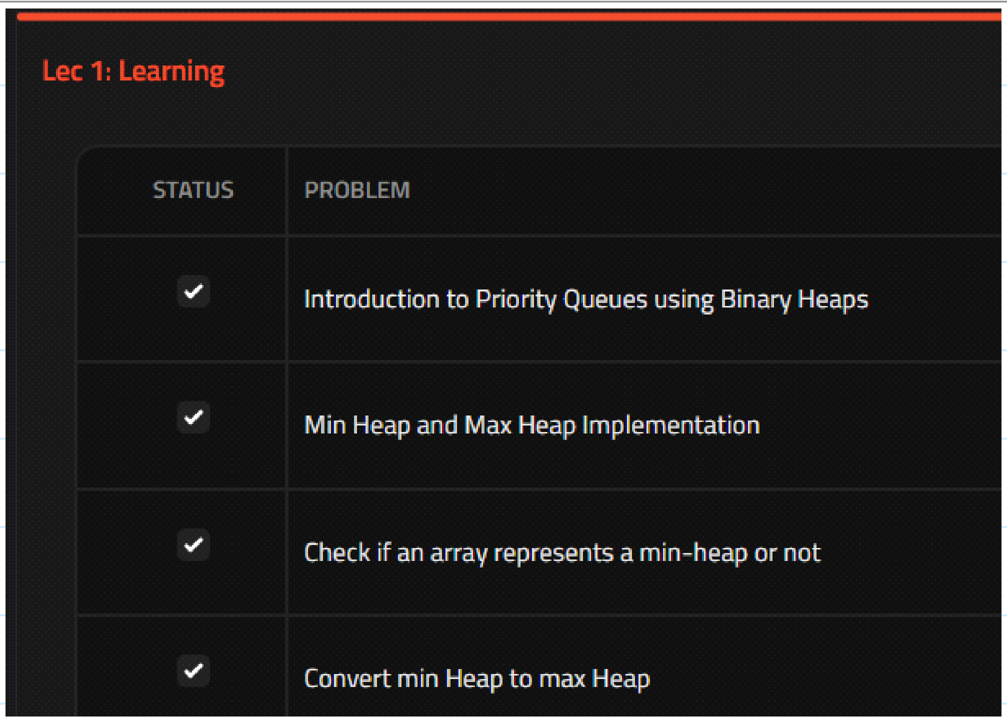
16 March 2025 01:48

Learning part 1

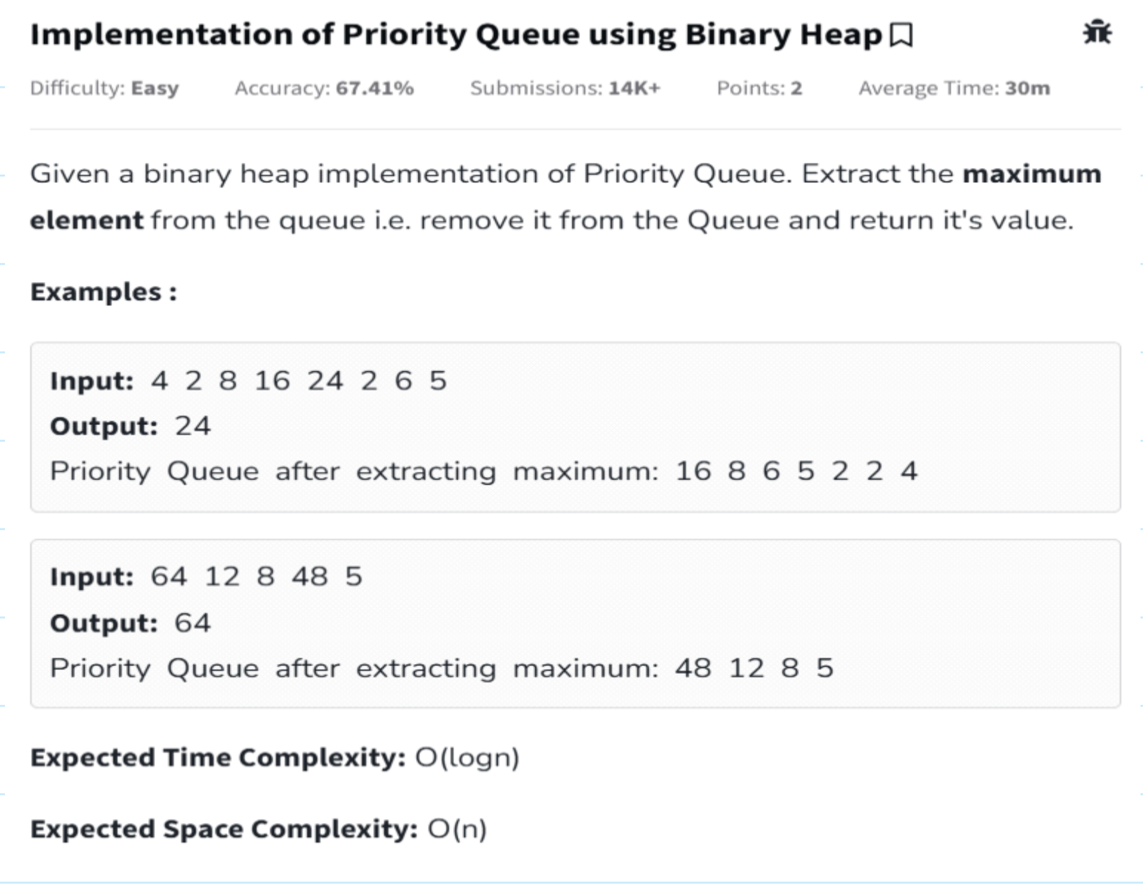


Lec 1: Learning

STATUS	PROBLEM
✓	Introduction to Priority Queues using Binary Heaps
✓	Min Heap and Max Heap Implementation
✓	Check if an array represents a min-heap or not
✓	Convert min Heap to max Heap

===

Quest1



Implementation of Priority Queue using Binary Heap

Difficulty: **Easy** Accuracy: **67.41%** Submissions: **14K+** Points: **2** Average Time: **30m**

Given a binary heap implementation of Priority Queue. Extract the **maximum element** from the queue i.e. remove it from the Queue and return it's value.

