CYDEO

Data Structures and Algorithms Course

Heap

Course Content

- Heap Data Structure
- Implementing Heaps
- Heapify algorithm

What is a Heap?

- Heap is special type of Binary tree with two properties.
    1) It must be a complete tree:
       - Every level except the last is completely filled.
       - And levels are complete from left to right.

    2) Value of every node must be equal or greater than the children.
       (Heap Property for a Max Heap)

Caution: Heap is not a Binary **Search** Tree!

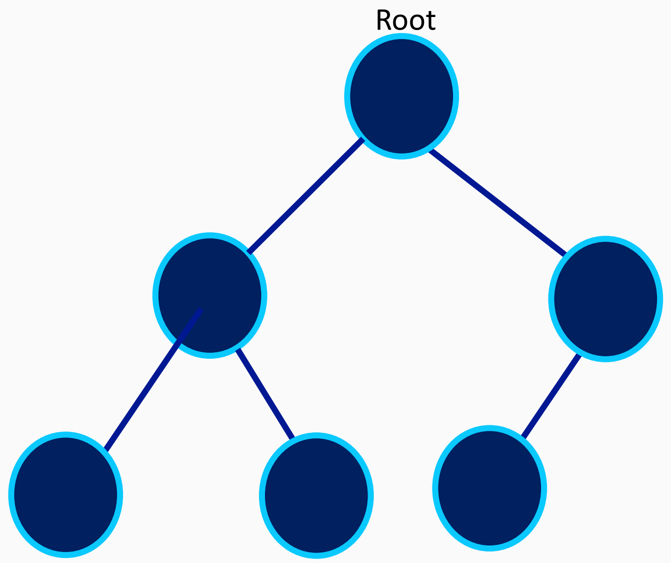
Types of Heap

- There two types of Heap.

1) **Max Heap:**
   - Value of every node must be equal or greater than the children.
   - Max Value at Top

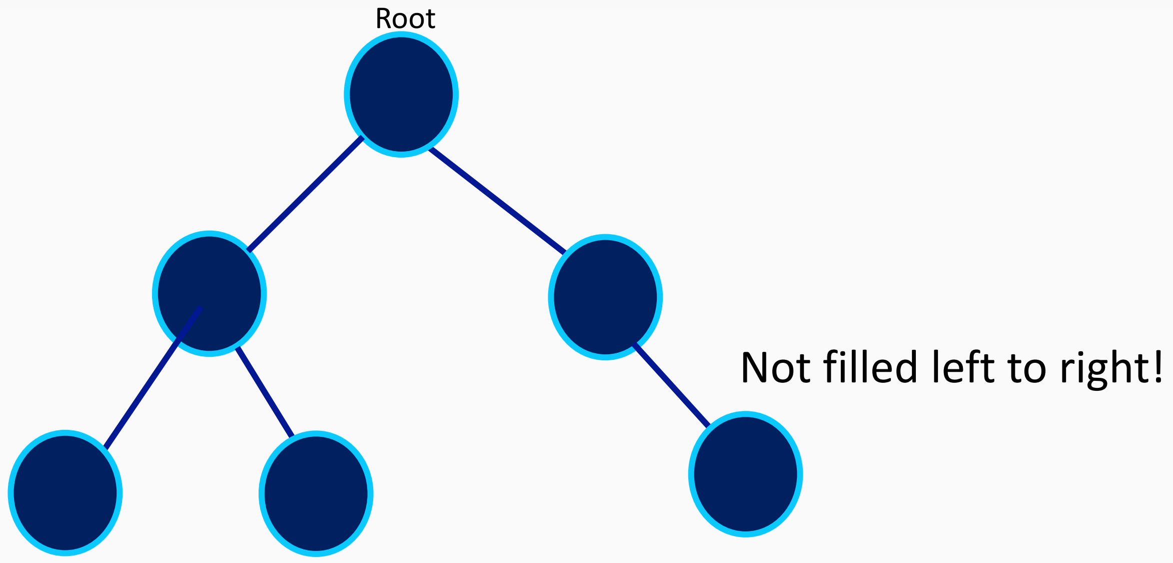
2) **Min Heap:**
   - Value of every node must be equal or smaller than the children.
   - Min Value at Top

Valid Heap?



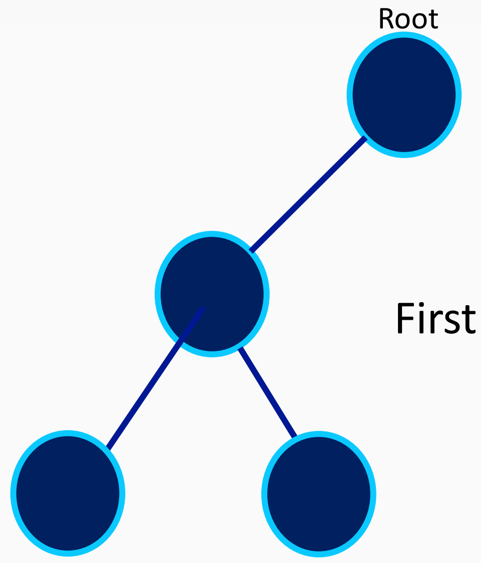
COMPLETE TREE ✓

HEAP PROPERTY ?

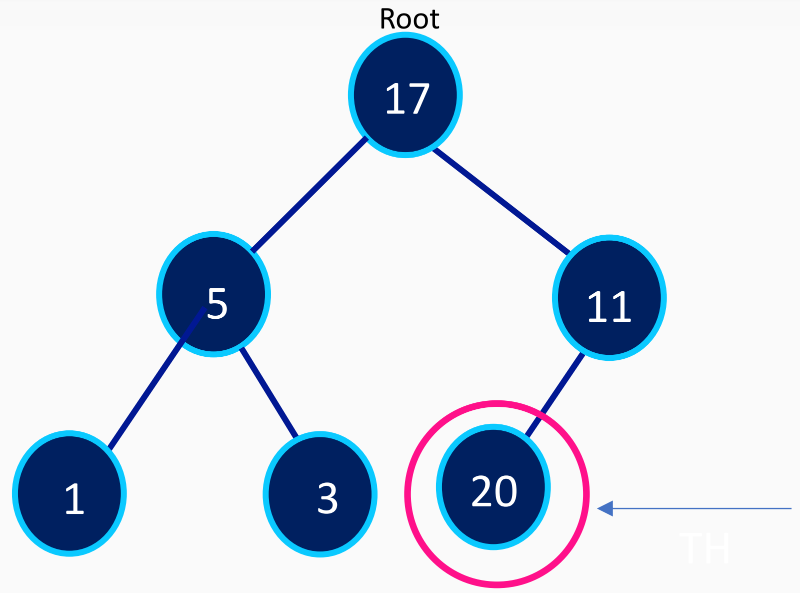


COMPLETE TREE ✗

Valid Heap?



First level is not filled!



This value is violating heap property.

COMPLETE TREE ✗

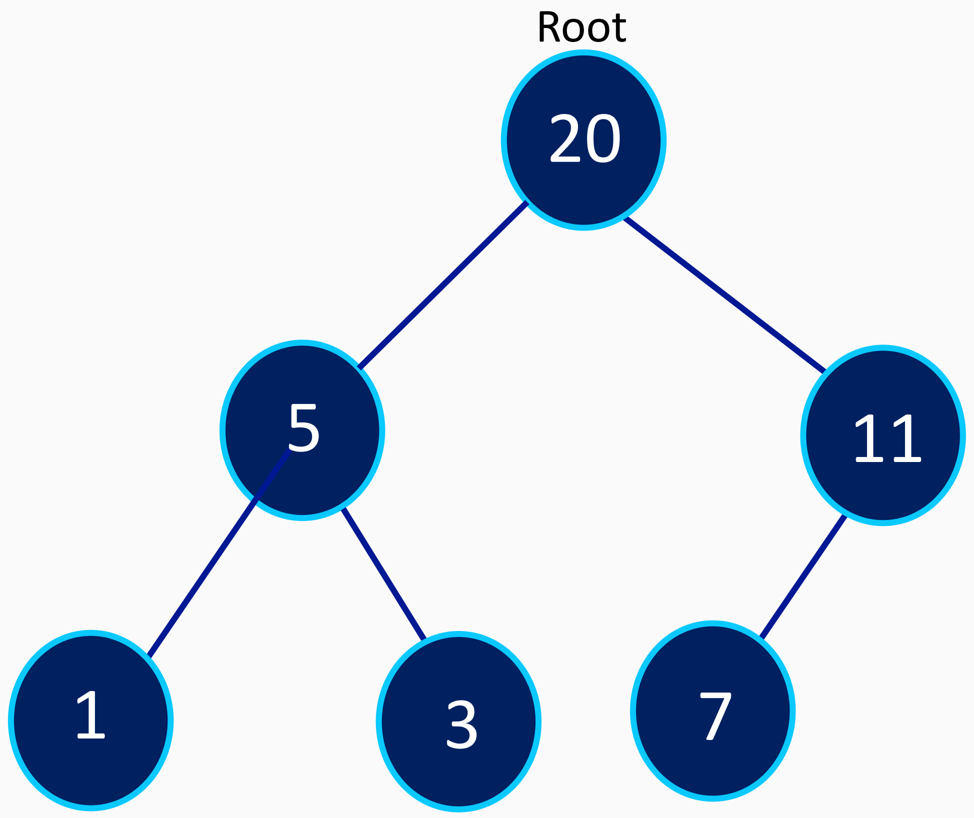
HEAP PROPERTY ?

COMPLETE TREE ✓

HEAP PROPERTY ✗

Value should be smaller than the parent!

