Day 4 Practice problems

1. **Kth Smallest**

Difficulty: **Medium**Accuracy: **35.17%**Submissions: **654K+**Points: **4**

Given an array **arr[]** and an integer **k** where k is smaller than the size of the array, the task is to find the **k^{th} smallest** element in the given array.

Follow up: Don't solve it using the inbuilt sort function.

Examples :

Input: arr[] = [7, 10, 4, 3, 20, 15], k = 3

Output: 7

Explanation: 3rd smallest element in the given array is 7.

Input: arr[] = [2, 3, 1, 20, 15], k = 4

Output: 15

Explanation: 4th smallest element in the given array is 15.

Expected Time Complexity: O(n+(max_element))

Expected Auxiliary Space: O(max_element)

Constraints:
$1 <= arr.size <= 10^6$
$1<= arr[i] <= 10^6$
$1 <= k <= n$

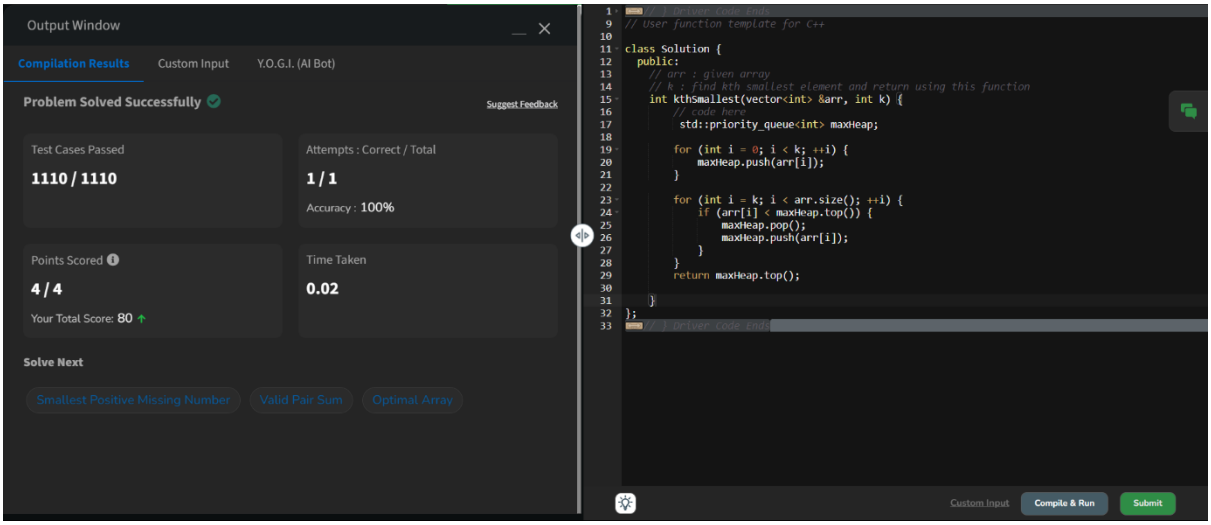
Code:

```
//{ 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
    // k : find kth smallest element and return using this function
    int kthSmallest(vector<int> &arr, int k) {
        // code here
```

```
        std::priority_queue<int> maxHeap;

        for (int i = 0; i < k; ++i) {
            maxHeap.push(arr[i]);
        }

        for (int i = k; i < arr.size(); ++i) {
            if (arr[i] < maxHeap.top()) {
                maxHeap.pop();
                maxHeap.push(arr[i]);
            }
        }
        return maxHeap.top();

    }
};

//{ Driver Code Starts.

int main() {
    int test_case;
    cin >> test_case;
    cin.ignore();
    while (test_case--) {

        int k;
        vector<int> arr, brr, crr;
        string input;
        getline(cin, input);
        stringstream ss(input);
        int number;
        while (ss >> number) {
            arr.push_back(number);
        }
        getline(cin, input);
```

```
        ss.clear();
        ss.str(input);
        while (ss >> number) {
            crr.push_back(number);
        }
        k = crr[0];
        int n = arr.size();
        Solution ob;
        cout << ob.kthSmallest(arr, k) << endl << "~\n";
    }
    return 0;
}

// } Driver Code Ends
```

Output:



2. **Minimize the Heights II**

Difficulty: **Medium**Accuracy: **15.06%**Submissions: **620K+**Points: **4**

Given an array **arr[]** denoting heights of **N** towers and a positive integer **K.**

For **each** tower, you must perform **exactly one** of the following operations **exactly once**.

- **Increase** the height of the tower by **K**
- **Decrease** the height of the tower by **K**

Find out the **minimum** possible difference between the height of the shortest and tallest towers after you have modified each tower.

You can find a slight modification of the problem here.
Note: It is **compulsory** to increase or decrease the height by K for each tower. **After** the operation, the resultant array should **not** contain any **negative integers**.

Examples :

Input: k = 2, arr[] = {1, 5, 8, 10}

Output: 5

Explanation: The array can be modified as {1+k, 5-k, 8-k, 10-k} = {3, 3, 6, 8}.The difference between the largest and the smallest is 8-3 = 5.