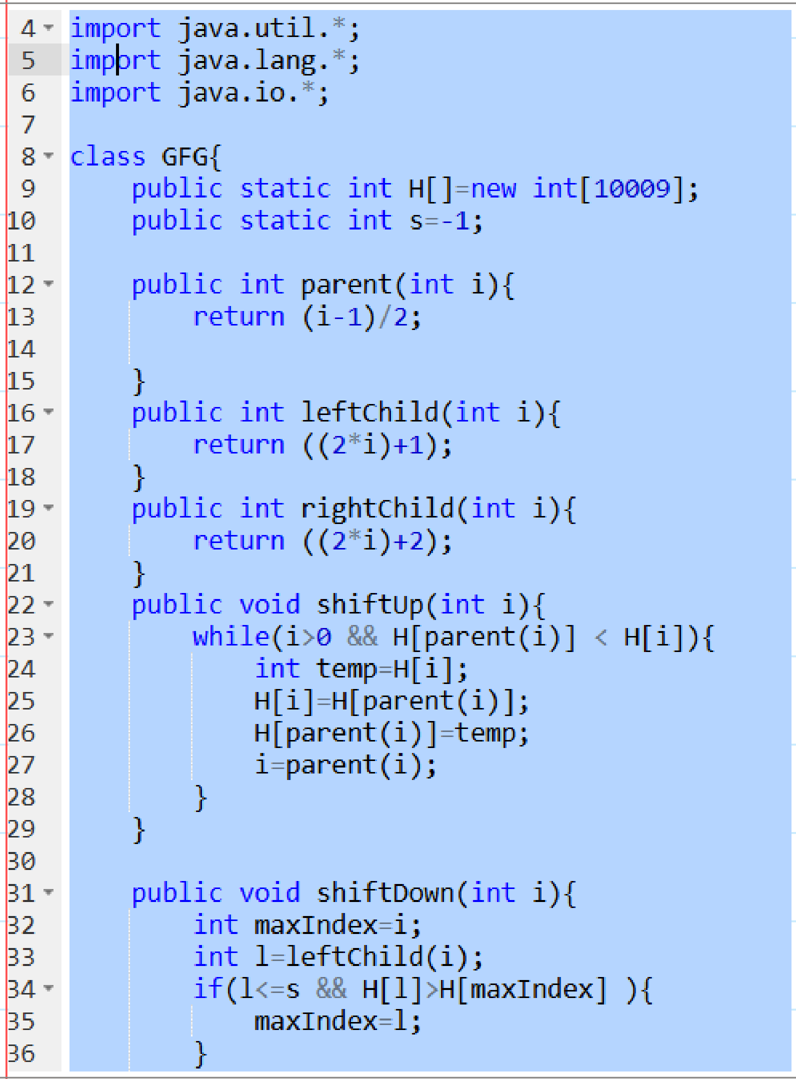
Examples :

```
Input: 4 2 8 16 24 2 6 5
Output: 24
Priority Queue after extracting maximum: 16 8 6 5 2 2 4
```

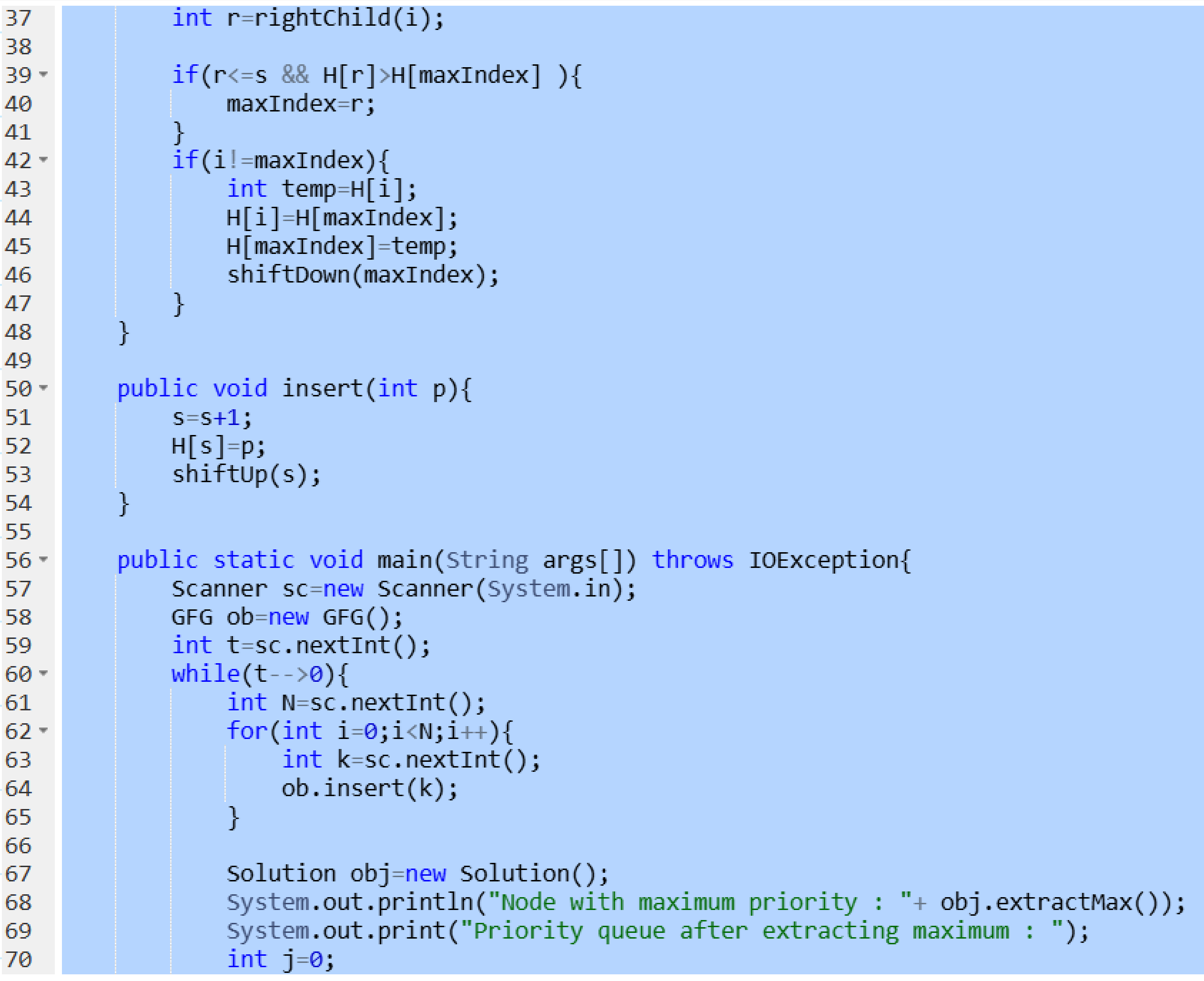
```
Input: 64 12 8 48 5
Output: 64
Priority Queue after extracting maximum: 48 12 8 5
```