Max Heap



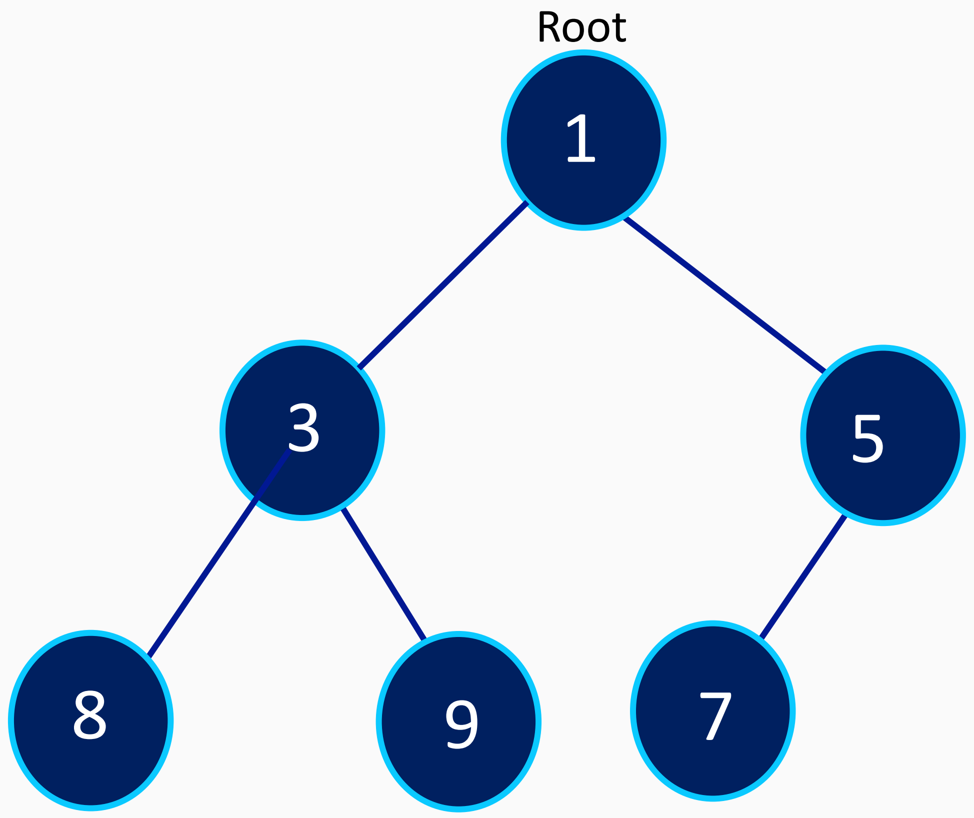
COMPLETE BINARY TREE ✓

HEAP PROPERTY ✓

MAX HEAP ✓

- Root value is the max value of heap : Max Heap
- All parents are greater than children

Min Heap



COMPLETE BINARY TREE ✓

HEAP PROPERTY ✓

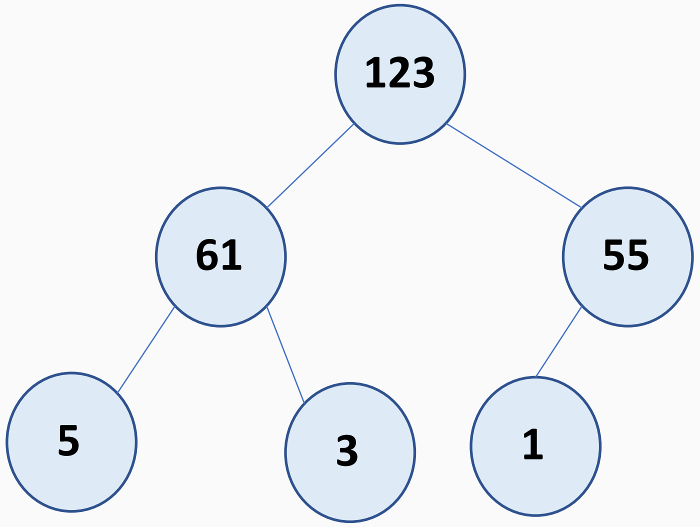
MIN HEAP ✓

- **Root value is the min value of heap : Min Heap**
- **All parents are smaller than children**

Max Heap vs Min Heap

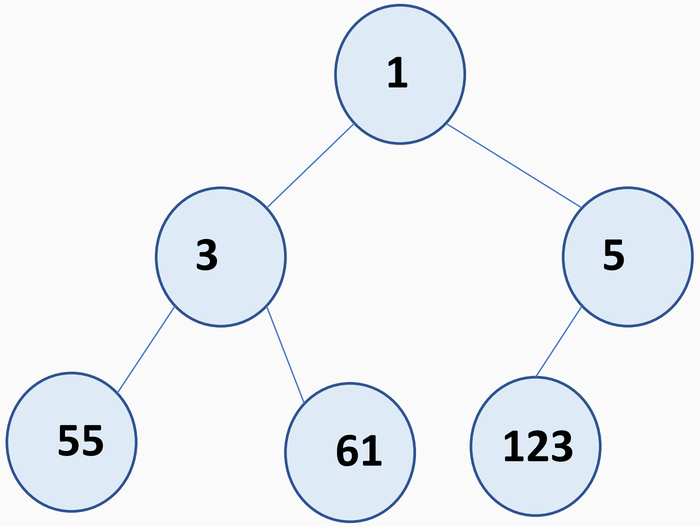
Max Heap

Every Node is less than or equal to its parent



Min Heap

Every Node is greater than or equal to its parent

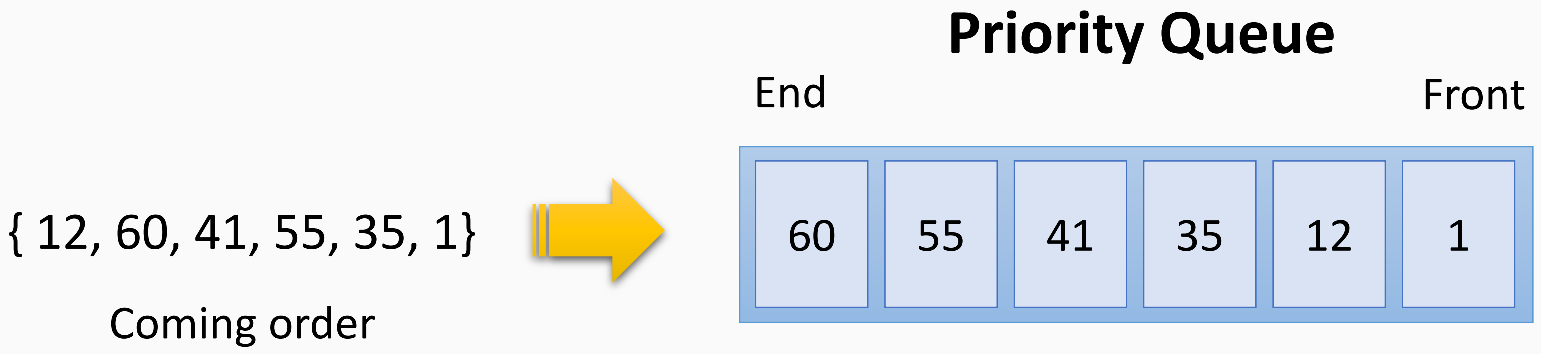


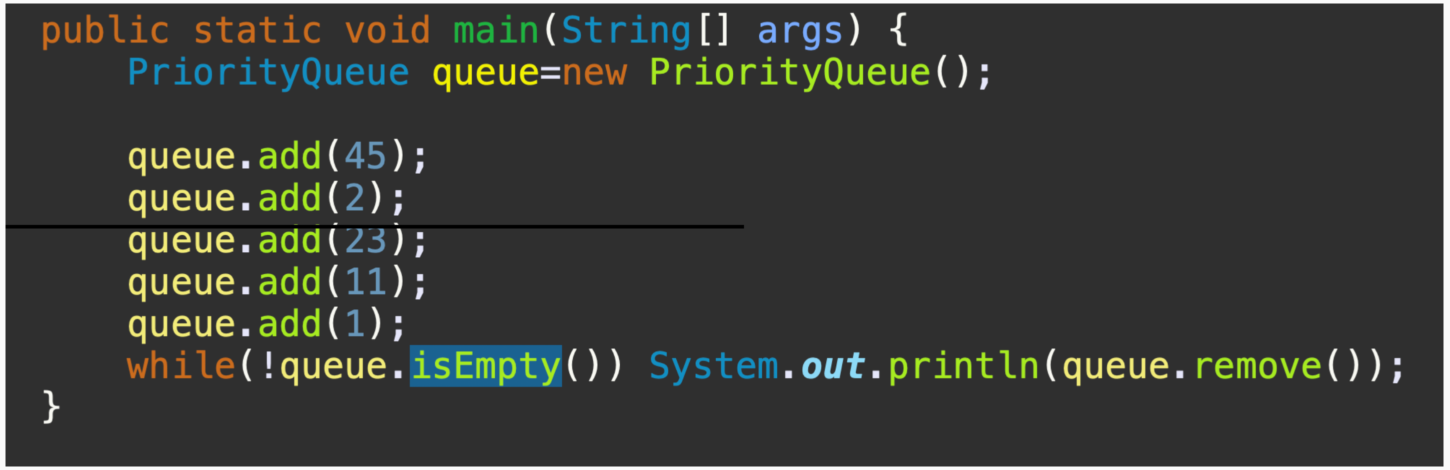
Where we use Heap Data Structure

- Sorting (HeapSort)
- Priority Queues
- Graph Algorithms (Djikstra’s shortest path)
- Selection Algorithms

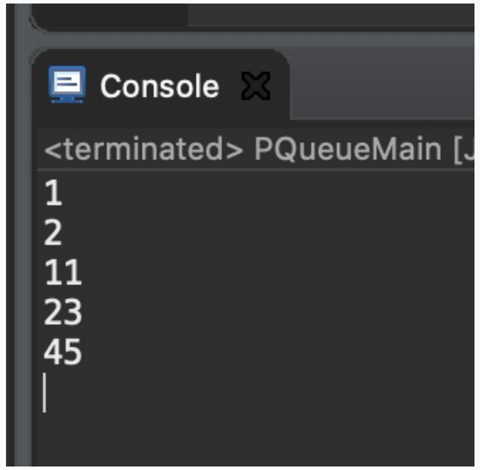
Priority Queue

- In priority queues, objects are processed based on their priorities, not the order they come to queue.





