Ex1.

Imagine that the Heap class has the array starting at index 1, not 0! How should we change the formulas which link the parent with its children (formulas slide 13)?

## Ex. 3

- ▶ Add a function to display the heap by levels. Test your implementation with the heap from ex 1.

# Heap Sort (max heap)

- Heap sort is a sorting method which uses the comparisons in an array.
- The main idea is to look at the array as if it is a binary tree. The first element is the root, the second and the third are the descendants of the root etc.
- In the next step of the algorithm, we construct a heap (we must check all its properties)
- Delete the root several times and put in on the last free place in the array. Adjust the heap.
- For a max heap, we will obtain the sorting in ascending order.

10	4	8	5	12	2	6	11	3	9	7	1
----	---	---	---	----	---	---	----	---	---	---	---

[https://commons.wikimedia.org/wiki/  
File:Heap\\_sort\\_example.gif](https://commons.wikimedia.org/wiki/File:Heap_sort_example.gif)

## Ex. 4

- ▶ We consider this array:  
25, 17, 36, 2, 3, 100, 1, 19, 17.
- ▶ Sort the array using heap sort (descending order)



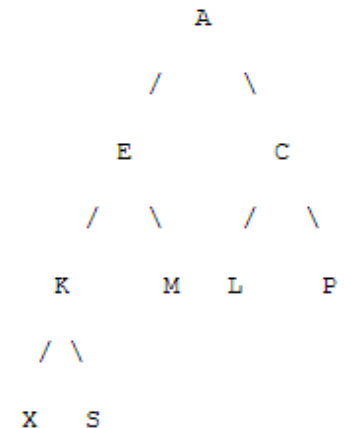
# Implementation of heap sort

## Hint – Steps (possible solution)

1. In a new constructor (with the parameters: dimension and array), convert the array elements into a heap.
  - 1.1. We have a binary tree. Firstly we go to the index of the last node that can be a parent (let's say "ind")
  - 1.2. Apply FilterDown for all the nodes between ind and 0.
2. Create a HeapSort function in which you sort the array using the heap (eg: extract the root multiple times and put it on the first free position in the array).
3. Create an array in the main function and read from keyboard its values. Create a heap and test the sort of the array.

# Extra ex

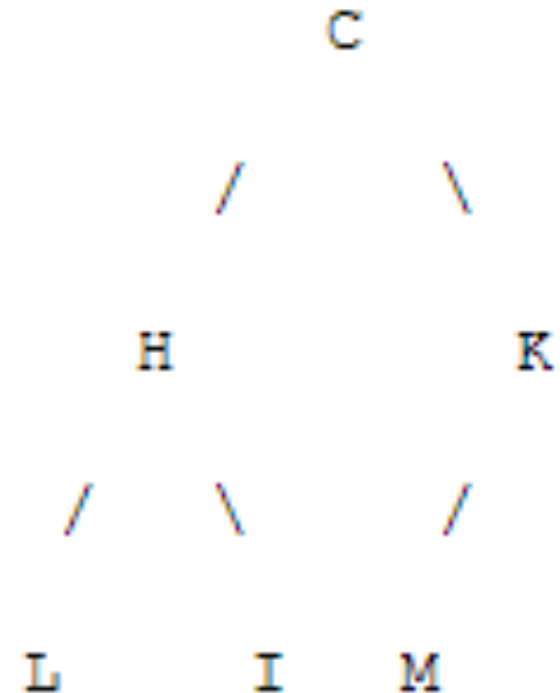
- Implement an (efficient) algorithm to find the largest element in a min-heap.
  - The property of the min heap requires that the parent node is less than its child node (s).
  - Therefore, we can conclude that a non-leaf node cannot be the maximum element because its child node(s) has/have a higher value.
  - You must find the index of the last parent node