Expected Time Complexity: O(logn)

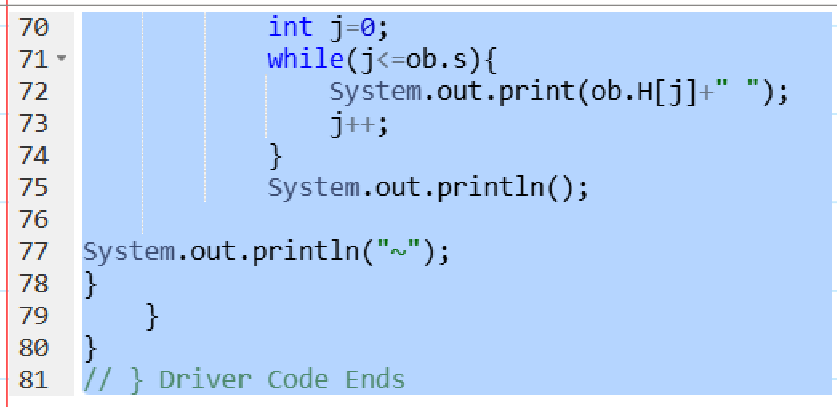
Expected Space Complexity: O(n)



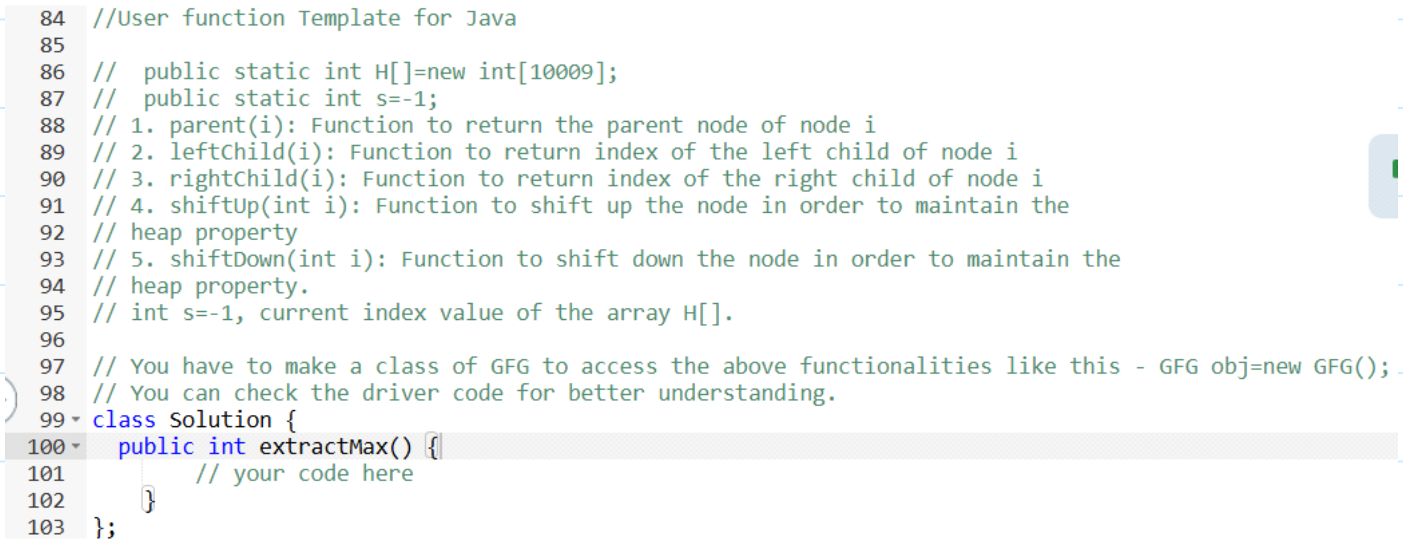
```java
import java.util.*;
import java.lang.*;
import java.io.*;

class GFG{
    public static int H[]=new int[10009];
    public static int s=-1;

    public int parent(int i){
        return (i-1)/2;

    }
    public int leftChild(int i){
        return ((2*i)+1);
    }
    public int rightChild(int i){
        return ((2*i)+2);
    }
    public void shiftUp(int i){
        while(i>0 && H[parent(i)] < H[i]){
            int temp=H[i];
            H[i]=H[parent(i)];
            H[parent(i)]=temp;
            i=parent(i);
        }
    }

    public void shiftDown(int i){
        int maxIndex=i;
        int l=leftChild(i);
        if(l<=s && H[l]>H[maxIndex] ){
            maxIndex=l;
        }
```



```java
        int r=rightChild(i);

        if(r<=s && H[r]>H[maxIndex] ){
            maxIndex=r;
        }
        if(i!=maxIndex){
            int temp=H[i];
            H[i]=H[maxIndex];
            H[maxIndex]=temp;
            shiftDown(maxIndex);
        }
    }

    public void insert(int p){
        s=s+1;
        H[s]=p;
        shiftUp(s);
    }

    public static void main(String args[]) throws IOException{
        Scanner sc=new Scanner(System.in);
        GFG ob=new GFG();
        int t=sc.nextInt();
        while(t-->0){
            int N=sc.nextInt();
            for(int i=0;i<N;i++){
                int k=sc.nextInt();
                ob.insert(k);
            }

            Solution obj=new Solution();
            System.out.println("Node with maximum priority : "+ obj.extractMax());
            System.out.print("Priority queue after extracting maximum : ");
            int j=0;
```