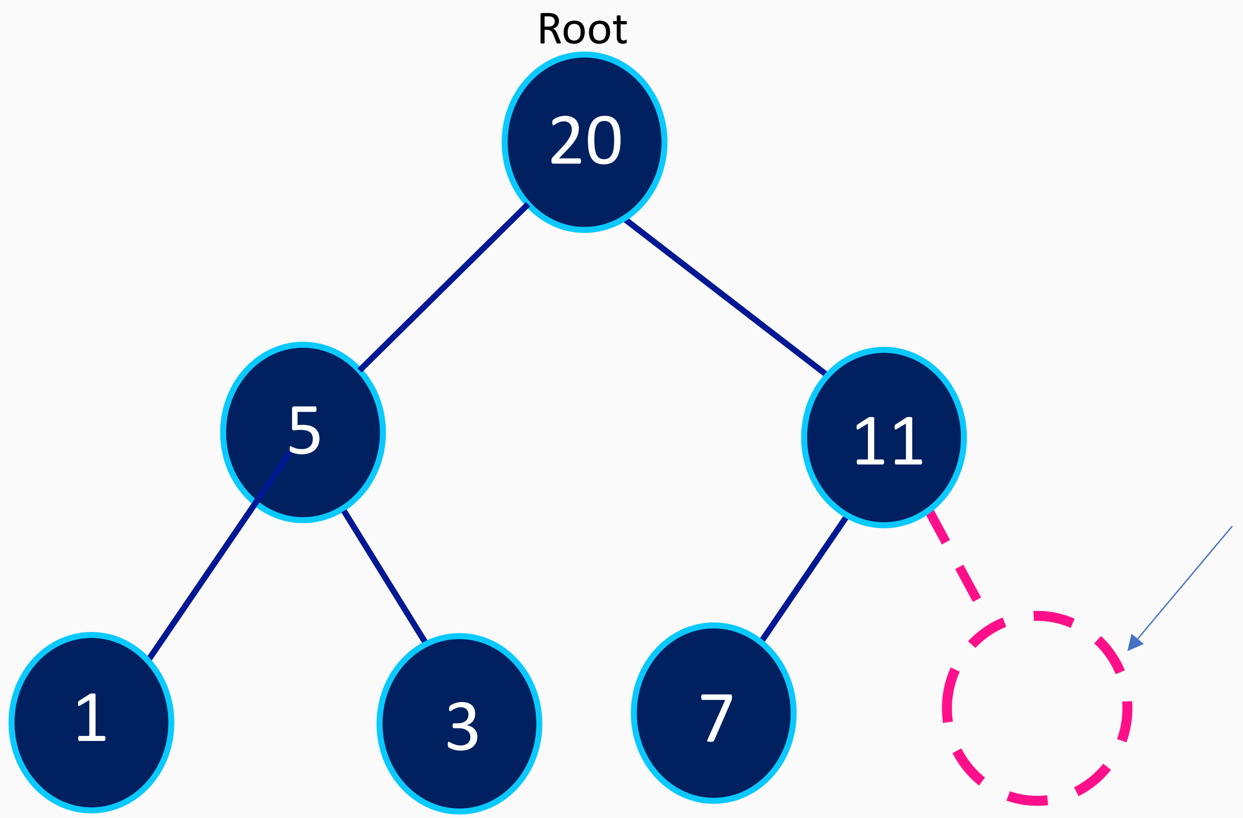
```
public static void main(String[] args) {
    PriorityQueue queue=new PriorityQueue();

    queue.add(45);
    queue.add(2);
    queue.add(23);
    queue.add(11);
    queue.add(1);
    while(!queue.isEmpty()) System.out.println(queue.remove());
}
```



```
Console
<terminated> PQueueMain [J
1
2
11
23
45
```

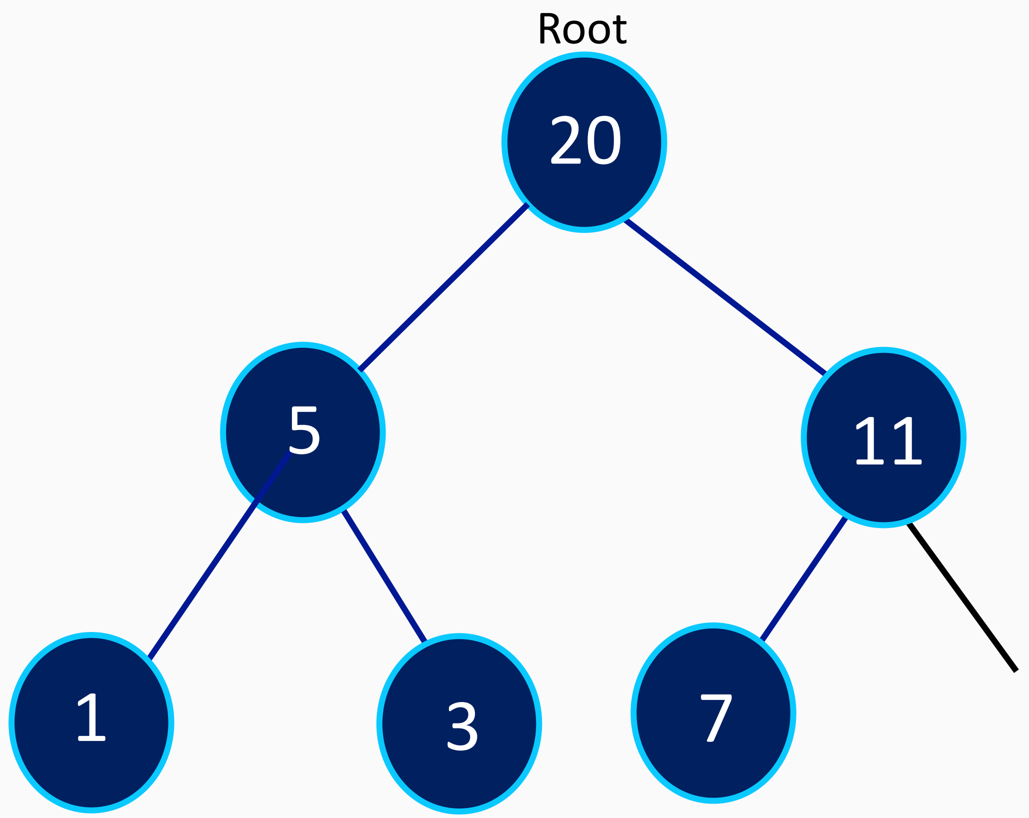
Operations on Heaps – Insertion to a Heap

Lets insert following value into the Heap:



- Place to insert new node !
- Whatever the value is !

Operations on Heaps – Insertion to a Heap



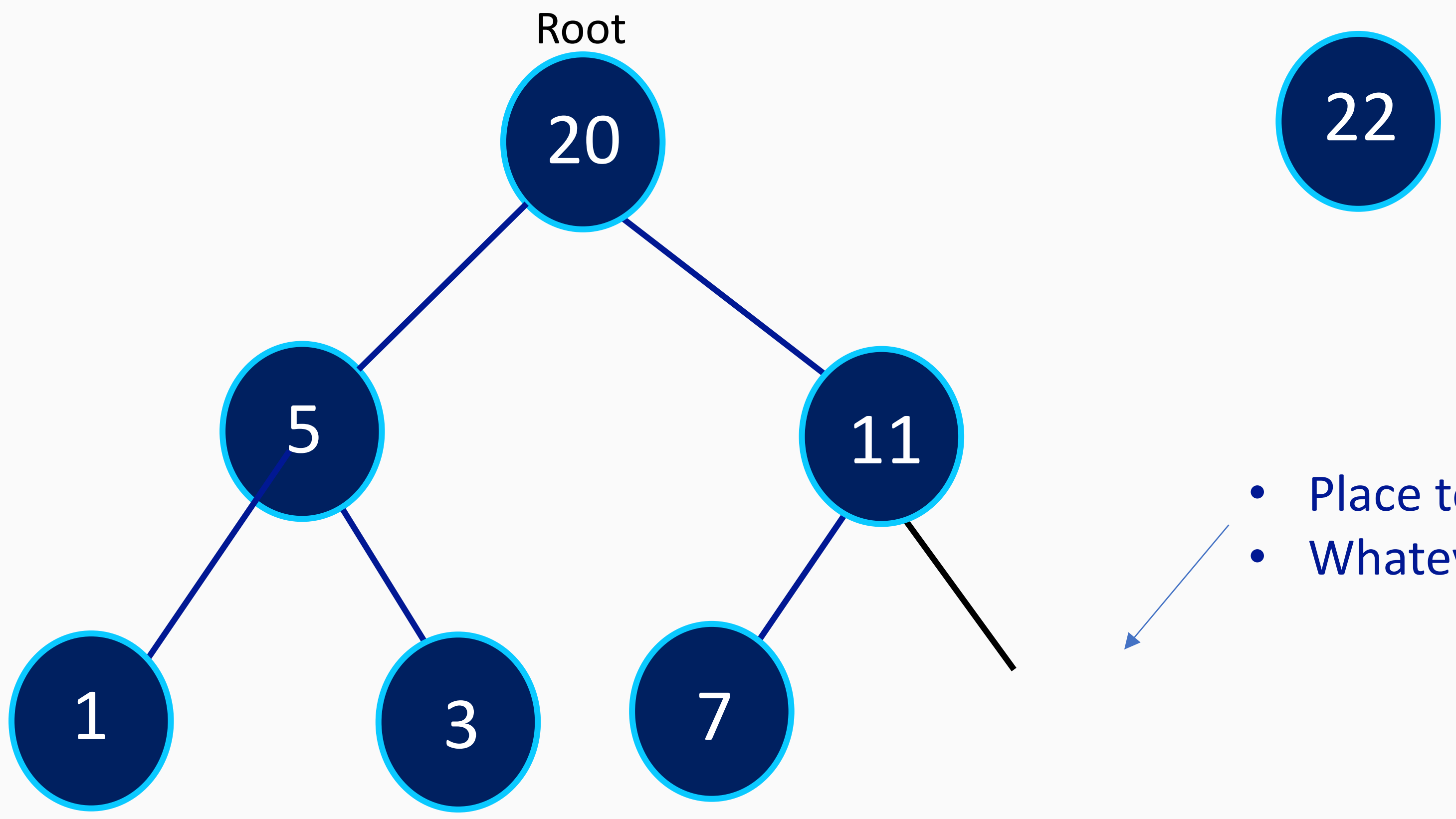
Lets insert following value into the Heap:

- Place to insert new node !
- Whatever the value is !

Violating Heap Property?

