Lesson:

Priority Queue

Pre Requisites:

- Tree data structure

List of concepts involved :

- Heap sort algorithm
- k closest points to origin problem
- Merge k sorted lists

Heap Sort Algorithm

- First convert the array into heap data structure using heapify, then one by one delete the root node of the Max-heap.
- Now replace it with the last node in the heap and then heapify the root of the heap.
- Repeat this process until the size of the heap is greater than 1.
- Build a heap from the given input array.

Repeat the following steps until the heap contains only one element:
a. Swap the root element of the heap (which is the largest element) with the last element of the heap.
b. Remove the last element of the heap (which is now in the correct position).
c. Heapify the remaining elements of the heap.

The sorted array is obtained by reversing the order of the elements in the input array.

Follow the given steps to solve the problem:

- Build a max heap from the input data.
- At this point, the maximum element is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of the heap by 1. Finally, heapify the root of the tree.
- Repeat step 2 while the size of the heap is greater than 1.

Note: The heapify procedure can only be applied to a node if its children nodes are heapified. So the heapification must be performed in the bottom-up order.

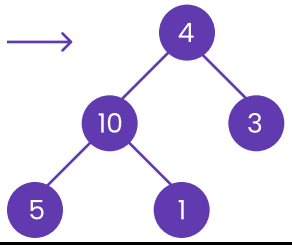
Detailed Working of Heap Sort :

To understand heap sort more clearly, let's take an unsorted array and try to sort it using heap sort.
Consider the array: arr[] = {4, 10, 3, 5, 1}.

Build Complete Binary Tree: Build a complete binary tree from the array.

Step 1 Build complete Binary Tree

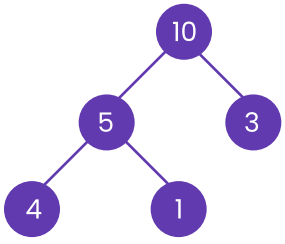
arr = {4, 10, 3, 5, 1} ⟶



Transform into max heap: After that, the task is to construct a tree from that unsorted array and try to convert it into maxheap.

- To transform a heap into a max-heap, the parent node should always be greater than or equal to the child nodes
- Here, in this example, as the parent node 4 is smaller than the child node 10, thus, swap them to build a max-heap.
- Now, as seen, 4 as a parent is smaller than the child 5, thus swap both of these again and the resulted heap and array should be like this:

Step 2 Make it a max heap (4 less than 5 & 5 is greater between th two children)



arr = {10, 5, 3, 4, 1}

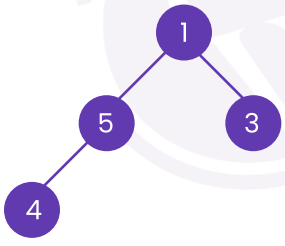
Make the tree a max heap

Perform heap sort: Remove the maximum element in each step (i.e., move it to the end position and remove that) and then consider the remaining elements and transform it into a max heap.

- Delete the root element (10) from the max heap. In order to delete this node, try to swap it with the last node, i.e. (1). After removing the root element, again heapify it to convert it into max heap.
  - Resulted heap and array should look like this:

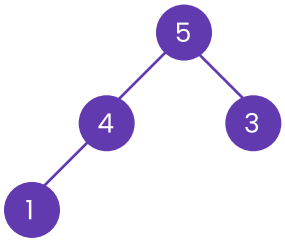
Step 3 Remove the max(10) & heapify

- Remove the max (i.e., move it to the end)



arr = {1, 5, 3, 4, 10}

- Heapify

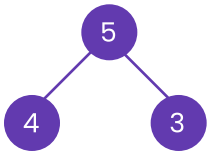


arr = {5, 4, 3, 1, 10}

- Repeat the above steps and it will look like the following:

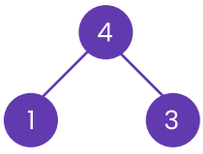
Step 4 Remove the current max(5) & heapify

- Remove the max (i.e., move it to the end)



arr = {1, 4, 3, 5, 10}

- Heapify

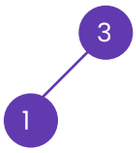


arr = {4, 1, 3, 5, 10}

- Now remove the root (i.e. 3) again and perform heapify.