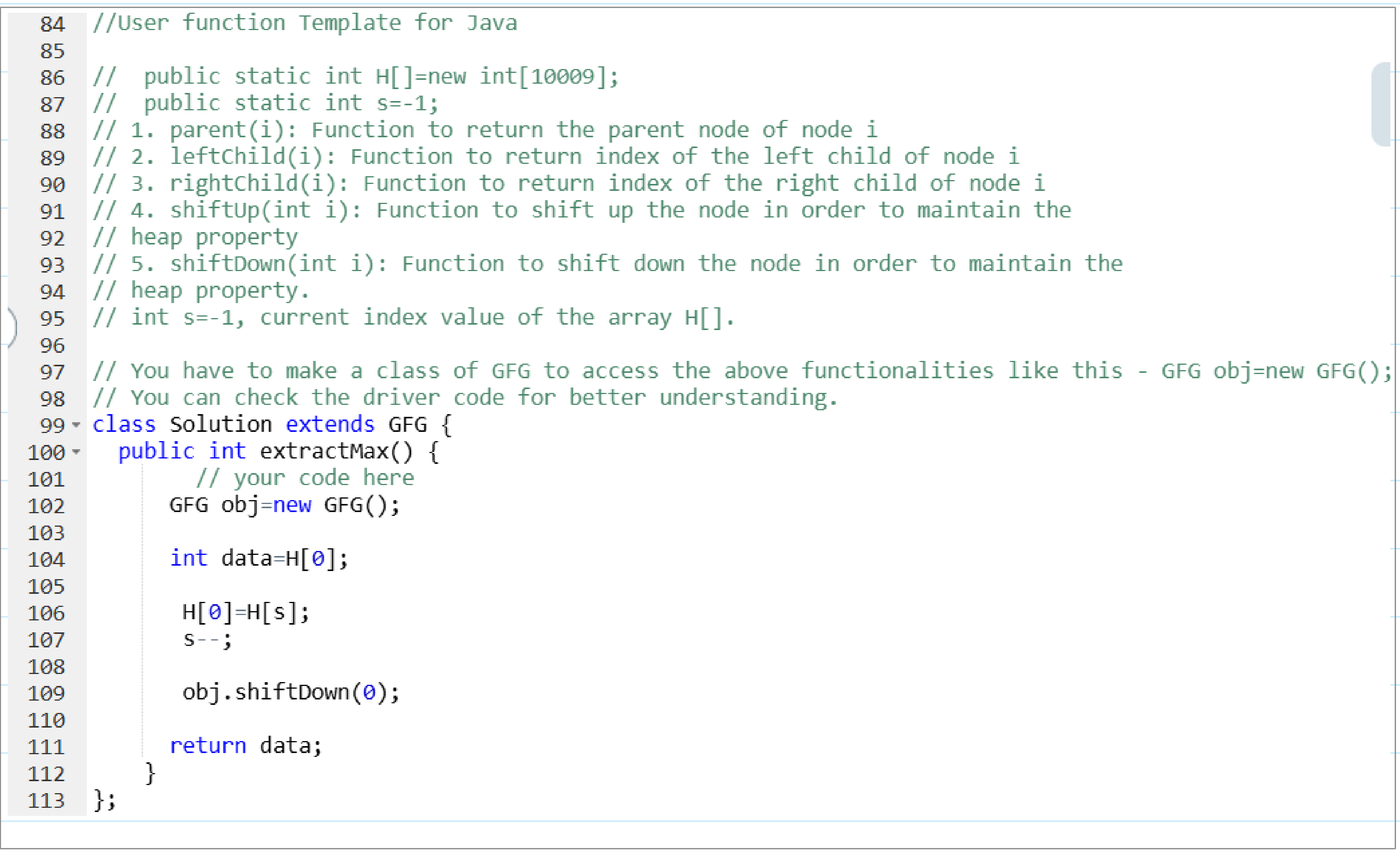
```java
            int j=0;
            while(j<=ob.s){
                System.out.print(ob.H[j]+" ");
                j++;
            }
            System.out.println();

System.out.println("~");
}
    }
}
// } Driver Code Ends
```



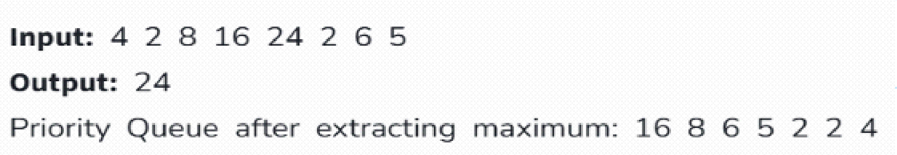
```java
//User function Template for Java

//  public static int H[]=new int[10009];
//  public static int s=-1;
// 1. parent(i): Function to return the parent node of node i
// 2. leftChild(i): Function to return index of the left child of node i
// 3. rightChild(i): Function to return index of the right child of node i
// 4. shiftUp(int i): Function to shift up the node in order to maintain the
// heap property
// 5. shiftDown(int i): Function to shift down the node in order to maintain the
// heap property.
// int s=-1, current index value of the array H[].

// You have to make a class of GFG to access the above functionalities like this - GFG obj=new GFG();
// You can check the driver code for better understanding.
class Solution {
  public int extractMax() {
        // your code here
    }
};
```