Operations on Heaps – Insertion to a Heap

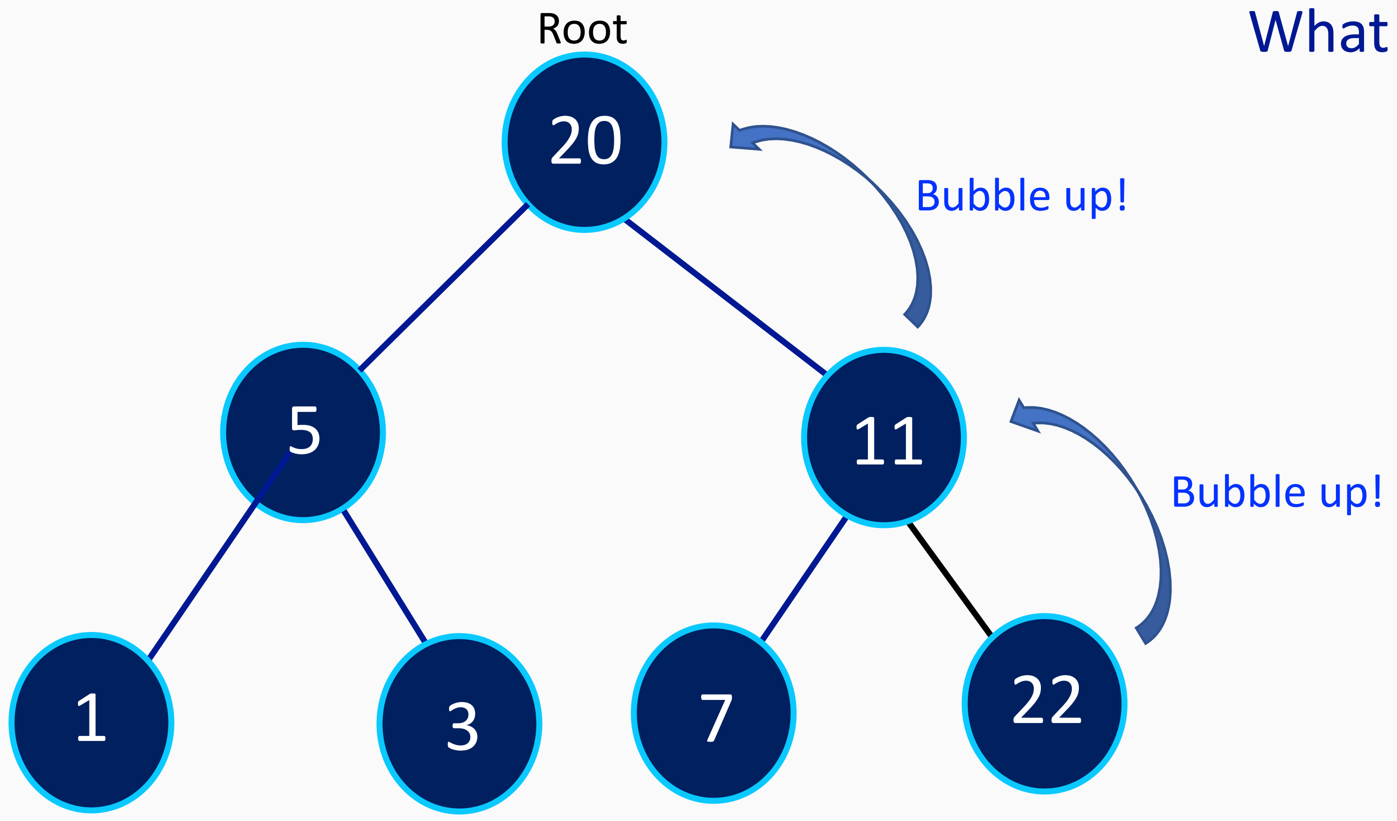
This time lets insert following value into the Heap:



- Place to insert new node !
- Whatever the value is !

Violating Heap Property

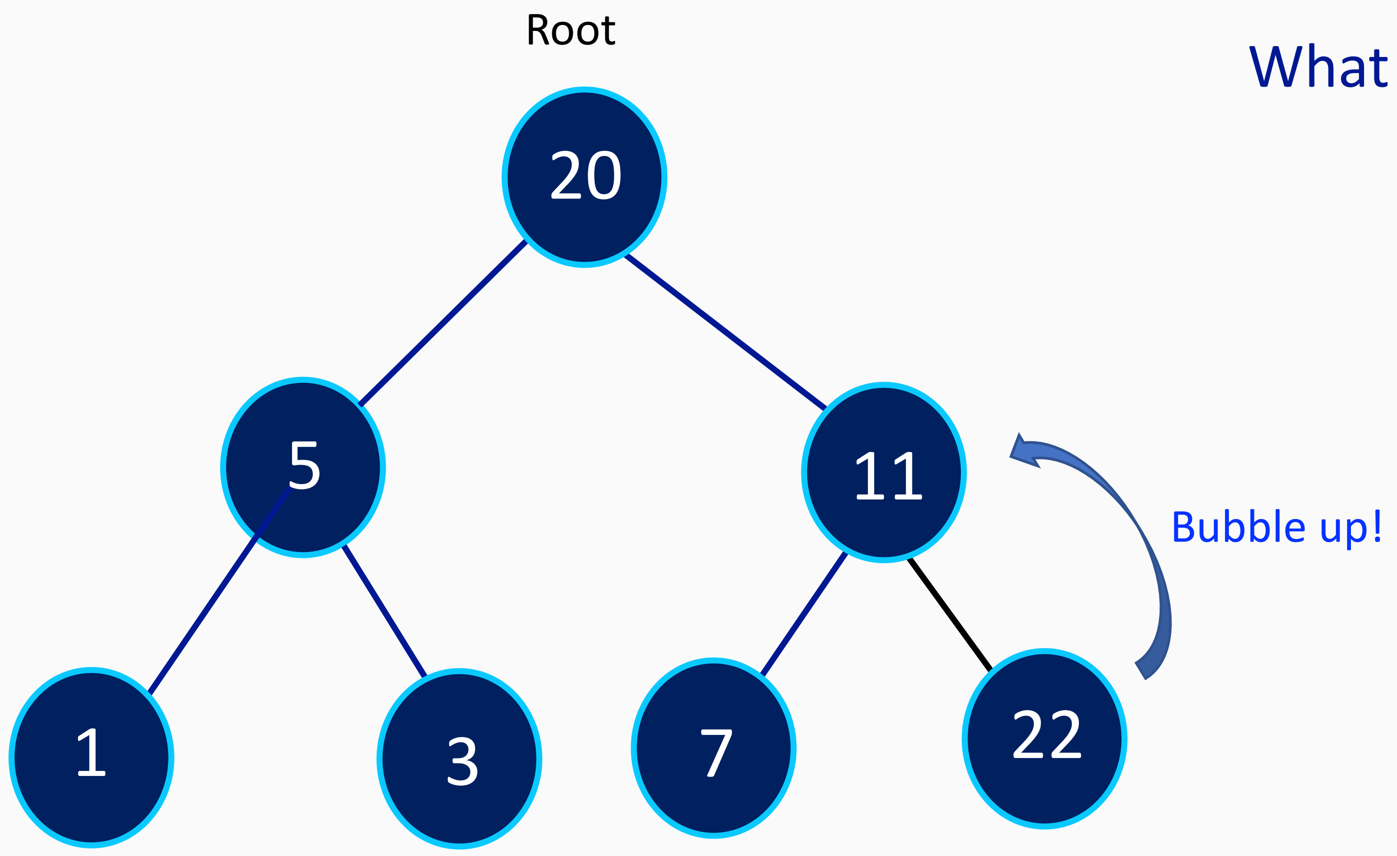
Operations on Heaps – Insertion to a Heap



What are we going to do?

Violating Heap Property

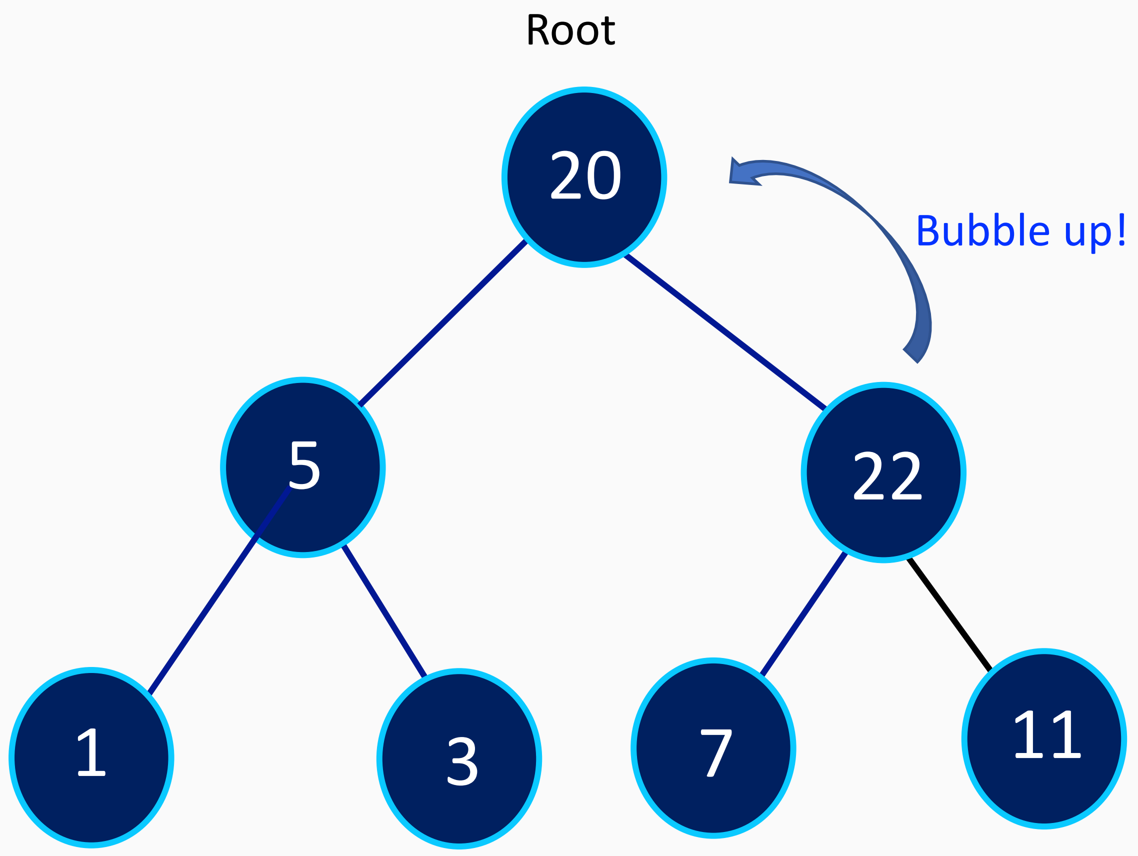
Operations on Heaps – Insertion to a Heap



What are we going to do?

