PRIORITY QUEUES -1

Code : Remove Min

Send Feedback

Implement the function RemoveMin for the min priority queue class.
For a minimum priority queue, write the function for removing the minimum element present.
Remove and return the minimum element.

Note : main function is given for your reference which we are using internally to test the code.

```
/*
	Implement the function RemoveMin for the min priority queue class.
	For a minimum priority queue, write the function for removing the minimum element
	present. Remove and return the minimum element.
	Note : main function is given for your reference which we are using internally to test
	the code.
	*/
	import java.util.ArrayList;
	public class PQ {
		private ArrayList<Integer> heap;
		public PQ() {
			heap = new ArrayList<Integer>();
		}
		boolean isEmpty() {
			return heap.size() == 0;
		}
		int size() {
			return heap.size();
		}
		int getMin() throws PriorityQueueException {
			if (isEmpty()) {
				// Throw an exception
				throw new PriorityQueueException();
			}
			return heap.get(0);
		}
		void insert(int element) {
			heap.add(element);
			int childIndex = heap.size() - 1;
			int parentIndex = (childIndex - 1) / 2;
			while (childIndex > 0) {
				if (heap.get(childIndex) < heap.get(parentIndex)) {
					int temp = heap.get(childIndex);
					heap.set(childIndex, heap.get(parentIndex));
					heap.set(parentIndex, temp);
					childIndex = parentIndex;
					parentIndex = (childIndex - 1) / 2;
				} else {
					return;
				}
			}
		}
		int removeMin() throws PriorityQueueException {
			// Complete this function
			// Throw the exception PriorityQueueException if queue is empty
			if (isEmpty()) {
				// Throw an exception
				throw new PriorityQueueException();
			}

		 int minEle=getMin();

		 //Set the last priority element as the new root
		 heap.set(0,heap.get(heap.size()-1));

		 //Remove the last priority element
		 heap.remove(heap.size()-1);

		 //Traverse from root to leaf, and swap values if needed to maintain min. heap
```

```
order property
        int parentIndex=0;
        int leftChildIndex=2*parentIndex+1,rightChildIndex=2*parentIndex+2;
        while(leftChildIndex < heap.size())
        {
            int minIndex=parentIndex;

            if(heap.get(minIndex) > heap.get(leftChildIndex))
            {
                minIndex=leftChildIndex;
            }

            if(rightChildIndex < heap.size())
            {
                if(heap.get(minIndex) > heap.get(rightChildIndex))
               {
                       minIndex=rightChildIndex;
               }
            }

            if(minIndex == parentIndex)
            {
                return minEle;
            }

            int temp=heap.get(parentIndex);
            heap.set(parentIndex,heap.get(minIndex));
            heap.set(minIndex,temp);

            parentIndex=minIndex;
            leftChildIndex=2*parentIndex+1;
            rightChildIndex=2*parentIndex+2;
        }
        return minEle;
    }
}
class PriorityQueueException extends Exception {
}
```

Code : Max Priority Queue

Send Feedback

Implement the class for Max Priority Queue which includes following functions -

1. getSize -

Return the size of priority queue i.e. number of elements present in the priority queue.

2. isEmpty -

Check if priority queue is empty or not. Return true or false accordingly.

3. insert -

Given an element, insert that element in the priority queue at the correct position.

4. getMax -

Return the maximum element present in the priority queue without deleting. Return -Infinity if priority queue is empty.

5. removeMax -

Delete and return the maximum element present in the priority queue. Return -Infinity if priority queue is empty.