Step 5 Remove the current max(4) & heapify

- Remove the max (i.e., move it to the end)



arr = {3, 1, 4, 5, 10}

It is already in max heap form

- Now when the root is removed once again it is sorted. and the sorted array will be like **arr[] = {1, 3, 4, 5, 10}.**

Step 6 Remove the max(3)

- Remove the max (i.e., move it to the end)

arr = {1, 3, 4, 5, 10}
The array is now sorted

Code 1 : LP_Code1.java

Output :

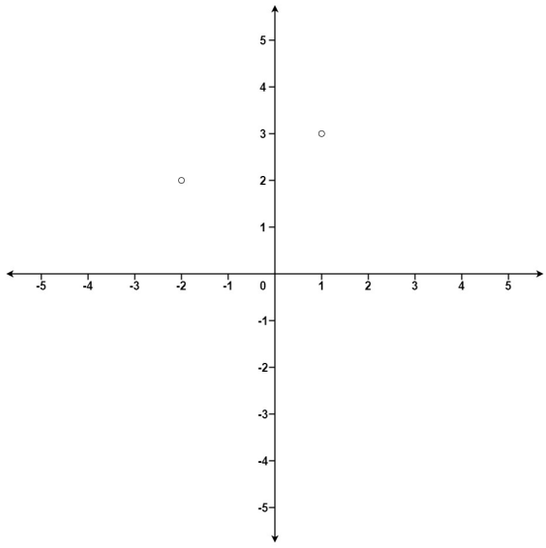
```
Sorted array is
3 4 5 8 9 12
```

k closest points to origin :

Q1. Given an array of points where points[i] = [xi, yi] represents a point on the X-Y plane and an integer k, return the k closest points to the origin (0, 0). The distance between two points on the X-Y plane is the Euclidean distance (i.e., √(x1 - x2)2 + (y1 - y2)2).

You may return the answer in any order.

Example 1:



Input: points = [[1,3],[-2,2]], k = 1
Output: [[-2,2]]

Explanation:
The distance between (1, 3) and the origin is sqrt(10).
The distance between (-2, 2) and the origin is sqrt(8).
Since sqrt(8) < sqrt(10), (-2, 2) is closer to the origin.

We only want the closest k = 1 points from the origin, so the answer is just [[-2,2]].

Example 2:
Input: points = [[3,3],[5,-1],[-2,4]], k = 2
Output: [[3,3],[-2,4]]

Explanation: The answer [[-2,4],[3,3]] would also be accepted.

Solution :

Code : LP_Code2.java

Output :

```
The desired output is :
4 1
2 3
1 2
```

Approach :

- Since we want K smallest elements, we use a K-size max heap.
- We repeatedly add elements to the heap.
- When the size is greater than K, we remove the largest element.

Merge k sorted lists :

Q2. You are given an array of k linked-lists lists, each linked-list is sorted in ascending order. Merge all the linked-lists into one sorted linked-list and return it.

Example 1:
Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]

Explanation: The linked-lists are:
[
1->4->5,
1->3->4,
2->6
]
merging them into one sorted list:
1->1->2->3->4->4->5->6

Example 2:
Input: lists = []

Output: []

Example 3:
Input: lists = [[]]
Output: []

Solution :

Code : LP_Code3.java

Output :

```
0 1 2 3 4 5 6 7 8 9 10 11
```

Approach :

- Make use of the fact that linked lists are sorted. We'll use a min priority queue.
- Since we know that the linked list is sorted , we first push heads of all the lists in the priority queue.
- The minimum of these will be the first node of the list.
- After that we move this min node to its next (if it exists) and push that next node again to the priority queue.
- This way we keep adding nodes to the resultant list and the moment the priority queue becomes empty, we get our resulting list.