16 March 2025 18:44

Sol



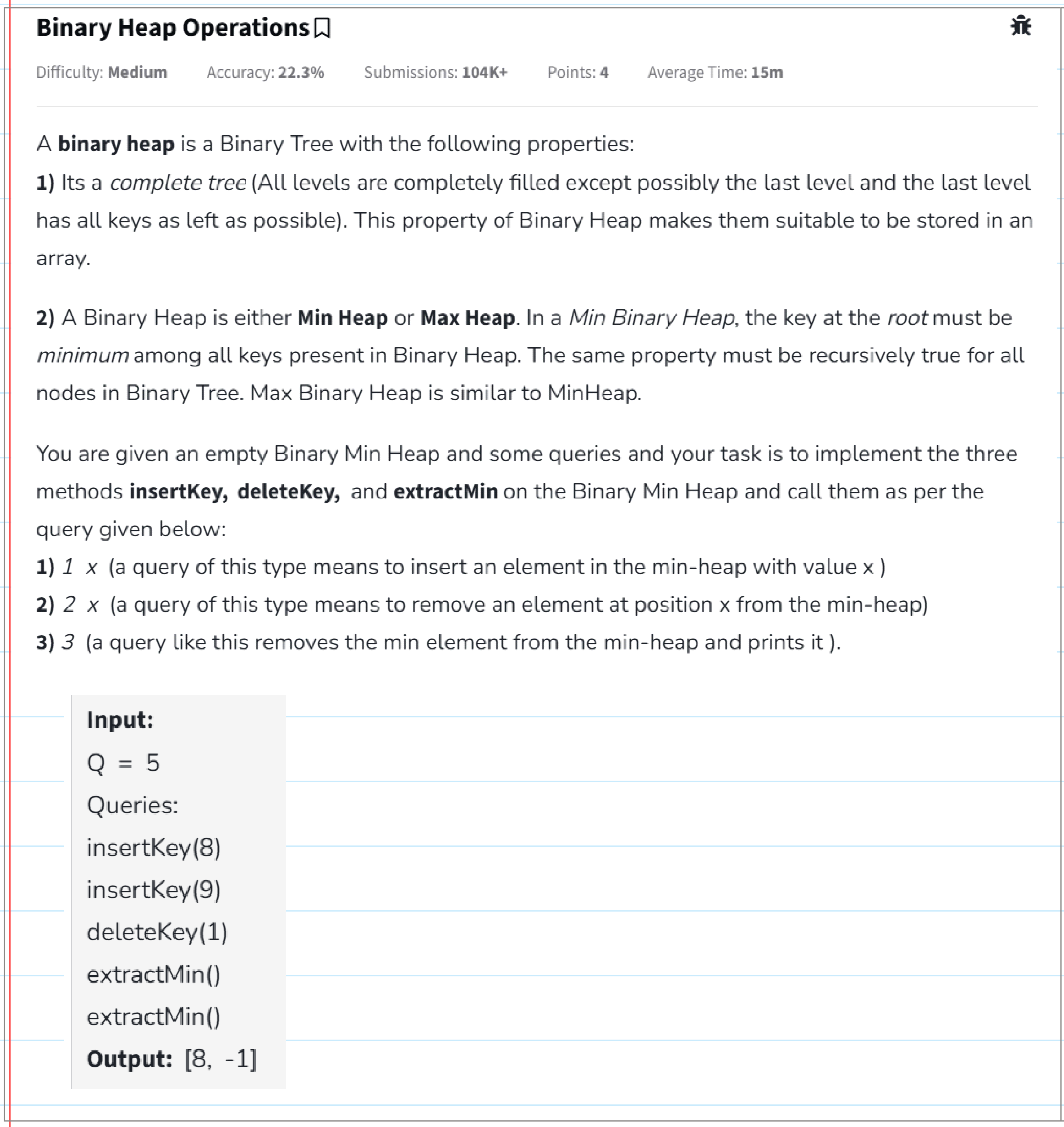
```
//User function Template for Java

//  public static int H[]=new int[10009];
//  public static int s=-1;
// 1. parent(i): Function to return the parent node of node i
// 2. leftChild(i): Function to return index of the left child of node i
// 3. rightChild(i): Function to return index of the right child of node i
// 4. shiftUp(int i): Function to shift up the node in order to maintain the
// heap property
// 5. shiftDown(int i): Function to shift down the node in order to maintain the
// heap property.
// int s=-1, current index value of the array H[].

// You have to make a class of GFG to access the above functionalities like this - GFG obj=new GFG();
// You can check the driver code for better understanding.
class Solution extends GFG {
  public int extractMax() {
        // your code here
      GFG obj=new GFG();

      int data=H[0];

       H[0]=H[s];
       s--;

       obj.shiftDown(0);

      return data;
    }
};
```



Input: 4 2 8 16 24 2 6 5
Output: 24
Priority Queue after extracting maximum: 16 8 6 5 2 2 4

Que 2

16 March 2025 18:50



Binary Heap Operations

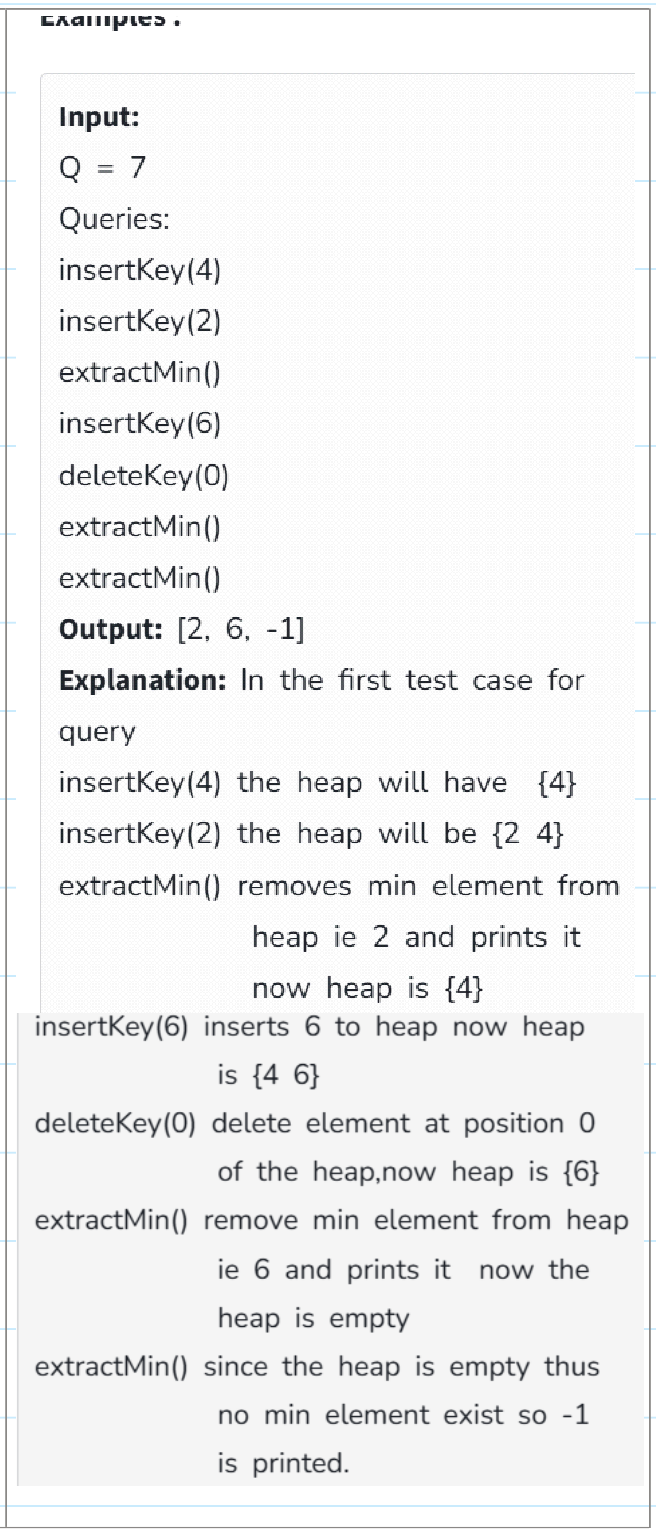
Difficulty: **Medium** Accuracy: **22.3%** Submissions: **104K+** Points: **4** Average Time: **15m**

A **binary heap** is a Binary Tree with the following properties:
1) Its a *complete tree* (All levels are completely filled except possibly the last level and the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.