Violating Heap Property

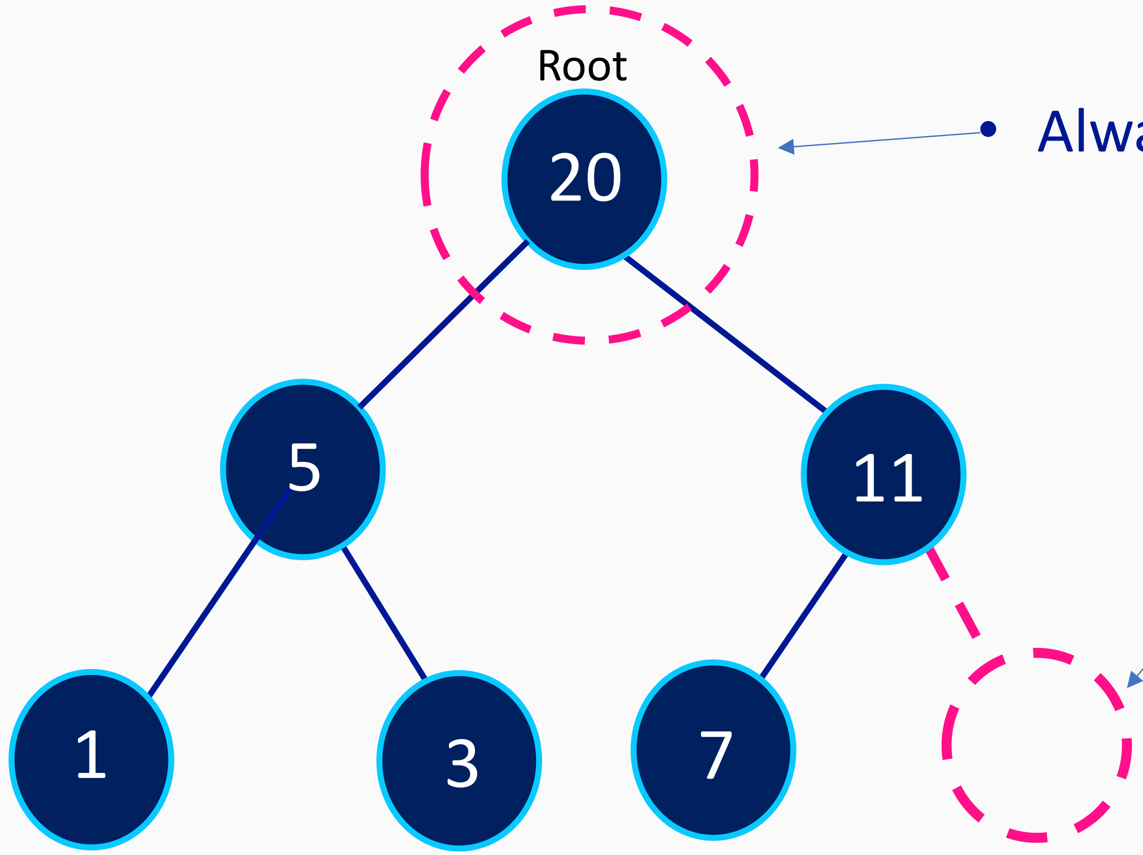
Operations on Heaps – Insertion to a Heap



What are we going to do?

Now I have the Heap Property

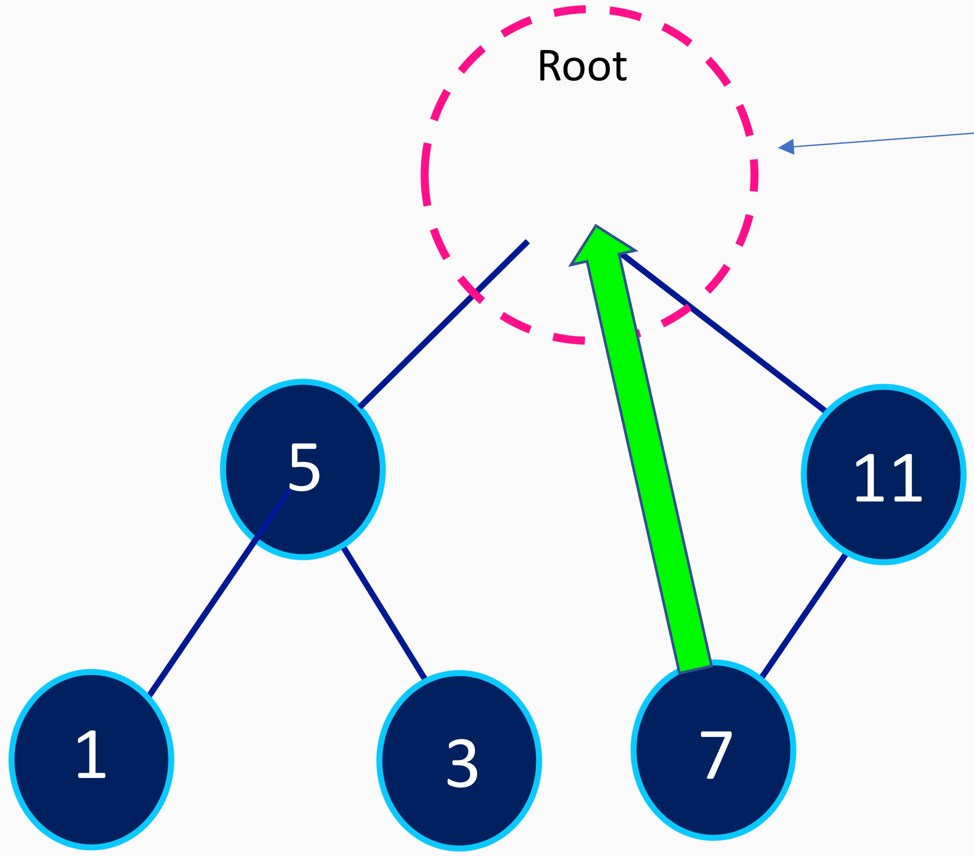
Operations on Heaps – Removing from a Heap



- Always remove the root !

- Place to insert new node !
- Whatever the value is !

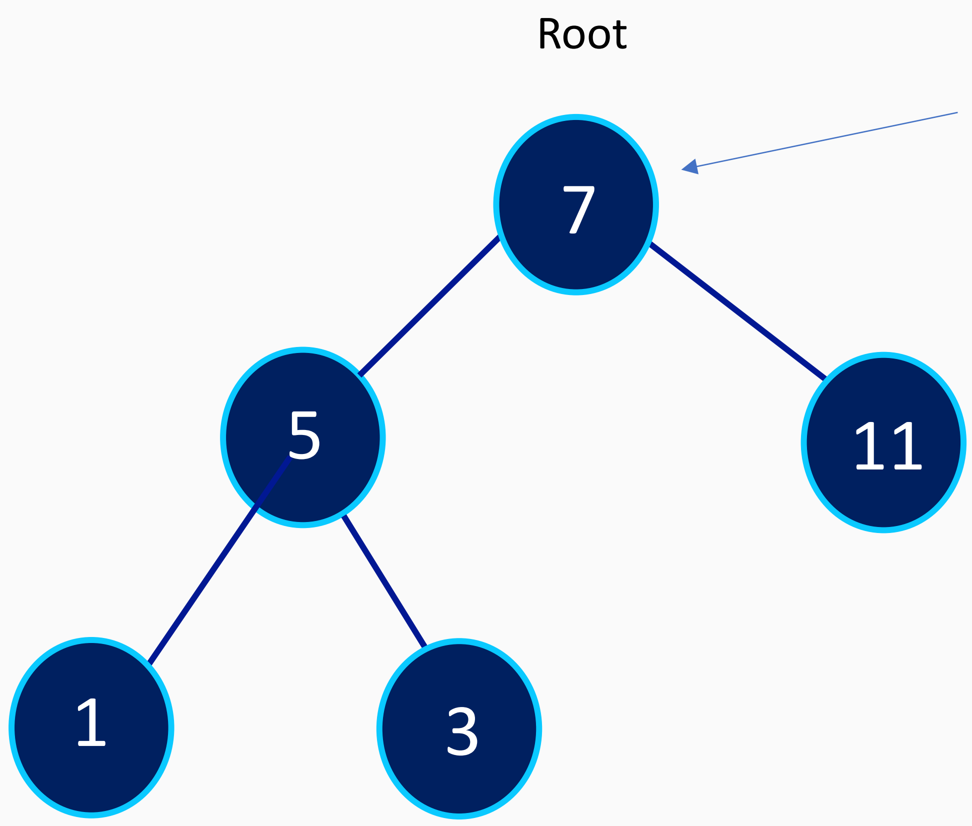
Operations on Heaps – Removing from a Heap



- Always remove the root !

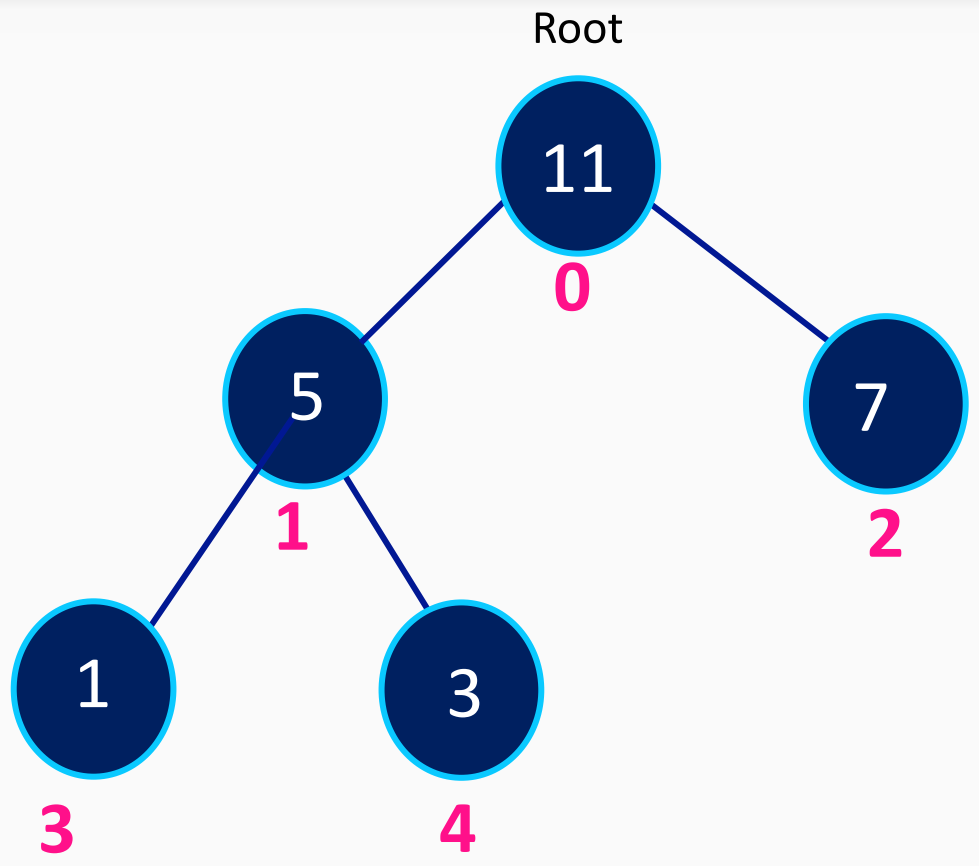
Move last item to root position
then
Check for the heap property!