# Theory

# Almost complete binary tree

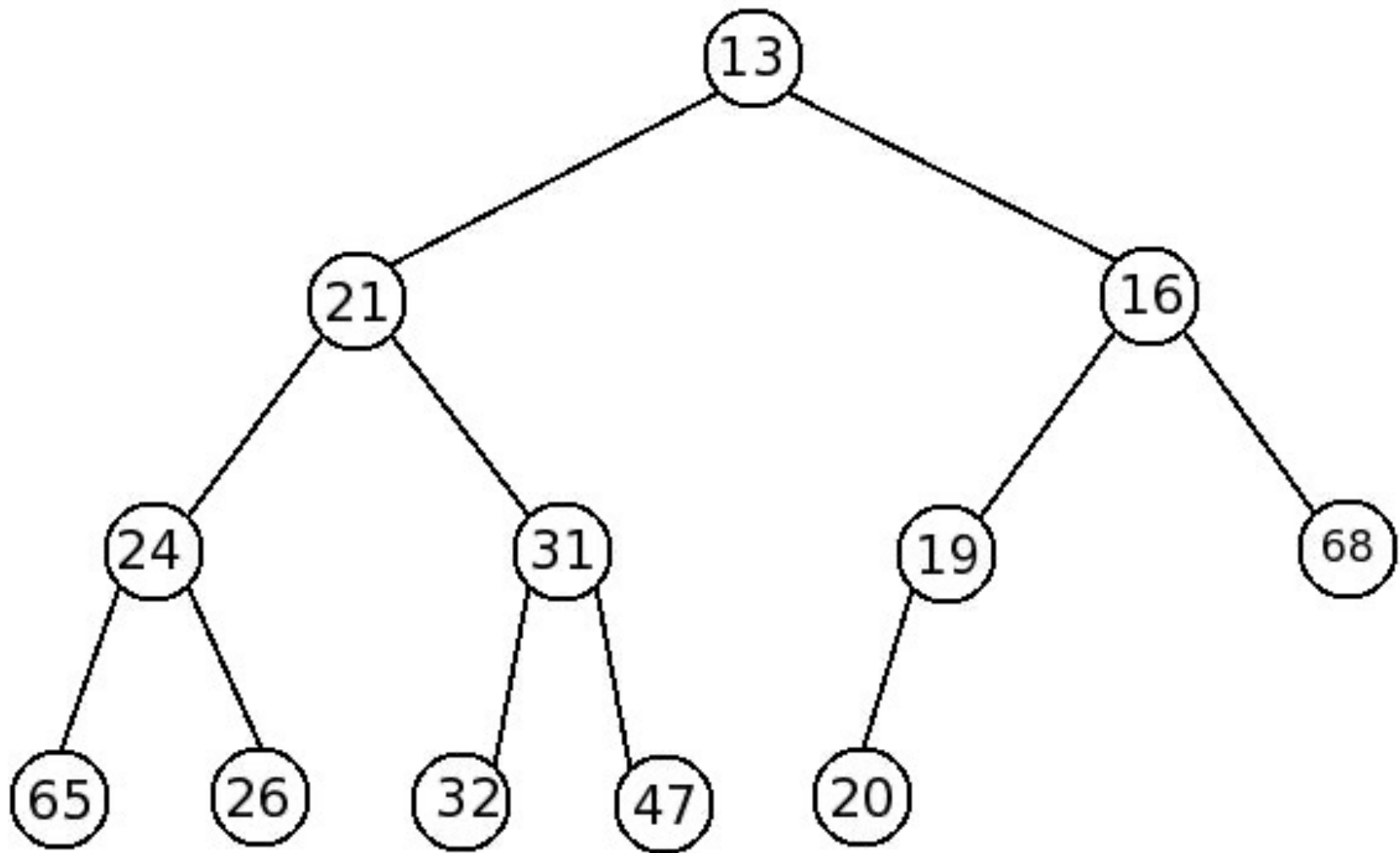
- ▶ **Def:** An almost complete binary tree is a binary tree which has the following properties:
  - All the leaves are on the last level of the tree (or the last 2 levels)
  - All the leaves are situated at the leftmost positions
  - All levels are fully occupied (except for, eventually, the last level)



