Lesson:

Heaps

Pre Requisites:

- Tree data structure

List of concepts involved :

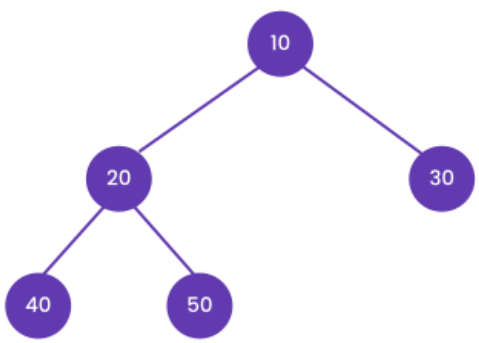
- What is a heap ?
- Min heap
- Max heap
- Insertion in a heap
- Deletion in a heap
- Priority Queue
- Kth smallest element using priority queue

What is Heap?

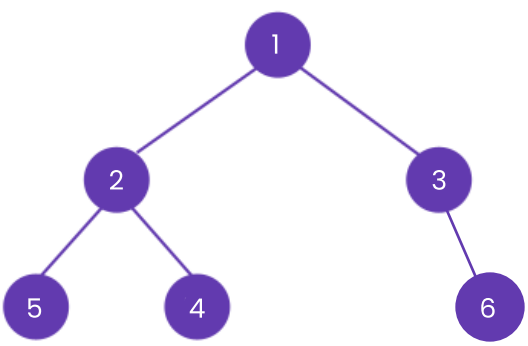
A heap is a complete binary tree, and the binary tree is a tree in which the node can have utmost two children.

What is a complete binary tree?

A complete binary tree is a **binary tree** in which all the levels except the last level, i.e., leaf node should be completely filled, and all the nodes should be left-justified. For example,



In the above figure, we can observe that all the internal nodes are completely filled except the leaf node; therefore, we can say that the above tree is a complete binary tree.



The above figure shows that all the internal nodes are completely filled except the leaf node, but the leaf nodes are added at the right part; therefore, the above tree is not a complete binary tree.

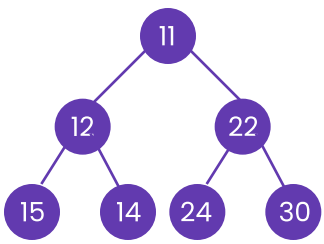
Types of heaps :

- Broadly there are two types of heaps:
  - Minheap
  - Maxheap

Minheap : The value of the parent node should be less than or equal to either of its children. In other words, the min-heap can be defined as, for every node i, the value of node i is greater than or equal to its parent value except the root node. Mathematically, it can be defined as:

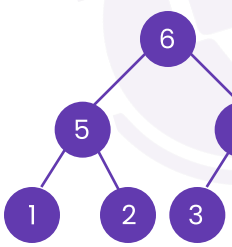
A[Parent(i)] <= A[i]

Let's understand the min-heap through an example.



In the above figure, 11 is the root node, and the value of the root node is less than the value of all the other nodes (left child or a right child).

Max Heap: The value of the parent node is greater than or equal to its children. In other words, the max heap can be defined as for every node i; the value of node i is less than or equal to its parent value except the root node. Mathematically, it can be defined as:
A[Parent(i)] >= A[i]



The above tree is a max heap tree as it satisfies the property of the max heap. Now, let's see the array representation of the max heap.

Insertion in a heap :

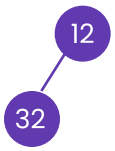
12, 32 , 10 , 15

Suppose we want to create the max heap tree. To create the max heap tree, we need to consider the following two cases:

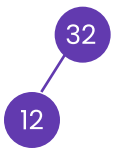
- First, we have to insert the element in such a way that the property of the complete binary tree must be maintained.
- Secondly, the value of the parent node should be greater than either of its children.

Step 1: First we add element 12 in the tree as shown below.

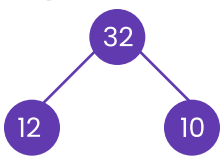
Step 2 : The element is 32. We will firstly add it as the left child of 12.



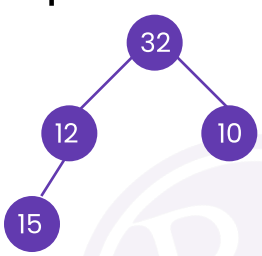
Step 3: Currently this is violating the condition of maxheap. So we will swap the two elements.



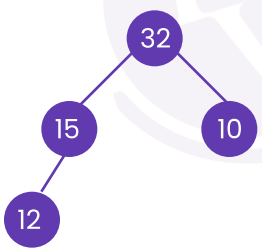
Step 4: The next element is 10. So we will add it as the right child of this tree.



Step 5 : Next element is 15. It will be added as the left child of 12.



Step 6 : Again this is violating the condition of maxheap so nodes with values 12 and 15 will be swapped.



This is how insertion in a maxheap works.

For a minheap, every time a node is inserted, one property will be maintained that any parent node must have a minimum value than its children nodes.

Generalizing insertion in a heap :

Step 1 - Create a new node at the end of the heap.
Step 2 - Assign new value to the node.
Step 3 - Compare the value of this child node with its parent.
Step 4 - If the value of the parent is less than the child, then swap them.
Step 5 - Repeat step 3 & 4 until Heap property holds.

Deletion in a heap :

Step 1 - Remove the root node.
Step 2 - Move the last element of the last level to root.
Step 3 - Compare the value of this child node with its parent.
Step 4 - If the value of the parent is less than the child, then swap them.
Step 5 - Repeat step 3 & 4 until Heap property holds.

Priority Queue :

A PriorityQueue is used when the objects are supposed to be processed based on the priority. It is known that a queue follows the First-In-First-Out algorithm, but sometimes the elements of the queue need to be processed according to the priority, that's when the PriorityQueue comes into play. The PriorityQueue is based on the heap. The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.

Syntax :
`PriorityQueue<E> pq = new PriorityQueue<E>();`

Q1. Given an integer array, find the kth smallest element using priority queue.

Input 1 : arr[] = {1,2,3,5,2,6,9} k = 3
Output 1: 2

Input 2 : arr[] = {1,7,8,5,2,6,9} k = 6
Output 2: 8

Solution :

Code : LP_Code1.java

Output :

```
k'th smallest array element is 4
```

Approach :

- **Method 1 :** Using minheap,
  - Insert all elements into a minheap.
  - Remove first (k-1) elements.
  - The element at the top of the heap is the kth smallest element

Time complexity : O(n + k(logn)) where n is the number of elements in the array.
Space complexity : O(n) because n elements are inserted in the heap.

- **Method 2 :** Using maxheap,
  - Insert first k elements into a maxheap.
  - for remaining n-k elements if the ith element is lesser than the top element of the queue, remove it from the top and insert this current element into the queue.
  - If the ith element is greater then do nothing and move to the next iteration.
  - Since the element at the top will never be considered in k smallest element until another element smaller than this peek element is outside the queue.
  - repeat this process until all the elements are traversed.
  - This way the kth smallest element will be at the top of the queue in the end.

Time complexity : O(k+nlogk)
Space complexity : O(k) because in the worst case queue will be only holding k elements.

Code : LP_Code2.java

Output :

```
k'th smallest array element is 4
```