Operations on Heaps – Removing from a Heap



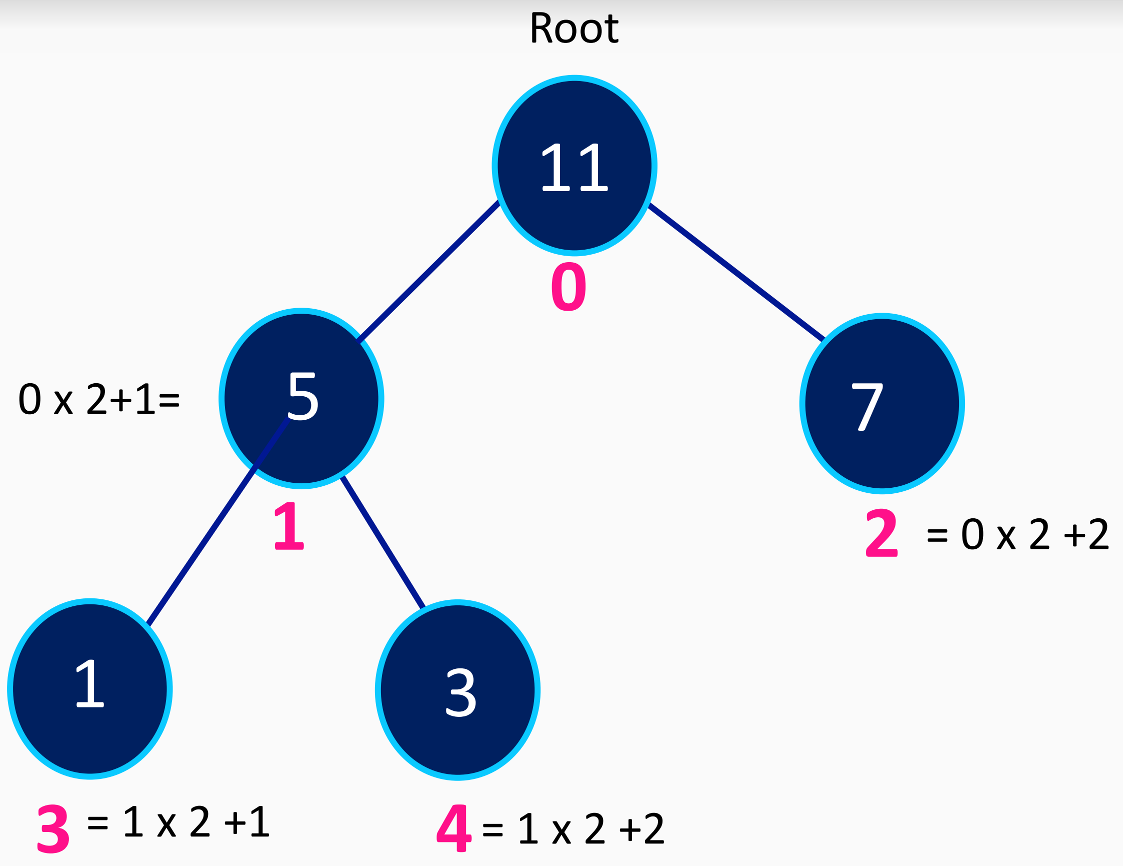
- Bubble down the root !
  - Swap with the greater child!

Heap Implementation



- Heaps can be implemented by using arrays as well as using a tree.
- Since its complete binary tree we don’t need a tree structure to implement it.

Heap Implementation



Formula:

Index of left = parent * 2 + 1
Index of right= parent * 2 + 2

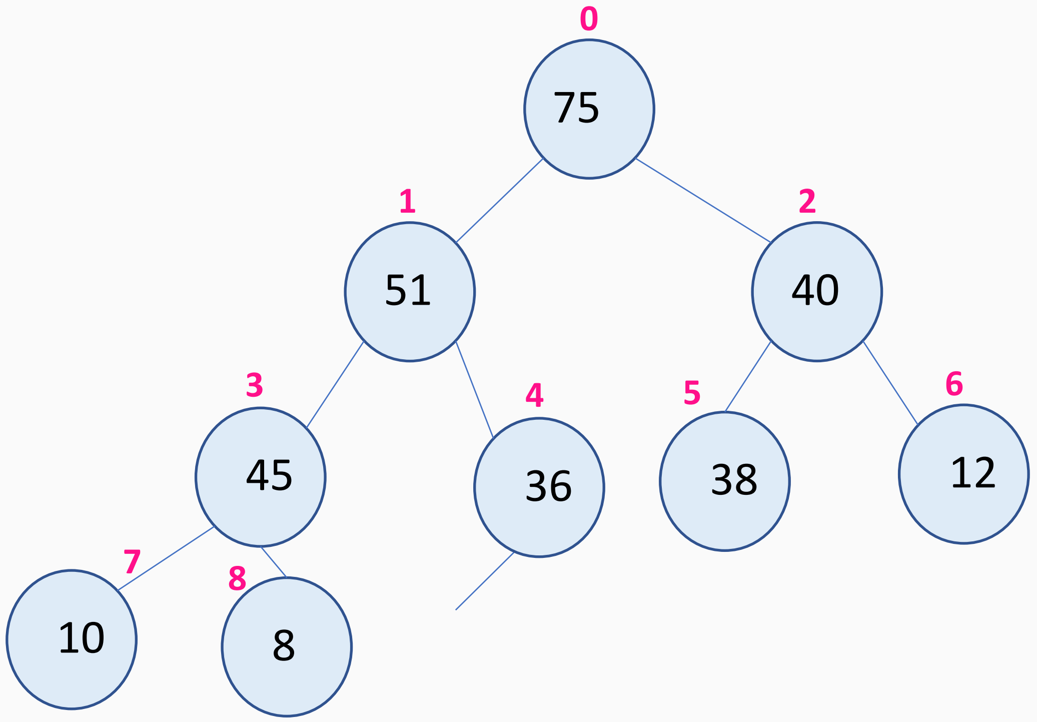
Parent= (index-1) / 2

Array=

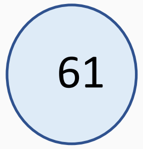
0	1	2	3	4
11	**5**	**7**	**1**	**3**

Insertion into Heap Example with Arrays

- Rule : - Maintain Heap property.
  - Insert new Node to the first available space in sequence.



Lets add this node to the heap.



0	1	2	3	4	5	6	7	8	9
75	51	40	45	36	38	12	10	8	61

Heap Implementation with Java

- Basic Integer Heap Implementation with Arrays
- Methods :
  - Insert(int value)
  - BubbleUp()
  - BubbleDown()
  - Remove()

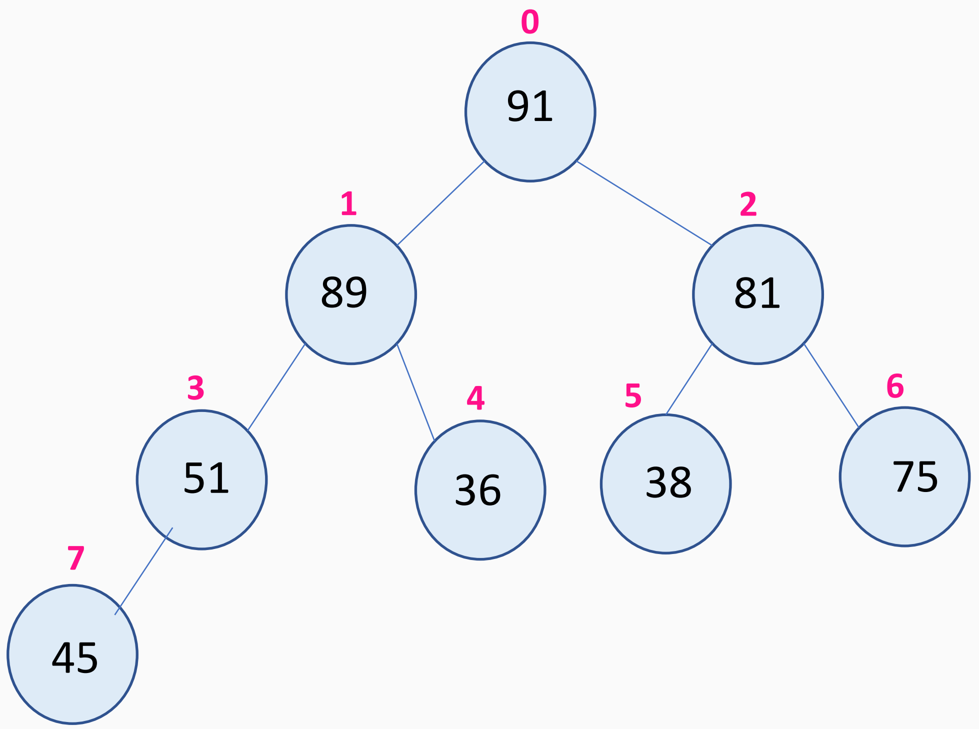
  - LeftChildIndex ()
  - RightChildIndex()
  - ParentIndex()
  - HasLeftChild()
  - HasRightChild()
  - IsValidParent()

Insertion into a Heap - Task

Question : Insert Array elements into a heap one by one, then remove the root.