Note : main function is given for your reference which we are using internally to test the class.

```
import java.util.*;

public class PQ {

        // Complete this class

   private ArrayList<Integer> heap = new ArrayList<Integer>();
```

```
    boolean isEmpty() {
            // Implement the isEmpty() function here
return heap.isEmpty();
    }

    int getSize() {
            // Implement the getSize() function here
return heap.size();
    }

    int getMax() {
            // Implement the getMax() function here
if(heap.isEmpty()){
   return Integer.MIN_VALUE;
}
return heap.get(0);
    }

    void insert(int element) {
            // Implement the insert() function here
heap.add(element);
int childIndex = heap.size()-1;
int parentIndex = (childIndex-1)/2;

while(parentIndex>=0){
   if(heap.get(childIndex)>heap.get(parentIndex)){
      int temp = heap.get(childIndex);
      heap.set(childIndex,heap.get(parentIndex));
      heap.set(parentIndex,temp);
      childIndex = parentIndex;
      parentIndex = (childIndex-1)/2;
   }
```

```
   else{
      return ;
   }
}
     }

     int removeMax() {
                // Implement the removeMax() function here
if(heap.isEmpty())
   return Integer.MIN_VALUE;

int maxEle = heap.get(0);
heap.set(0,heap.get(heap.size()-1));
heap.remove(heap.size()-1);

int parentIndex = 0;
int leftChildIndex = 2*parentIndex+1;
int rightChildIndex = 2*parentIndex+2;

while(leftChildIndex<heap.size()){
  int maxIndex=parentIndex;
   if(heap.get(maxIndex)<heap.get(leftChildIndex)){
      maxIndex = leftChildIndex;
   }

   if(rightChildIndex<heap.size()){
      if(heap.get(maxIndex)<heap.get(rightChildIndex)){
         maxIndex=rightChildIndex;
      }
   }
   if(maxIndex==parentIndex){
      return maxEle;
```

```
        }


        int temp = heap.get(maxIndex);
        heap.set(maxIndex,heap.get(parentIndex));
        heap.set(parentIndex,temp);


        parentIndex=maxIndex;
        leftChildIndex=2*parentIndex+1;
        rightChildIndex=2*parentIndex+2;
      }
      return maxEle;
          }
}
```

PRIORITY QUEUES -2

Inplace Heap Sort

Send Feedback

Given an integer array of size N. Sort this array (in decreasing order) using heap sort.
Note: Space complexity should be O(1).

Input Format:

The first line of input contains an integer, that denotes the value of the size of the array or N.
The following line contains N space separated integers, that denote the value of the elements of the array.

Output Format :

The first and only line of output contains array elements after sorting. The elements of the array in the output are separated by single space.

Constraints :

1 <= n <= 10^6
Time Limit: 1 sec

Sample Input 1:

```
6
2 6 8 5 4 3
```

Sample Output 1:

```
8 6 5 4 3 2
```

```
import java.util.*;

public class Solution {


        public static void inplaceHeapSort(int arr[]) {
                /* Your class should be named Solution
                * Don't write main().
                * Don't read input, it is passed as function argument.
```

```
                * Change in the given input itself.
                * Taking input and printing output is handled automatically.
                */
                int n = arr.length;
for(int i=(n/2)-1;i>=0;i--){
   downHeapify(arr,i,n);
}

for(int i=n-1;i>=0;i--){
   int temp=arr[i];
   arr[i]=arr[0];
   arr[0] = temp;
   downHeapify(arr,0,i);

}
     }
private static void downHeapify(int[] arr, int i, int n){
   int parentIndex = i;
   int leftChildIndex = 2*parentIndex+1;
   int rightChildIndex = 2*parentIndex+2;

   while(leftChildIndex<n){
      int minIndex = parentIndex;
      if(arr[minIndex]>arr[leftChildIndex])
         minIndex=leftChildIndex;

      if(rightChildIndex<n && arr[minIndex]>arr[rightChildIndex])
         minIndex=rightChildIndex;

      if(minIndex==parentIndex)
         return;
```

```
            int temp = arr[parentIndex];
            arr[parentIndex]= arr[minIndex];
            arr[minIndex]= temp;

            parentIndex = minIndex;
            leftChildIndex = 2*parentIndex+1;
            rightChildIndex = 2*parentIndex+2;
        }
    }

}
```

K Largest Elements

Send Feedback

You are given with an integer k and an array of integers that contain numbers in random order. Write a program to find k largest numbers from given array. You need to save them in an array and return it.
Time complexity should be O(nlogk) and space complexity should be not more than O(k).

Order of elements in the output is not important.