2) A Binary Heap is either **Min Heap** or **Max Heap**. In a *Min Binary Heap*, the key at the *root* must be *minimum* among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to MinHeap.

You are given an empty Binary Min Heap and some queries and your task is to implement the three methods **insertKey, deleteKey,** and **extractMin** on the Binary Min Heap and call them as per the query given below:
1) *1 x* (a query of this type means to insert an element in the min-heap with value x)
2) *2 x* (a query of this type means to remove an element at position x from the min-heap)
3) *3* (a query like this removes the min element from the min-heap and prints it).

Input:
Q = 5
Queries:
insertKey(8)
insertKey(9)
deleteKey(1)
extractMin()
extractMin()
Output: [8, -1]



Examples :

Input:
Q = 7
Queries:
insertKey(4)
insertKey(2)
extractMin()
insertKey(6)
deleteKey(0)
extractMin()
extractMin()
Output: [2, 6, -1]
Explanation: In the first test case for query
insertKey(4) the heap will have {4}
insertKey(2) the heap will be {2 4}
extractMin() removes min element from heap ie 2 and prints it now heap is {4}
insertKey(6) inserts 6 to heap now heap is {4 6}
deleteKey(0) delete element at position 0 of the heap,now heap is {6}
extractMin() remove min element from heap ie 6 and prints it now the heap is empty
extractMin() since the heap is empty thus no min element exist so -1 is printed.

Syntax	

Syntax

```
class MinHeap {
    int[] harr;
    int capacity;
    int heap_size;
    MinHeap(int cap) {
        heap_size = 0;
        capacity = cap;
        harr = new int[cap];
    }
    int parent(int i) { return (i - 1) / 2; }
    int left(int i) { return (2 * i + 1); }
    int right(int i) { return (2 * i + 2); }
    //Function to extract minimum value in heap and then to store
    //next minimum value at first index.
    int extractMin()
    {
        // Your code here.
    }
    //Function to insert a value in Heap.
    void insertKey(int k)
    {
        // Your code here.
    }
    //Function to delete a key at ith index.
    void deleteKey(int i)
    {
        // Your code here.
    }
    //Function to change value at ith index and store that value at first index.
    void decreaseKey(int i, int new_val)
    {
        harr[i] = new_val;
        while (i != 0 && harr[parent(i)] > harr[i]) {
            int temp = harr[i];
            harr[i] = harr[parent(i)];
            harr[parent(i)] = temp;
            i = parent(i);
        }
    }
    /* You may call below MinHeapify function in
      above codes. Please do not delete this code
      if you are not writing your own MinHeapify */
    void MinHeapify(int i) {
        int l = left(i);
        int r = right(i);
        int smallest = i;
        if (l < heap_size && harr[l] < harr[i]) smallest = l;
        if (r < heap_size && harr[r] < harr[smallest]) smallest = r;
        if (smallest != i) {
            int temp = harr[i];
            harr[i] = harr[smallest];
            harr[smallest] = temp;
            MinHeapify(smallest);
        }
    }
}
```

16 March 2025 18:55

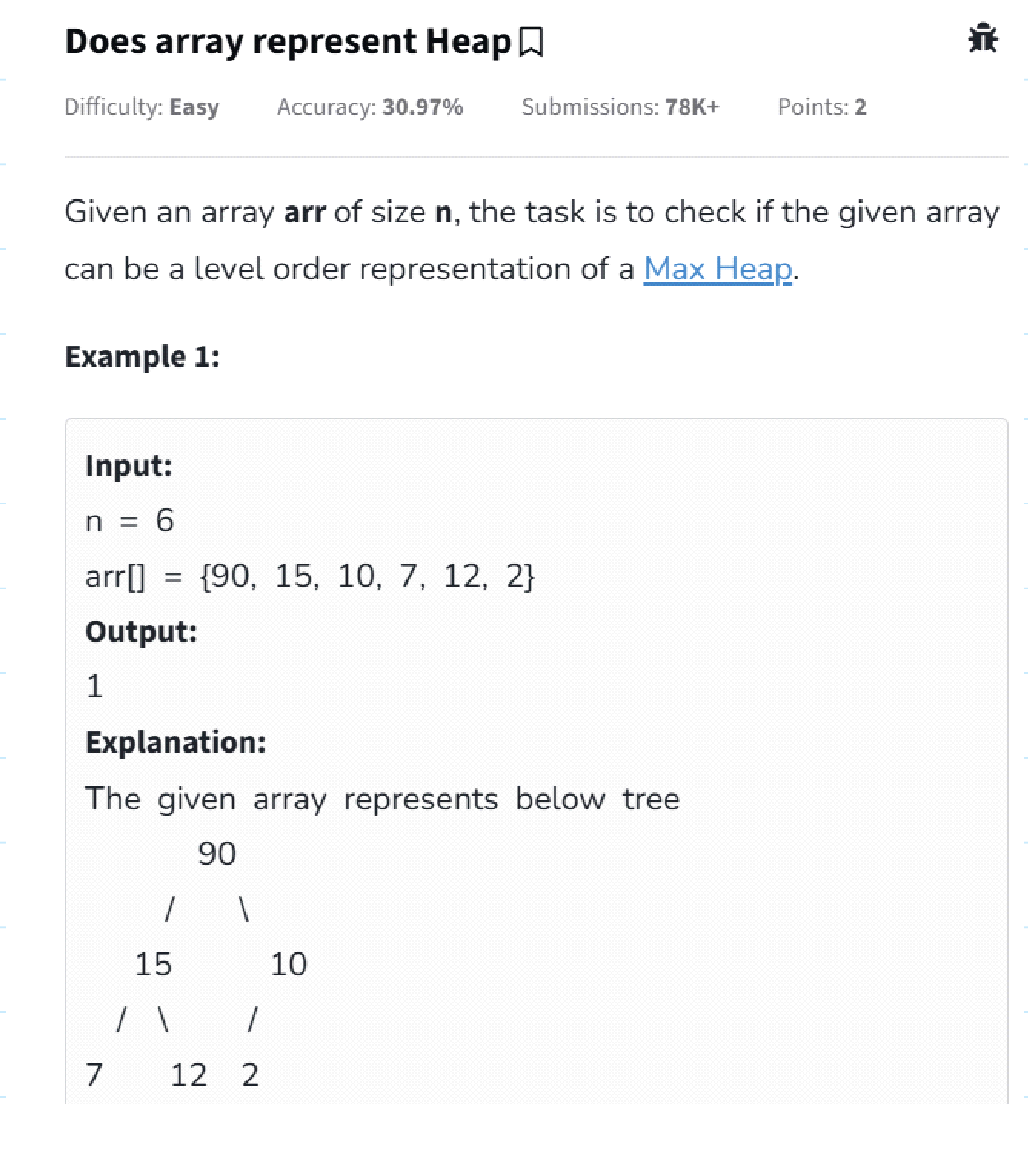
Code

```
class MinHeap {
    int[] harr;
    int capacity;
    int heap_size;
    MinHeap(int cap) {
        heap_size = 0;
        capacity = cap;
        harr = new int[cap];
    }
    int parent(int i) { return (i - 1) / 2; }
    int left(int i) { return (2 * i + 1); }
    int right(int i) { return (2 * i + 2); }
    //Function to extract minimum value in heap and then to store
    //next minimum value at first index.
    int extractMin()
    {
        // Your code here.
        if(heap_size==0) return -1;
        int data=harr[0];
        harr[0]=harr[heap_size-1];

        heap_size--;

        MinHeapify(0);
        return data;
    }
    //Function to insert a value in Heap.
    void insertKey(int k)
    {
        if(heap_size==capacity)
        return ;

        harr[heap_size]=k;
        heap_size++;

        int idx=heap_size-1;

        while(idx!=0 && harr[parent(idx)]>harr[idx]){
            int temp=harr[parent(idx)];
            harr[parent(idx)]=harr[idx];
            harr[idx]=temp;

            idx=parent(idx);
        }
        // Your code here.
    }
    //Function to delete a key at ith index.
    void deleteKey(int i)
    {
        if(i>=heap_size || i<0)
        return ;

        // Your code here.
        harr[i]=harr[heap_size-1];
        heap_size--;

        if(i!=0 && harr[parent(i)]>harr[i]){
            while(i!=0 && harr[parent(i)]>harr[i]){
                int temp=harr[parent(i)];
                harr[parent(i)]=harr[i];
                harr[i]=temp;

                i=parent(i);
            }
        }else{
            MinHeapify(i);
        }
    }
    //Function to change value at ith index and store that value at first index.
    void decreaseKey(int i, int new_val)
    {
        harr[i] = new_val;
        while (i != 0 && harr[parent(i)] > harr[i]) {
            int temp = harr[i];
            harr[i] = harr[parent(i)];
            harr[parent(i)] = temp;
            i = parent(i);
        }
    }
    /* You may call below MinHeapify function in
      above codes. Please do not delete this code
      if you are not writing your own MinHeapify */
    void MinHeapify(int i) {
        int l = left(i);
        int r = right(i);
        int smallest = i;
        if (l < heap_size && harr[l] < harr[i]) smallest = l;
        if (r < heap_size && harr[r] < harr[smallest]) smallest = r;
        if (smallest != i) {
            int temp = harr[i];
            harr[i] = harr[smallest];
            harr[smallest] = temp;
            MinHeapify(smallest);
        }
    }
}
```