Input Data =

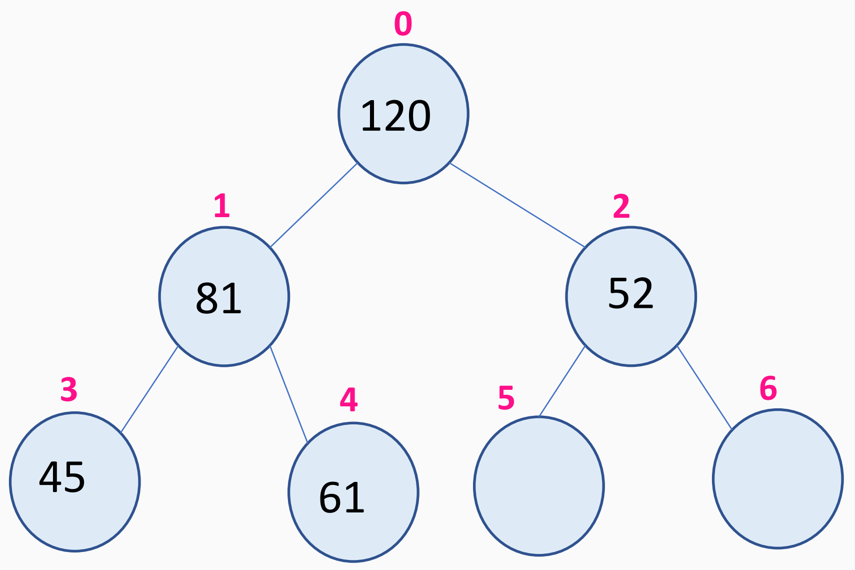
0	1	2	3	4	5	6	7	8	9
75	51	81	45	36	38	91	89		



Index= 7 = size

Heap=

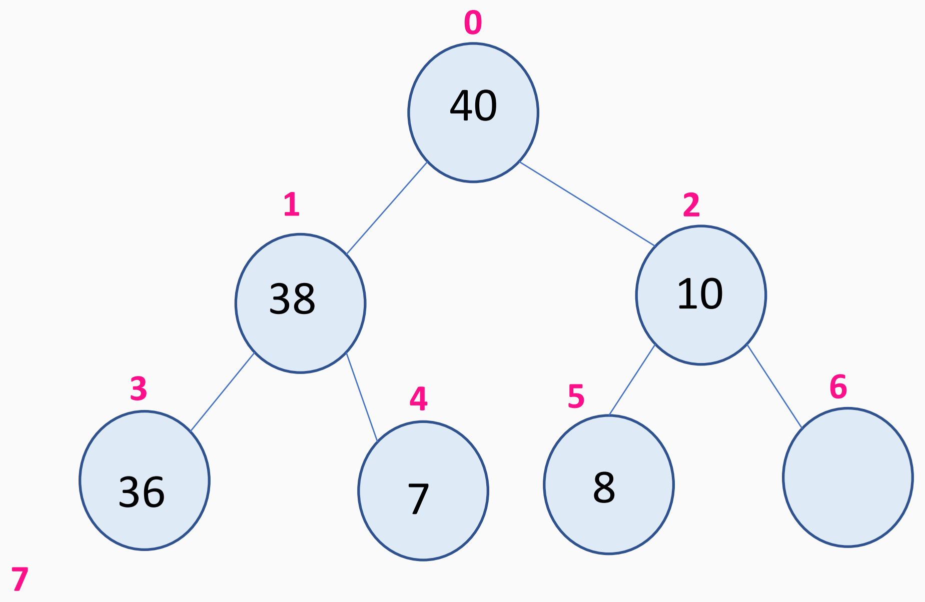
0	1	2	3	4	5	6	7	8	9
91	89	81	51	36	38	75	45		


0
120
1
81
2
52
3
45
4
61
5
6

Deletion from Heap Example with Arrays

- Rule : Can only delete/remove the root node.

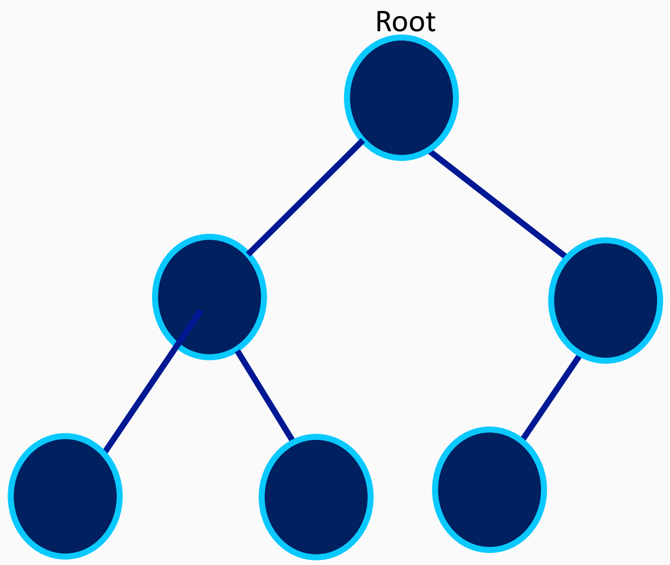
45



	0	1	2	3	4	5	6	7	8	9
Heap =	40	38	10	36	7	8				

Heap Performance

- For a max heap Return the max value is O(1),
- For Insertion/Deletion height is the factor. Height= log n so, O(log n)

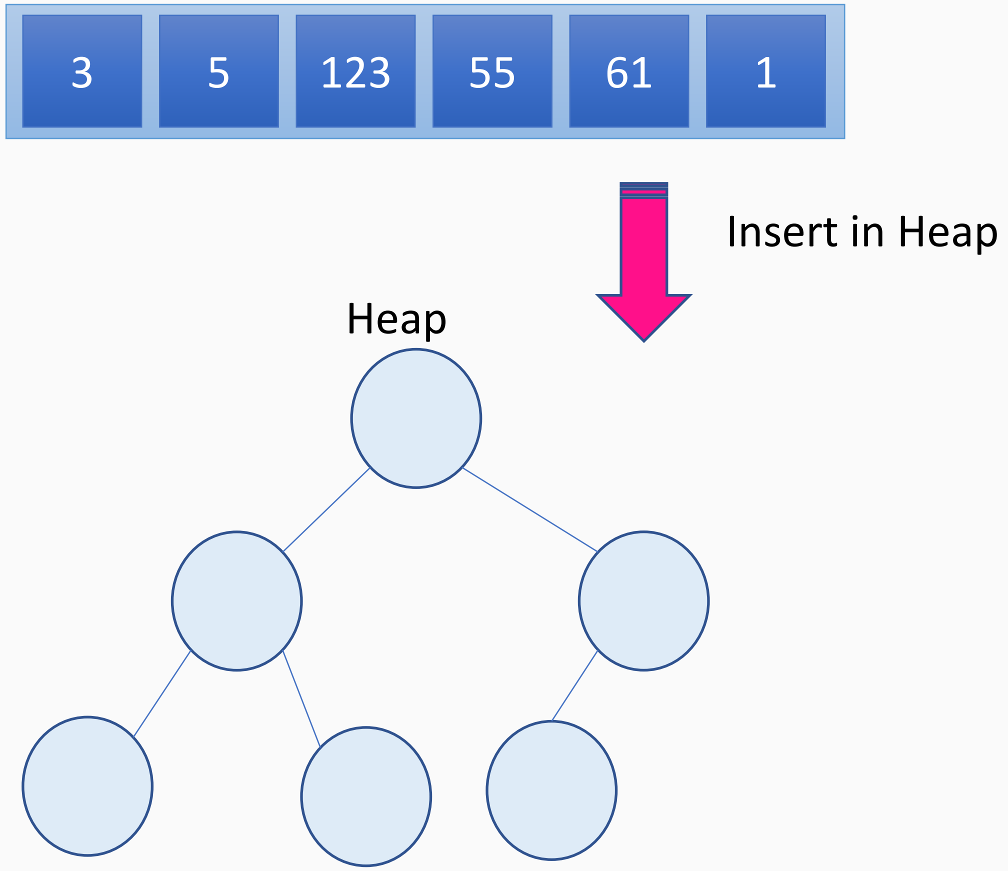


Heap Implementation with Java

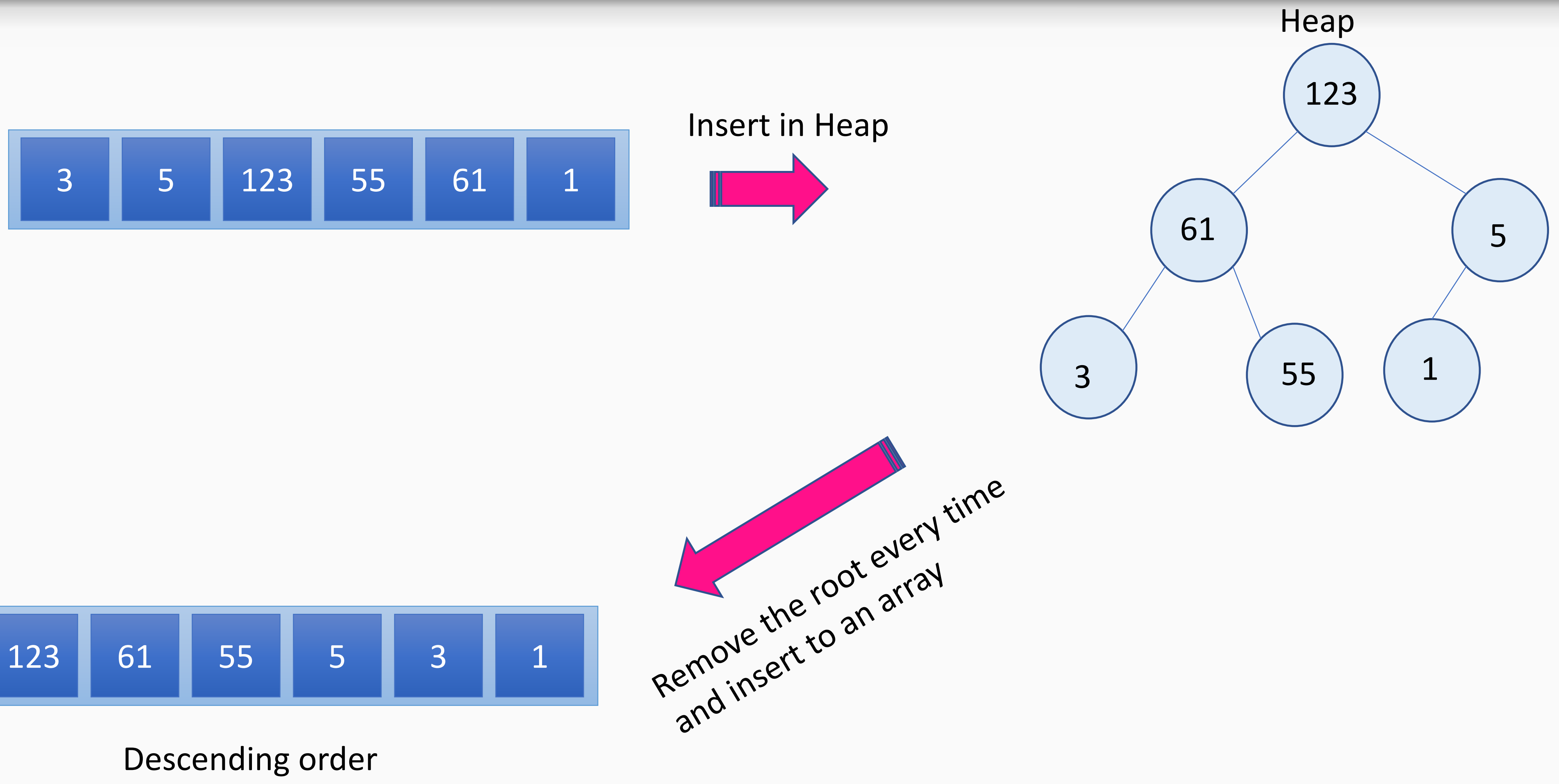
Lets switch to IntelliJ for our Heap Implementation