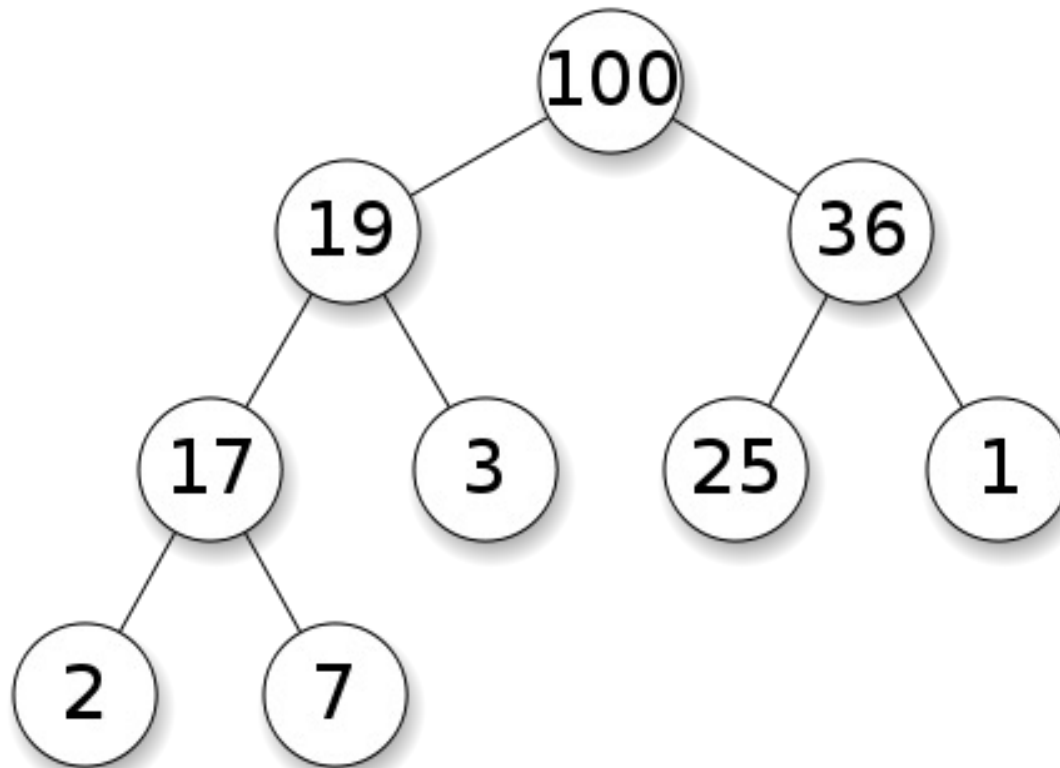
# Heap

- ▶ **Def:** A min heap is an almost complete binary tree where the value of each parent node is less or equal to the values of the children.
- ▶ **Observations:**
  - The min value is in the root.
  - A path from a leaf to the root gives a sequence of numbers in decreasing order.

# Min heap: lowest at top



# MAX Heap



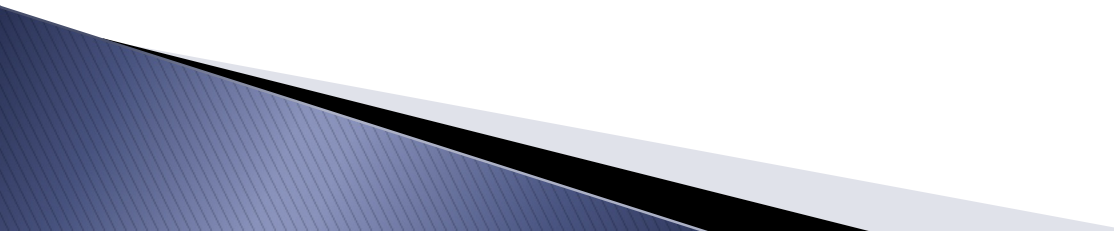
# Operations

- ▶ Insertion
- ▶ Removal



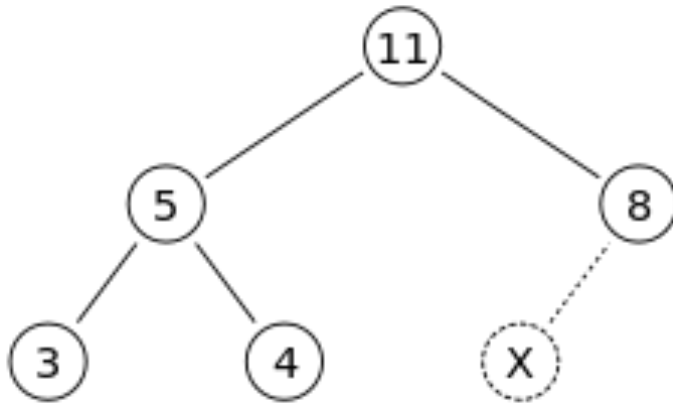
# 1. Insertion

## ► Steps:

- 1. Add the element on the last level, leftmost possible;
  - 2. Compare the added element with its parent; if the order is right, stop;
  - 3. If not, change the element with its parent and return to step 2.
- 

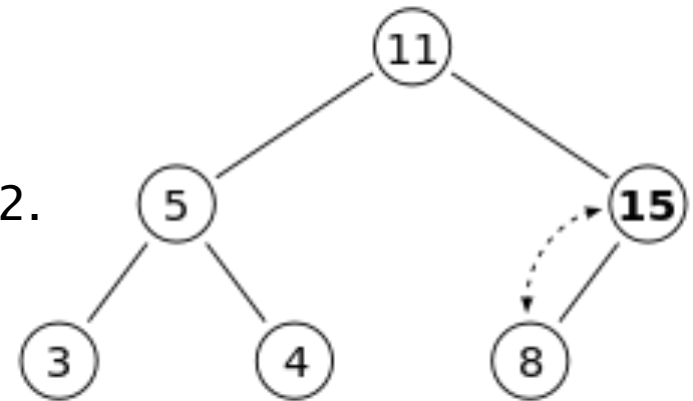
# Example (max heap):

1.