Input Format :

```
Line 1 : Size of array (n)
Line 2 : Array elements (separated by space)
Line 3 : Integer k
```

Output Format :

```
k largest elements
```

Sample Input :

```
13
2 12 9 16 10 5 3 20 25 11 1 8 6
4
```

Sample Output :

```
12
16
20
25
```

```
import java.util.PriorityQueue;

import java.util.ArrayList;


public class Solution {


    public static ArrayList<Integer> kLargest(int input[], int k) {
        /* Your class should be named Solution
        * Don't write main().
        * Don't read input, it is passed as function argument.
```

```
                * Return output and don't print it.
                * Taking input and printing output is handled automatically.
                */
                PriorityQueue<Integer>pq = new PriorityQueue<Integer>();

    for(int i=0;i<k;i++){
       pq.add(input[i]);
    }

    for(int i=k;i<input.length;i++){
       int minVal = pq.peek();
       if(minVal<input[i])
          minVal=input[i];
       if(minVal!=pq.peek()){
          pq.poll();
          pq.add(minVal);
       }
    }
    ArrayList<Integer> arr = new ArrayList<Integer>();
    while(!pq.isEmpty())
       arr.add(pq.poll());
         return arr;
         }
}
```

K Smallest Elements

Send Feedback

You are given with an integer k and an array of integers that contain numbers in random order. Write a program to find k smallest numbers from given array. You need to save them in an array and return it.
Time complexity should be O(n * logk) and space complexity should not be more than O(k).
Note: Order of elements in the output is not important.

Input Format :

The first line of input contains an integer, that denotes the value of the size of the array. Let us denote it with the symbol N.
The following line contains N space separated integers, that denote the value of the elements of the array.
The following line contains an integer, that denotes the value of k.

Output Format :

The first and only line of output print k smallest elements. The elements in the output are separated by a single space.

Constraints:

Time Limit: 1 sec

Sample Input 1 :

```
13
2 12 9 16 10 5 3 20 25 11 1 8 6
4
```

Sample Output 1 :

```
1 2 3 5
```

```
import java.util.*;

import java.util.ArrayList;



public class Solution {



        public static ArrayList<Integer> kSmallest(int n, int[] input, int k) {

                // Write your code here

        PriorityQueue<Integer>pq = new PriorityQueue<Integer>(Collections.reverseOrder());



    for(int i=0;i<k;i++){

       pq.add(input[i]);

    }



    for(int i=k;i<input.length;i++){

       int maxVal = pq.peek();

       if(maxVal>input[i])

          maxVal=input[i];

       if(maxVal!=pq.peek()){

          pq.poll();

          pq.add(maxVal);

       }

    }

    ArrayList<Integer> arr = new ArrayList<Integer>();

    while(!pq.isEmpty())

       arr.add(pq.poll());

         return arr;

         }
```

}

Check Max-Heap

Send Feedback

Given an array of integers, check whether it represents max-heap or not. Return true if the given array represents max-heap, else return false.

Input Format:

The first line of input contains an integer, that denotes the value of the size of the array. Let us denote it with the symbol N.
The following line contains N space separated integers, that denote the value of the elements of the array.

Output Format :

The first and only line of output contains true if it represents max-heap and false if it is not a max-heap.

Constraints:

1 <= N <= 10^5
1 <= Ai <= 10^5
Time Limit: 1 sec

Sample Input 1:

8
42 20 18 6 14 11 9 4

Sample Output 1:

true

```
import java.util.*;

public class Solution {


	public static boolean checkMaxHeap(int arr[]) {

		/*

		 * Your class should be named Solution Don't write main(). Don't read input, it

		 * is passed as function argument. Return output and don't print it. Taking

		 * input and printing output is handled automatically.

		 */

    return checkmaxHeapHelper(arr,0);

	}


  private static boolean checkmaxHeapHelper(int[] arr,int startIndex){

    if(startIndex==arr.length){

      return true;

    }


    int parentIndex = startIndex;
```

```
        int leftChildIndex=2*parentIndex+1;
        int rightChildIndex=2*parentIndex+2;

        boolean leftAns = true;
        boolean rightAns = true;

        if(leftChildIndex<arr.length){
           if(arr[parentIndex]<arr[leftChildIndex])
              return false;
           else
              leftAns = checkmaxHeapHelper(arr,leftChildIndex);


           }
        if(rightChildIndex<arr.length){
           if(arr[parentIndex]<arr[rightChildIndex])
              return false;
           else
              rightAns = checkmaxHeapHelper(arr,rightChildIndex);
        }
        return (leftAns && rightAns);

    }
}
```

Kth largest element

Send Feedback

Given an array A of random integers and an integer k, find and return the kth largest element in the array.
Note: Try to do this question in less than O(N * logN) time.