Que 3 learning

16 March 2025 18:59

Que3



Does array represent Heap

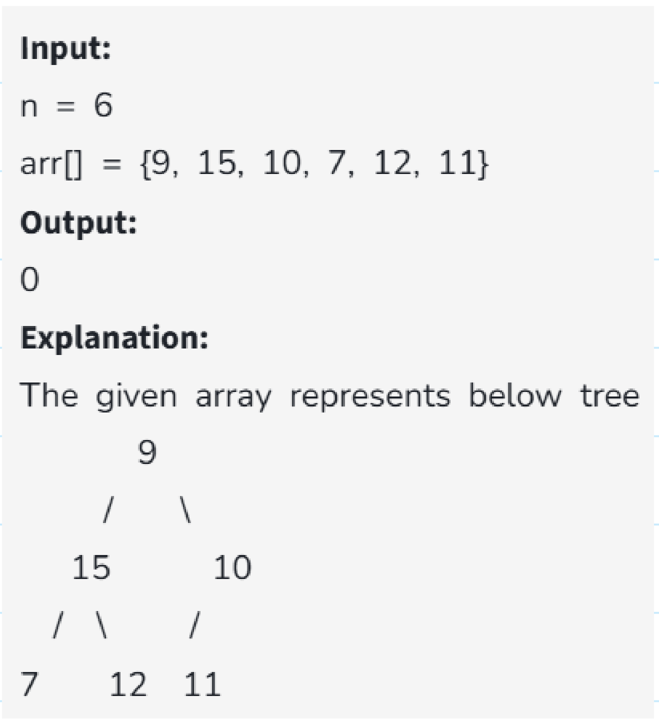
Difficulty: **Easy** Accuracy: **30.97%** Submissions: **78K+** Points: **2**

Given an array **arr** of size **n**, the task is to check if the given array can be a level order representation of a Max Heap.

Example 1:

Input:
n = 6
arr[] = {90, 15, 10, 7, 12, 2}
Output:
1
Explanation:
The given array represents below tree

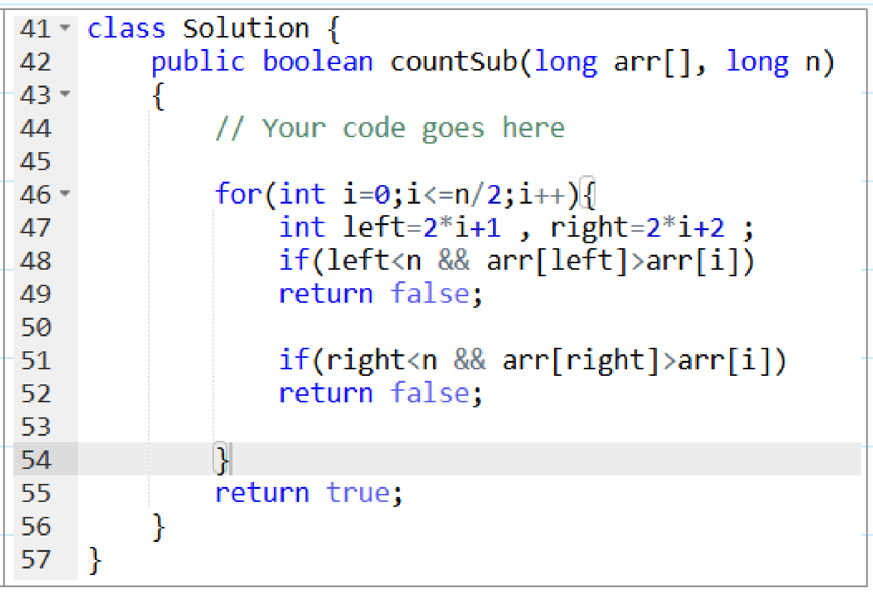
```
        90
      /    \
    15       10
   /  \     /
7     12   2
```



