Kartikay

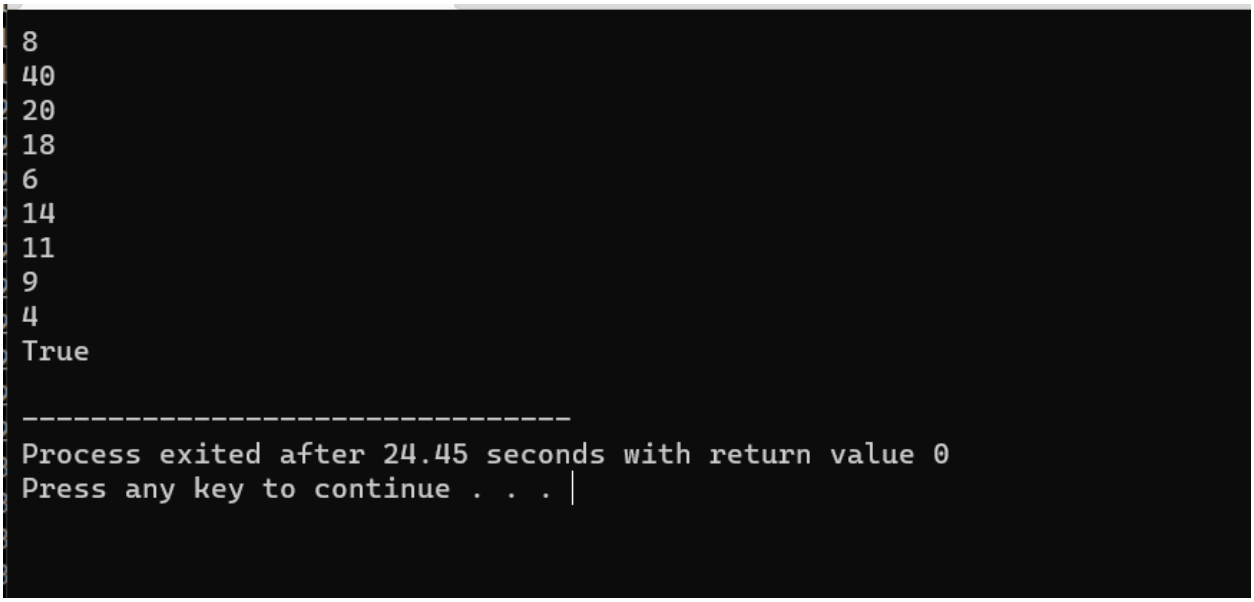
Priority Queue

```
Chapter Assignment
Problem Statement: Check Max-Heap
Problem Level: EASY
Problem Description:
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
```

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
  vector<int> heap;
  int n;
  cin>>n;
  for(int i=0;i<n;i++)
  {
    int val;
    cin>>val;
    heap.push_back(val);
  }

  int c=0;
  for(int i=0;i<n;i++)
  {
    int parent=i;
    int left=2*i+1;
    int right=2*i+2;
```

```
    if(left<n)
    {
      if(heap[parent]>heap[left])
      {
        if(right<n)
        {
          if(heap[parent]>heap[right])
          continue;
          else
          c++;
        }
      }
      else
      c++;
    }
    if(c!=0){

    cout<<"FALSE"<<endl;
    return 0;
  }
  }
  cout<<"True"<<endl;
}

Time Complexity:O(n)
Space Complexity:O(n)
```

OUTPUT—>



Kth smallest element

Medium Accuracy: **35.17%** Submissions: **414K+** Points: **4**

Unable to Crack Interviews of Your Dream Companies ? Click Here to End This Problem!

Given an array **arr[]** and an integer **K** where K is smaller than size of array, the task is to find the **K^{th} smallest** element in the given array. It is given that all array elements are distinct.

Example 1:

Input:
N = 6
arr[] = 7 10 4 3 20 15
K = 3
Output : 7
Explanation :
3rd smallest element in the given array is 7.

```
//{ Driver Code Starts
//Initial function template for C++

#include<bits/stdc++.h>
using namespace std;

// } Driver Code Ends
//User function template for C++

class Solution{
    public:
    // arr : given array
```

```
    // l : starting index of the array i.e 0
    // r : ending index of the array i.e size-1
    // k : find kth smallest element and return using this function
    int kthSmallest(int arr[], int l, int r, int k) {
        //code here
        priority_queue<int> pq;
        for(int i=0;i<k;i++)
        {
            pq.push(arr[i]);
        }
        for(int i=k;i<r+1;i++)
        {
            if(pq.top()>arr[i])
            {
                pq.pop();
                pq.push(arr[i]);
            }

        }
        return pq.top();

    }
};

//{ Driver Code Starts.

int main()
{
    int test_case;
    cin>>test_case;
    while(test_case--)
    {
        int number_of_elements;
        cin>>number_of_elements;
        int a[number_of_elements];

        for(int i=0;i<number_of_elements;i++)
            cin>>a[i];

        int k;
        cin>>k;
        Solution ob;
        cout<<ob.kthSmallest(a, 0, number_of_elements-1, k)<<endl;
    }
    return 0;
}
// } Driver Code Ends