Input Format :

The first line of input contains an integer, that denotes the value of the size of the array. Let us denote it with the symbol N.
The following line contains N space separated integers, that denote the value of the elements of the array.
The following contains an integer, that denotes the value of k.

Output Format :

The first and only line of output contains the kth largest element

Constraints :

1 <= N, Ai, k <= 10^5
Time Limit: 1 sec

Sample Input 1 :

```
6
9 4 8 7 11 3
2
```

Sample Output 1 :

```
9
```

Sample Input 2 :

```
8
2 6 10 11 13 4 1 20
4
```

Sample Output 2 :

```
10
```

```
import java.util.*;

public class Solution {


        public static int kthLargest(int n, int[] input, int k) {

                // Write your code here

    PriorityQueue<Integer> pq = new PriorityQueue<Integer>();


    for(int i=0;i<k;i++){

       pq.add(input[i]);

    }


    for(int i=k;i<input.length;i++){

       int minVal = pq.peek();

       if(minVal<input[i])

          minVal=input[i];

       if(minVal!=pq.peek()){

          pq.poll();

          pq.add(minVal);

       }

            }

    int minVal = Integer.MAX_VALUE;

    while(!pq.isEmpty()){

       int check = pq.poll();

       if(check<minVal)

          minVal=check;
```

```
    }

    return minVal;

        }

}
```

Buy the ticket

Send Feedback

You want to buy a ticket for a well-known concert which is happening in your city. But the number of tickets available is limited. Hence the sponsors of the concert decided to sell tickets to customers based on some priority.
A queue is maintained for buying the tickets and every person is attached with a priority (an integer, 1 being the lowest priority).
The tickets are sold in the following manner -
1. The first person (pi) in the queue requests for the ticket.
2. If there is another person present in the queue who has higher priority than pi, then ask pi to move at end of the queue without giving him the ticket.
3. Otherwise, give him the ticket (and don't make him stand in queue again).
Giving a ticket to a person takes exactly 1 minute and it takes no time for removing and adding a person to the queue. And you can assume that no new person joins the queue.
Given a list of priorities of N persons standing in the queue and the index of your priority (indexing starts from 0). Find and return the time it will take until you get the ticket.

Input Format:

The first line of input contains an integer, that denotes the value of total number of people standing in queue or the size of the array of priorities. Let us denote it with the symbol N.
The following line contains N space separated integers, that denote the value of the elements of the array of priorities.
The following contains an integer, that denotes the value of index of your priority. Let us denote it with symbol k.

Output Format :

The first and only line of output contains the time required for you to get the ticket.

Constraints:

Time Limit: 1 sec

Sample Input 1 :

```
3
3 9 4
2
```

Sample Output 1 :

```
2
```

Sample Output 1 Explanation :

Person with priority 3 comes out. But there is a person with higher priority than him. So he goes and then stands in the queue at the end. Queue's status : {9, 4, 3}. Time : 0 secs.
Next, the person with priority 9 comes out. And there is no person with higher priority than him. So he'll get the ticket. Queue's status : {4, 3}. Time : 1 secs.
Next, the person with priority 4 comes out (which is you). And there is no person with higher priority than you. So you'll get the ticket. Time : 2 secs.

Sample Input 2 :

```
5
2 3 2 2 4
3
```

Sample Output 2 :

```
4
```

```
import java.util.*;

public class Solution {


        public static int buyTicket(int input[], int k) {
```

```
                /* Your class should be named Solution
                * Don't write main().
                * Don't read input, it is passed as function argument.
                * Return output and don't print it.
                * Taking input and printing output is handled automatically.
                */
    Queue<Integer> queue = new LinkedList<>();
    PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());

    for(int i=0;i<input.length;i++){
       queue.add(i);
       pq.add(input[i]);
    }
    int time = 0;
    while(!queue.isEmpty()){
       if(input[queue.peek()]<pq.peek()){
          queue.add(queue.poll());
       }
       else{
          int index = queue.poll();
          pq.remove();
          time++;

          if(index==k){
             break;
          }
       }
    }
    return time;

      }
}
```