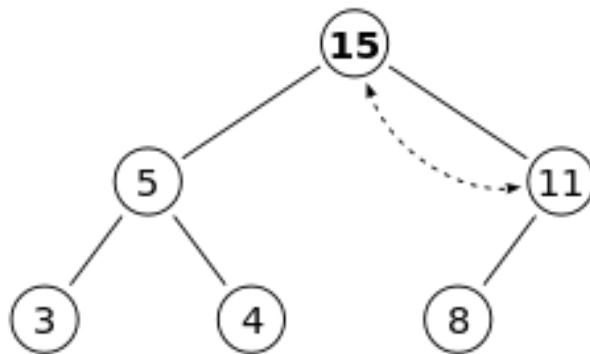
We add 15 (instead of X)

2.



We compare 15 with its parent and swap them

3.

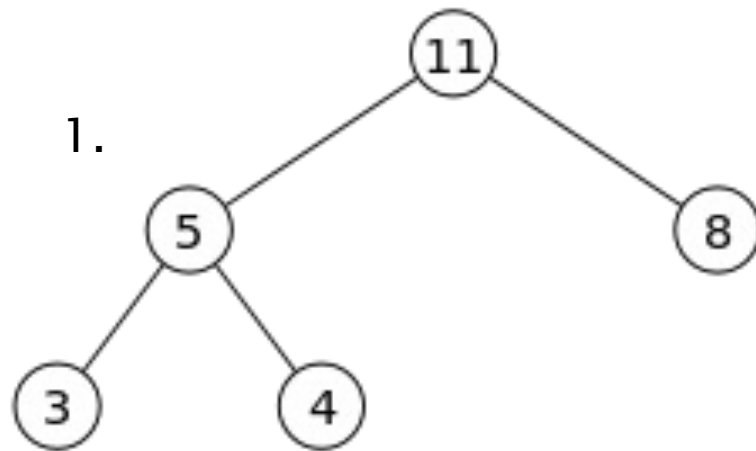


Same operation between 15 and the root

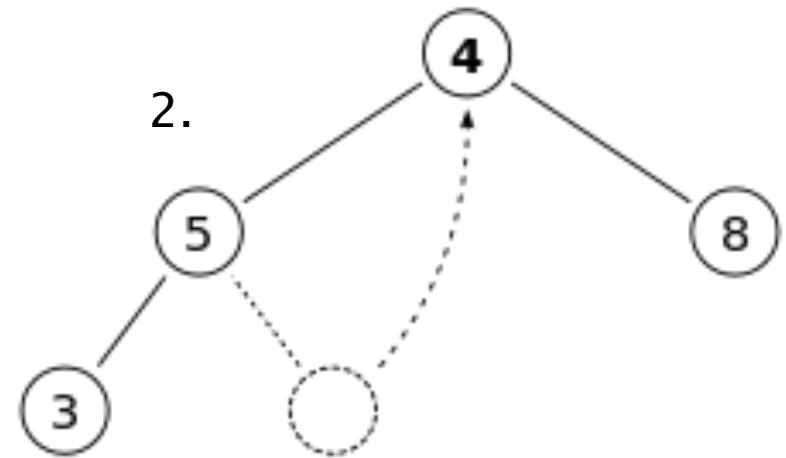
## 2. Removal

- ▶ We delete always the root of the heap.
- ▶ Steps:
  - 1. Temporarily replace the root with the last element from the last level.
  - 2. Compare the new root with its children; if the order is right, stop
  - 3. If not, change the element with its biggest child and return to step 2.

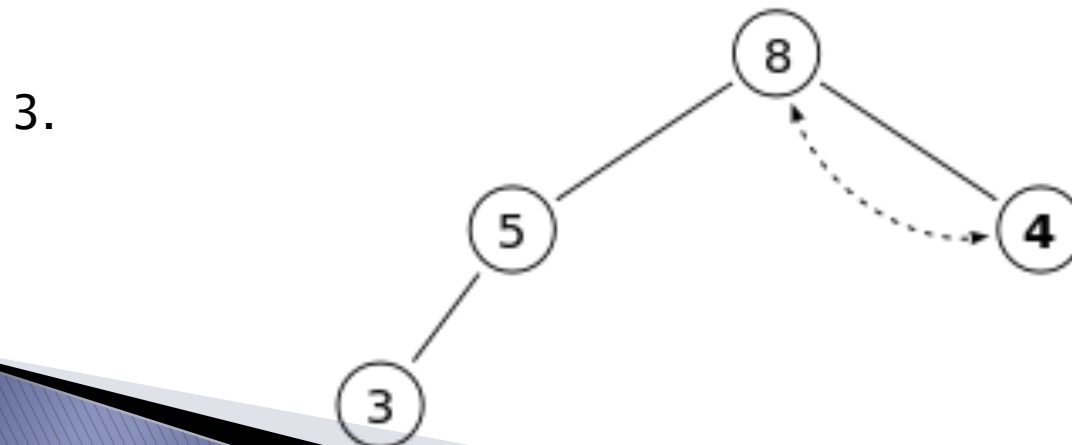
# Example (max heap):



We delete the root (11)



We put the last children in its place



We replace it  
with its biggest  
child