Time Complexity:O(nlogk)
Space Complexity:O(k)
```

Nearly sorted

Medium Accuracy: **75.25%** Submissions: **25K+** Points: **4**

Unable to Crack Interviews of Your Dream Companies ? Click Here to End This Problem!

Given an array of **n** elements, where each element is at most **k** away from its target position, you need to sort the array optimally.

Example 1:

Input:
n = 7, k = 3
arr[] = {6,5,3,2,8,10,9}
Output: 2 3 5 6 8 9 10
Explanation: The sorted array will be
2 3 5 6 8 9 10

```
//{ Driver Code Starts
#include<bits/stdc++.h>
using namespace std;


// } Driver Code Ends
class Solution
{
    public:
    //Function to return the sorted array.
    vector <int> nearlySorted(int arr[], int num, int K){
        // Your code here
        vector<int> ans;
        priority_queue<int,vector<int>, greater<int>> pq;

        for(int i=0;i<K+1;i++)
        {
            pq.push(arr[i]);
```

```
        }
        //cout<<pq.top();
        for(int i=K+1;i<num;i++)
        {
            ans.push_back(pq.top());
            pq.pop();
            pq.push(arr[i]);

        }
        while(!pq.empty())
        {
            ans.push_back(pq.top());
            pq.pop();
        }
        //sort(ans.begin(),ans.end());
        return ans;
    }
};

//{ Driver Code Starts.

int main()
 {
  int T;
  cin>> T;

  while (T--)
  {
      int num, K;
      cin>>num>>K;

      int arr[num];
      for(int i = 0; i<num; ++i){
          cin>>arr[i];
      }
      Solution ob;
      vector <int> res = ob.nearlySorted(arr, num, K);
      for (int i = 0; i < res.size (); i++)
          cout << res[i] << " ";

      cout<<endl;
  }

  return 0;
}

// } Driver Code Ends

Time Complexity:O(nlogk)
Space Complexity:O(k)
```

K largest elements

Basic Accuracy: **61.15%** Submissions: **45K+** Points: **1**

Unable to Crack Interviews of Your Dream Companies ? Click Here to End This Problem!

Given an array of N positive integers, print k largest elements from the array.

Example 1:

Input:
N = 5, k = 2
arr[] = {12,5,787,1,23}
Output: 787 23
Explanation: First largest element in the array is 787 and the second largest is 23.

```
//{ Driver Code Starts
#include<bits/stdc++.h>
using namespace std;


// } Driver Code Ends


class Solution
{
    public:
    //Function to return k largest elements from an array.
    vector<int> kLargest(int arr[], int n, int k)
    {
        // code here
        priority_queue<int,vector<int>,greater<int>> pq;
```

```
        for(int i=0;i<k;i++)
        pq.push(arr[i]);

        for(int i=k;i<n;i++)
        {
            if(pq.top()<arr[i])
            {
                pq.pop();
                pq.push(arr[i]);
            }
        }
        vector<int> ans;
        while(!pq.empty())
        {
            ans.push_back(pq.top());
            pq.pop();
        }
        sort(ans.begin(),ans.end(),greater<int>());
        return ans;
    }
};

//{ Driver Code Starts.

int main(){
    int t;
    cin >> t;
    while(t--){
        int n, k;
        cin >> n >> k;

        int arr[n];
        for(int i = 0; i < n;i++)
            cin>>arr[i];
        Solution ob;
        vector<int> result = ob.kLargest(arr, n, k);
        for (int i = 0; i < result.size(); ++i)
            cout<<result[i]<<" ";
        cout << endl;

    }
    return 0;
}

// } Driver Code Ends

Time Complexity:O(nlogk)
Space Complexity:O(k)
```

Merge K Sorted Arrays

DS Contributed b
Dhruv Sharm

Medium | 80/80 | Avg time to solve 15 mins | Success Rate 85 % | Share | 67 upvot

Problem Statement Suggest Ed

You have been given 'K' different arrays/lists, which are sorted individually (in ascending order). You need to merge al the given arrays/list such that the output array/list should be sorted in ascending order.

Detailed explanation (Input/output format, Notes, Constraints, Images)

Sample Input 1:

```
1
2
3
3 5 9
4
1 2 3 8
```

Sample Output 1:

```
1 2 3 3 5 8 9
```

Explanation Of Sample Input 1:

```
After merging the two given arrays/lists [3, 5, 9] and [ 1, 2, 3, 8], the output sorted array
will be [1, 2, 3, 3, 5, 8, 9].
```

```
//Using Priority_Queue
#include <bits/stdc++.h>
#include <vector>
#include <queue>
vector<int> mergeKSortedArrays(vector<vector<int>>&vect, int k)
{
    // Write your code here.
    vector<int> ans;
    priority_queue<int,vector<int>,greater<int>> pq;
   for (int i = 0; i < vect.size(); i++)
    {
     for (int j = 0; j < vect[i].size(); j++) {
       pq.push(vect[i][j]);
     }
```

```
   }

   while(!pq.empty())
   {
       ans.push_back(pq.top());
       pq.pop();
   }
   return ans;
}
```

```
//Using vectors sort
#include <bits/stdc++.h>
#include <vector>
vector<int> mergeKSortedArrays(vector<vector<int>>&vect, int k)
{
    // Write your code here.
    vector<int> ans;
   for (int i = 0; i < vect.size(); i++)
    {
     for (int j = 0; j < vect[i].size(); j++) {
       ans.push_back(vect[i][j]);
     }
   }

   sort(ans.begin(),ans.end());
   return ans;
}

    Time Complexity: O((N * K) * log(N * K))
    Space Complexity: O(N * K)
```