Heap Sort

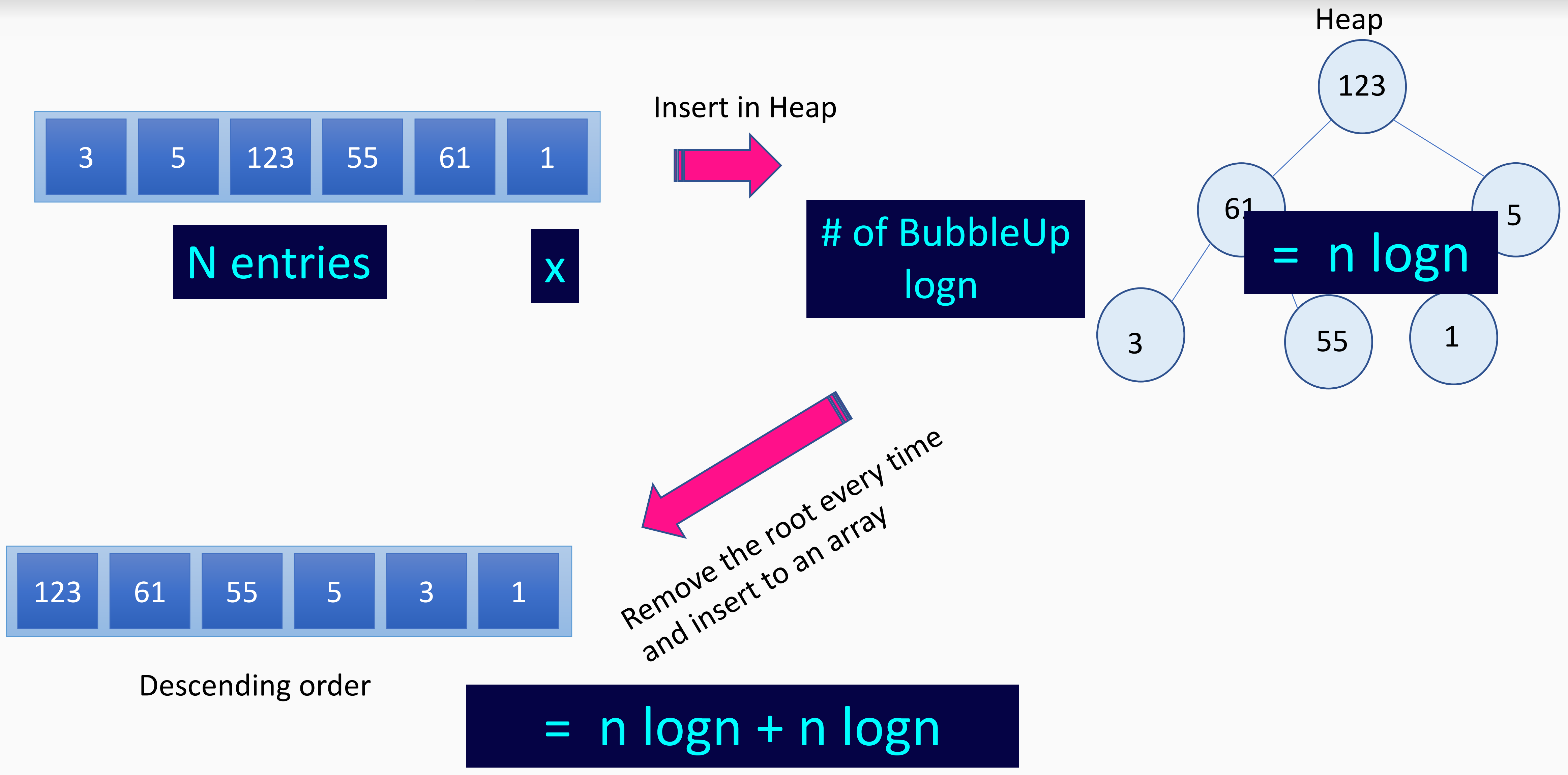
3 5 123 55 61 1



Heap Sort

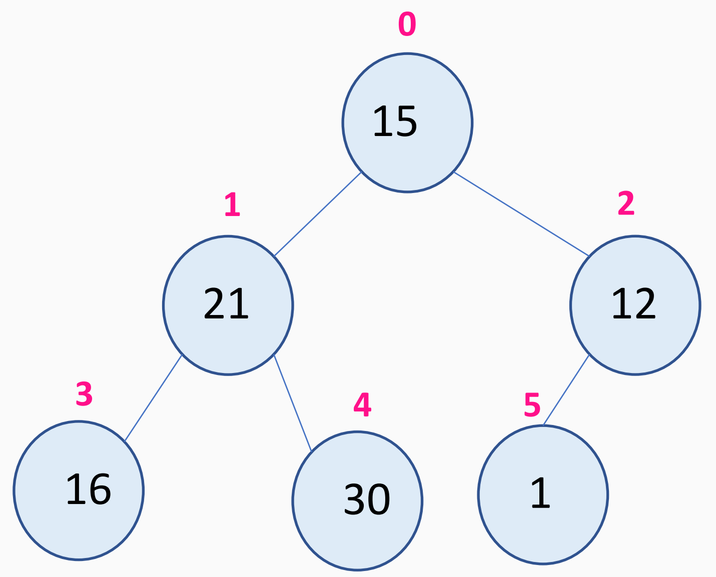

3
5
123
55
61
1
Insert in Heap
Heap
123
61
5
3
55
1
Remove the root every time and insert to an array
123
61
55
5
3
1
Descending order

Heap Sort- Performance Discussion



Heapify-Interview Question

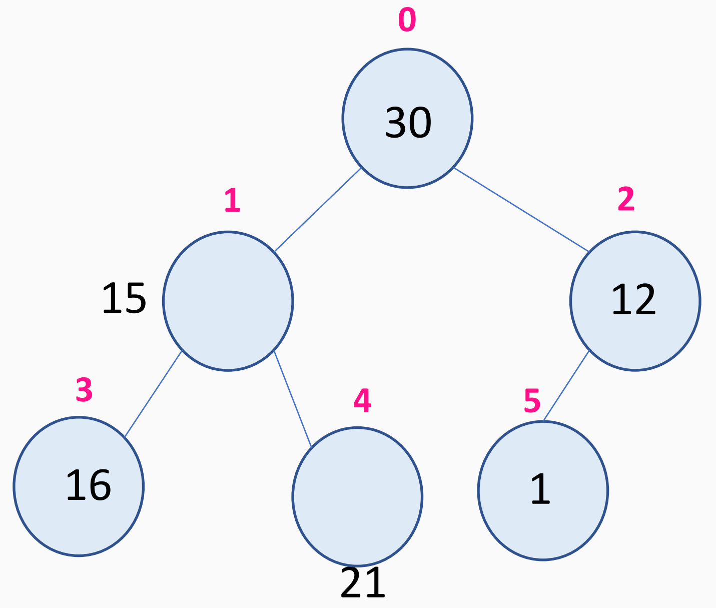
- Converting an array to a heap in place.
- Why we need Heapify?



0	1	2	3	4	5	6	7	8	9
15	21	12	16	30	1				

Heapify-Interview Question

- <u>Algorithm:</u>
    1. Start from (n/2 -1) to index 0; (To exclude leaves)
    2. Compare the greatest of children with the parent
       - if parent<child then bubbleUp(theChild)
    3. Continue until indexth element is in place

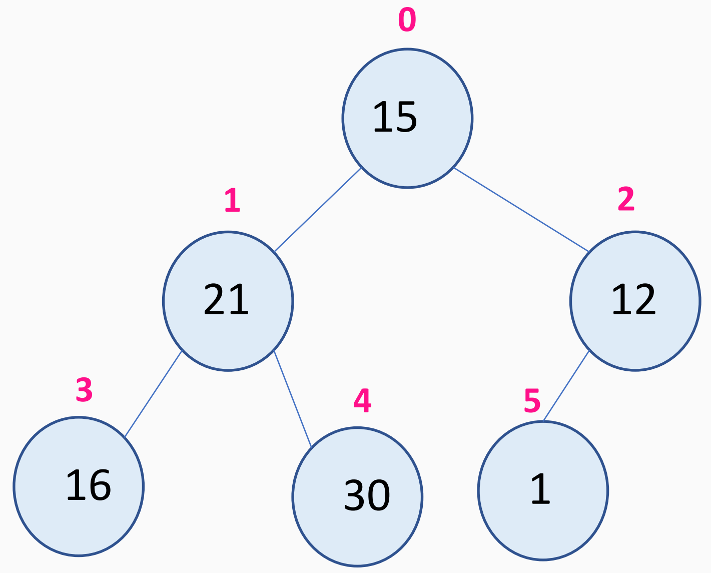


0	1	2	3	4	5	6	7	8	9
15	21	12	16	30	1	null	null	null	null

size=6

Heapify-Interview Question

- **Algorithm:**
  1. Start from (n/2 -1) to index 0; (To exclude leaves)
  2. Compare the greatest of children with the parent
     - if (child>parent) then bubbleUp(theChild)
  3. Continue until indexth element is in place



0	1	2	3	4	5	6
15	21	12	16	30	1	null

Heap Assignment

Kth Largest Element in an Array

Medium

- Given an integer array nums and an integer k, return *the* kth *largest element in the array*.
- Note that it is the kth largest element in the sorted order, not the kth distinct element.

Example 1:

- **Input:** nums = [3,2,1,5,6,4], k = 2 **Output:** 5

Example 2:

- **Input:** nums = [3,2,3,1,2,4,5,5,6], k = 4 **Output:** 4

Constraints:

- 1 <= k <= nums.length <= 10^4
- -10^4 <= nums[i] <= 10^4

Questions?