Input:
n = 6
arr[] = {9, 15, 10, 7, 12, 11}
Output:
0
Explanation:
The given array represents below tree

```
        9
      /    \
    15       10
   /  \     /
7     12   11
```

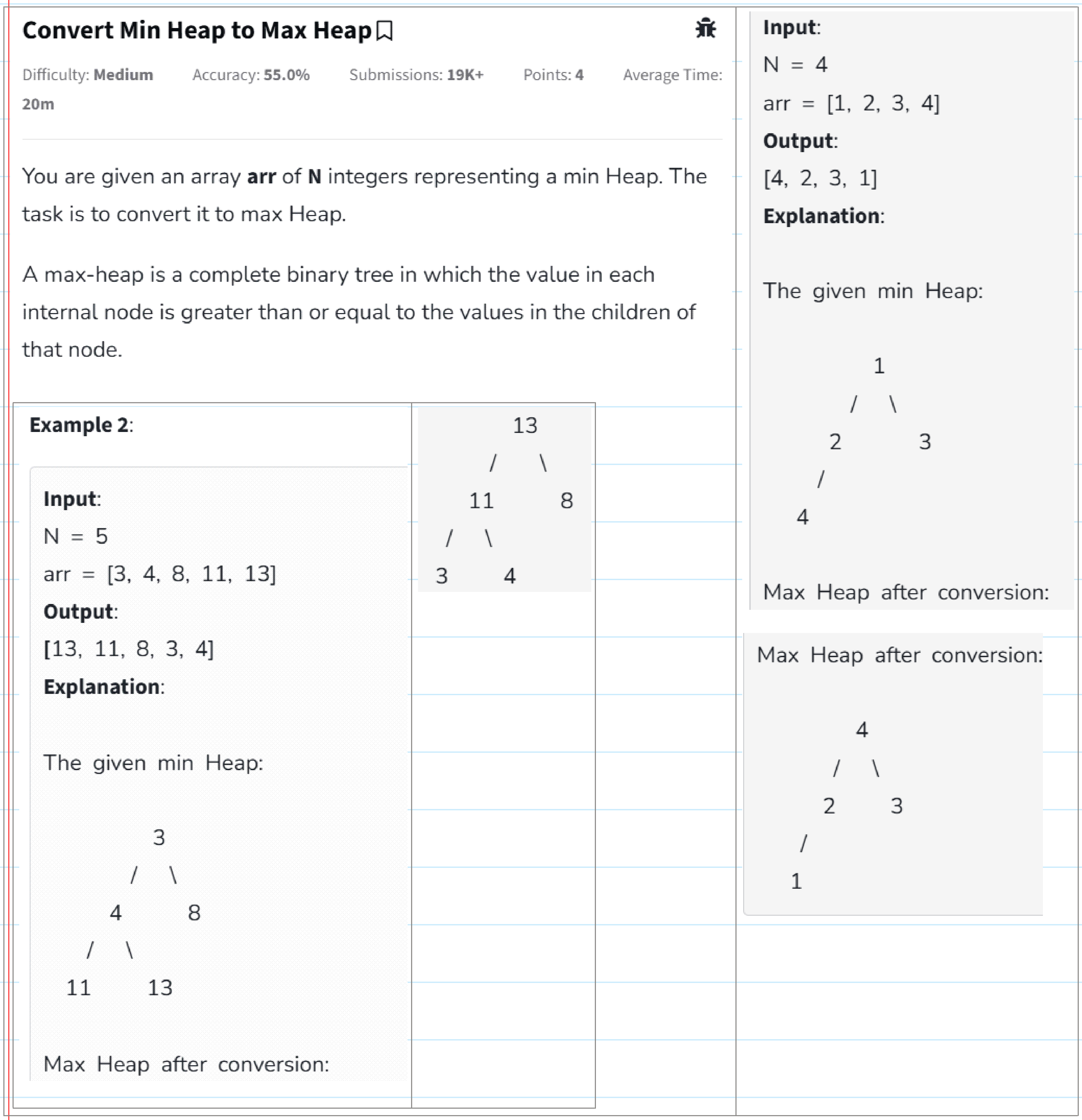
Solution



```
class Solution {
    public boolean countSub(long arr[], long n)
    {
        // Your code goes here

        for(int i=0;i<=n/2;i++){
            int left=2*i+1 , right=2*i+2 ;
            if(left<n && arr[left]>arr[i])
            return false;

            if(right<n && arr[right]>arr[i])
            return false;

        }
        return true;
    }
}
```

Que 4

16 March 2025 19:05



Convert Min Heap to Max Heap

Difficulty: **Medium** Accuracy: **55.0%** Submissions: **19K+** Points: **4** Average Time: **20m**

You are given an array **arr** of **N** integers representing a min Heap. The task is to convert it to max Heap.

A max-heap is a complete binary tree in which the value in each internal node is greater than or equal to the values in the children of that node.

Example 2:

Input:
N = 5
arr = [3, 4, 8, 11, 13]
Output:
[13, 11, 8, 3, 4]
Explanation:

The given min Heap:

```
          3
        /   \
      4       8
    /   \
  11     13
```

Max Heap after conversion:

```
         13
       /    \
     11       8
   /   \
 3       4
```

Input:
N = 4
arr = [1, 2, 3, 4]
Output:
[4, 2, 3, 1]
Explanation:

The given min Heap:

```
          1
        /   \
      2       3
    /
  4
```

Max Heap after conversion:

Max Heap after conversion:

```
          4
        /   \
      2       3
    /
  1
```

Solution

```java
class Solution {
    static void convertMinToMaxHeap(int N, int arr[]) {
    // code here

    for(int i=N/2;i>=0;i--)
    heapify(i,arr);
  }

  private static void heapify(int i, int[] arr){
      int maxi=i;
      int left=2*i+1;
      int right=2*i+2;

      if(left<arr.length && arr[left]>arr[maxi])
      maxi=left;

      if(right<arr.length && arr[right]>arr[maxi])
      maxi=right;

      if(maxi!=i){
          int temp=arr[maxi];
          arr[maxi]=arr[i];
          arr[i]=temp;

          heapify(maxi ,arr);
      }
  }
}
```