NAME: Roshan Vijey K C
DEPT: EEE

1. Kth Smallest Element

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

class Solution {
  public:
    int kthSmallest(vector<int> &arr, int k) {
        // code here
        priority_queue<int>pq;
        for(int i = 0;i<arr.size();i++){
            if(pq.size() < k){
                pq.push(arr[i]);
            }
            else{
                if(pq.top() > arr[i]){
                    pq.pop();
                    pq.push(arr[i]);
                }
            }
        }
        return pq.top();
    }
};


int main() {
    int test_case;
    cin >> test_case;
    cin.ignore();
    while (test_case--) {
        int k;
        cin >> k;
        cin.ignore();
        vector<int> arr;
        string input;
        getline(cin, input);
        stringstream ss(input);
        int number;
        while (ss >> number) {
            arr.push_back(number);
        }
        int n = arr.size();
        Solution ob;
```

```
        cout << ob.kthSmallest(arr, k) << endl;
    }
    return 0;
}
```

OUTPUT:

The 4-th smallest element is 7

Complexity:

Time: O(nlogn)

Space: O(n)

2. Minimize the heights

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

class Solution {
  public:
    int getMinDiff(vector<int> &arr, int k) {
        sort(arr.begin(),arr.end());
        int ans = arr[arr.size()-1]-arr[0];
        for(int i = 0;i<arr.size();i++){
            if(arr[i]>=k){
                int minH = min(arr[0]+k,arr[i]-k);
                int maxH = max(arr[i-1]+k,arr[arr.size()-1]-k);
                ans = min(ans,maxH-minH);
            }
        }
        return ans;
    }
};

int main() {
    int t;
    cin >> t;
    cin.ignore();
    while (t--) {
        int k;
        cin >> k;
        cin.ignore();
        vector<int> a;
        string input;
        getline(cin, input);
        stringstream ss(input);
        int num;
        while (ss >> num)
            a.push_back(num);

        Solution ob;
        auto ans = ob.getMinDiff(a, k);
        cout << ans << endl;
    }
    return 0;
}
```

OUTPUT:

The minimum possible difference is: 5

Complexity:

Time: O(nlogn)

Space: O(n)

3. Paranthesis Chechup

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



class Solution {
  public:
    bool isParenthesisBalanced(string& s) {
        unordered_map<char,char>mp = {{')','('},{'}','{'},{']','['}};
        stack<char>st;
        for(int i = 0;i<s.size();i++){
            if(mp.find(s[i]) == mp.end()){
                st.push(s[i]);
            }
            else{
                if(!st.empty() && mp[s[i]] == st.top()){
                    st.pop();
                }
                else{
                    return false;
                }
            }
        }
        if(st.empty()){
            return true;
        }
```

```
        else{
            return false;
        }
    }
};



int main() {
    int t;
    string a;
    cin >> t;
    cin.ignore();
    while (t--) {
        cin >> a;
        cin.ignore();
        Solution obj;
        if (obj.isParenthesisBalanced(a))
            cout << "true" << endl;
        else
            cout << "false" << endl;


    }
}
```

OUTPUT:

The parentheses are balanced.

Complexity:

Time: O(n)

Space: O(n)

4. Equilibrium Points

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

class Solution {
```

```
  public:

    int equilibriumPoint(vector<int> &arr) {

        vector<int>pre(arr.size(),0);
        vector<int>suf(arr.size(),0);

        int t = 0;
        pre[0] = 0;
        for(int i = 1;i<arr.size();i++){
            t+=arr[i-1];
            pre[i] = t;
        }
        t = 0;
        suf[arr.size()-1] = 0;
        for(int i = arr.size()-2;i>=0;i--){
            t+=arr[i+1];
            suf[i] = t;
        }
        for(int i = 0;i<arr.size();i++){
            if(pre[i] == suf[i]){
                return i+1;
            }
        }
        return -1;
    }
};

int main() {
    int t;
    cin >> t;
    cin.ignore();
    while (t--)
    {
        vector<int> arr;
        string input;
        getline(cin, input);
        stringstream ss(input);
        int number;
        while (ss >> number) {
            arr.push_back(number);
        }

        Solution ob;
        cout << ob.equilibriumPoint(arr) << endl;
    }
}
```

OUTPUT:

Equilibrium point found at index: 2

Complexity:

Time: O(n)

Space: O(1)

5. Binary Search

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

class Solution {
  public:
    int binarysearch(vector<int> &arr, int k) {
        // code here
        int ans = -1;
        int l = 0,r=arr.size()-1;
        while(l<=r){
            int m = (l+r)/2;
            if(arr[m] == k){
                return m;
            }
            else if(arr[m]>k){
                r = m-1;
            }
            else{
                l = m+1;
            }
        }
        return -1;
    }
};



int main() {
```

```
    int t;
    cin >> t;
    while (t--) {
        int k;
        cin >> k;
        vector<int> arr;
        string input;
        cin.ignore();
        getline(cin, input);
        stringstream ss(input);
        int number;
        while (ss >> number) {
            arr.push_back(number);
        }
        Solution ob;
        int res = ob.binarysearch(arr, k);
        cout << res << endl;
    }
    return 0;
}
```

OUTPUT:

Element found at index: 3

Complexity:

Time:O(logn)

Space:O(1)

6. Next Greater Element

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

void NGE(vector<int>&nums,vector<int>&nge){
  stack<int>st;
```

```
  for(int i = nums.size()-1;i>=0;i--){
    while(!st.empty() && nums[st.top()]<=nums[i]){
      st.pop();
    }
    if(!st.empty()){
      nge[i] = nums[st.top()];
    }
    st.push(i);
  }
 }

 int main()
 {
  vector<int>nums = {13,7,6,12};
  vector<int>nge(nums.size(),-1);
  NGE(nums,nge);
  for(int i = 0;i<nums.size();i++){
    cout << nums[i] << " -> " << nge[i] << endl;
  }
  return 0;
}
```

OUTPUT:

4 --> 5

5 --> 10

2 --> 10

10 --> -1

8 --> -1

Complexity:

Time: O(n)

Space:O(n)

7. union of two arrays with duplicate elements

```
#include <bits/stdc++.h>
using namespace std;
class Solution {
  public:
    int findUnion(vector<int>& a, vector<int>& b) {
        unordered_set<int>st;
        for(auto i:a){
            st.insert(i);
        }
        for(auto i:b){
            st.insert(i);
        }
        return st.size();
    }
};

int main() {
    int t;
    cin >> t;
    cin.ignore();

    while (t--) {
        vector<int> a;
        vector<int> b;

        string input;
        getline(cin, input);
        stringstream ss(input);
        int number;
        while (ss >> number) {
            a.push_back(number);
        }

        getline(cin, input);
        stringstream ss2(input);
        while (ss2 >> number) {
            b.push_back(number);
        }

        Solution ob;
        cout << ob.findUnion(a, b) << endl;
    }

    return 0;
}
```

OUTPUT:

Union of two arrays:

[1, 2, 3, 4, 5, 6]

Complexity:

Time: O(n+m)

Space:O(n+m)