Input: k = 3, arr[] = {3, 9, 12, 16, 20}

Output: 11

Explanation: The array can be modified as {3+k, 9+k, 12-k, 16-k, 20-k} -> {6, 12, 9, 13, 17}.The difference between the largest and the smallest is 17-6 = 11.

Expected Time Complexity: O(n*logn)
Expected Auxiliary Space: O(n)

Constraints
$1 \le k \le 10^7$
$1 \le n \le 10^5$
$1 \le arr[i] \le 10^7$

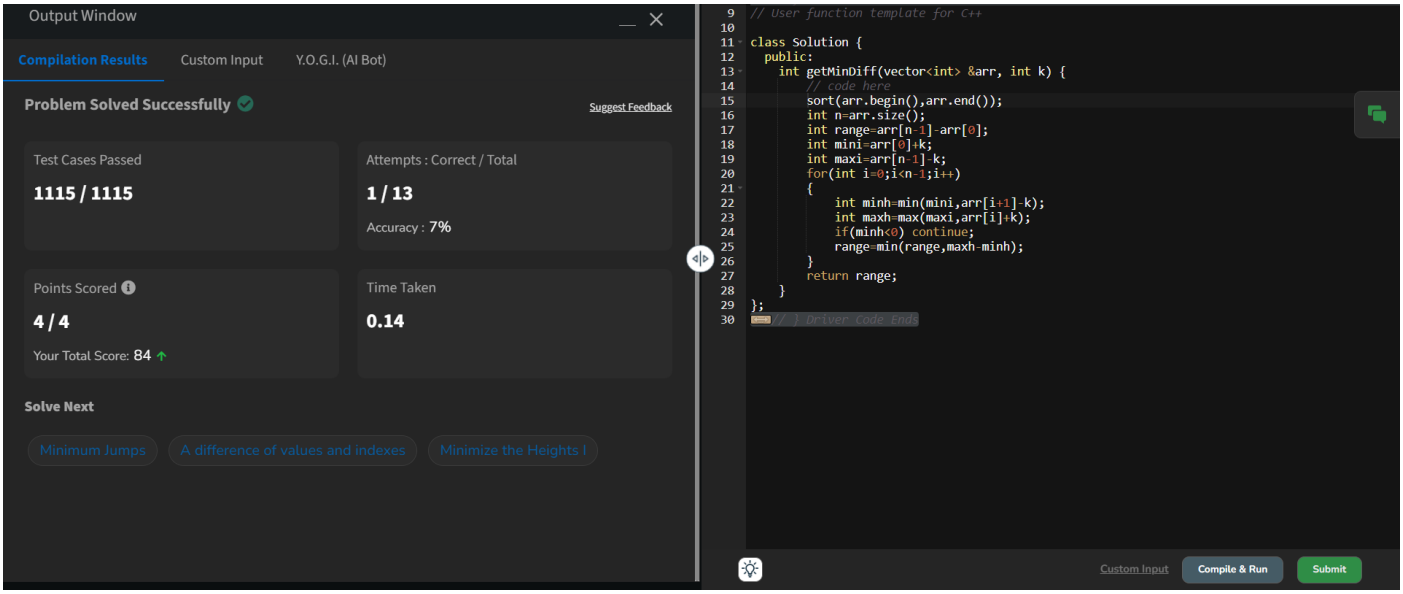
Try more examples

Code:

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

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



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

class Solution {
  public:
    int getMinDiff(vector<int> &arr, int k) {
        // code here
        sort(arr.begin(),arr.end());
        int n=arr.size();
        int range=arr[n-1]-arr[0];
        int mini=arr[0]+k;
        int maxi=arr[n-1]-k;
        for(int i=0;i<n-1;i++)
```

```
        {
            int minh=min(mini,arr[i+1]-k);
            int maxh=max(maxi,arr[i]+k);
            if(minh<0) continue;
            range=min(range,maxh-minh);
        }
        return range;
    }
};

//{ Driver Code Starts.
int main() {
    int t;
    cin >> t;
    cin.ignore();
    while (t--) {
        int n, k;
        cin >> k;
        cin.ignore();
        vector<int> a, b, c, d;
        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 << "\n";
        cout << '~' << endl;
    }
    return 0;
}
// } Driver Code Ends
```

Output:



3. **Parenthesis Checker**

Difficulty: **Easy**Accuracy: **28.56%**Submissions: **617K+**Points: **2**

You are given a string **s** representing an expression containing various types of brackets: {}, (), and []. Your task is to determine whether the brackets in the expression are balanced. A balanced expression is one where every opening bracket has a corresponding closing bracket in the correct order.

Examples :

Input: s = "{([])}"

Output: true

Explanation:
- In this expression, every opening bracket has a corresponding closing bracket.
- The first bracket { is closed by }, the second opening bracket (is closed by), and the third opening bracket [is closed by].
- As all brackets are properly paired and closed in the correct order, the expression is considered balanced.

Input: s = "()"

Output: true

Explanation:
- This expression contains only one type of bracket, the parentheses (and).
- The opening bracket (is matched with its corresponding closing bracket).
- Since they form a complete pair, the expression is balanced.

Input: s = "([]"

Output: false

Explanation:
- This expression contains only one type of bracket, the parentheses (and).
- The opening bracket (is matched with its corresponding closing bracket).
- Since they form a complete pair, the expression is balanced.

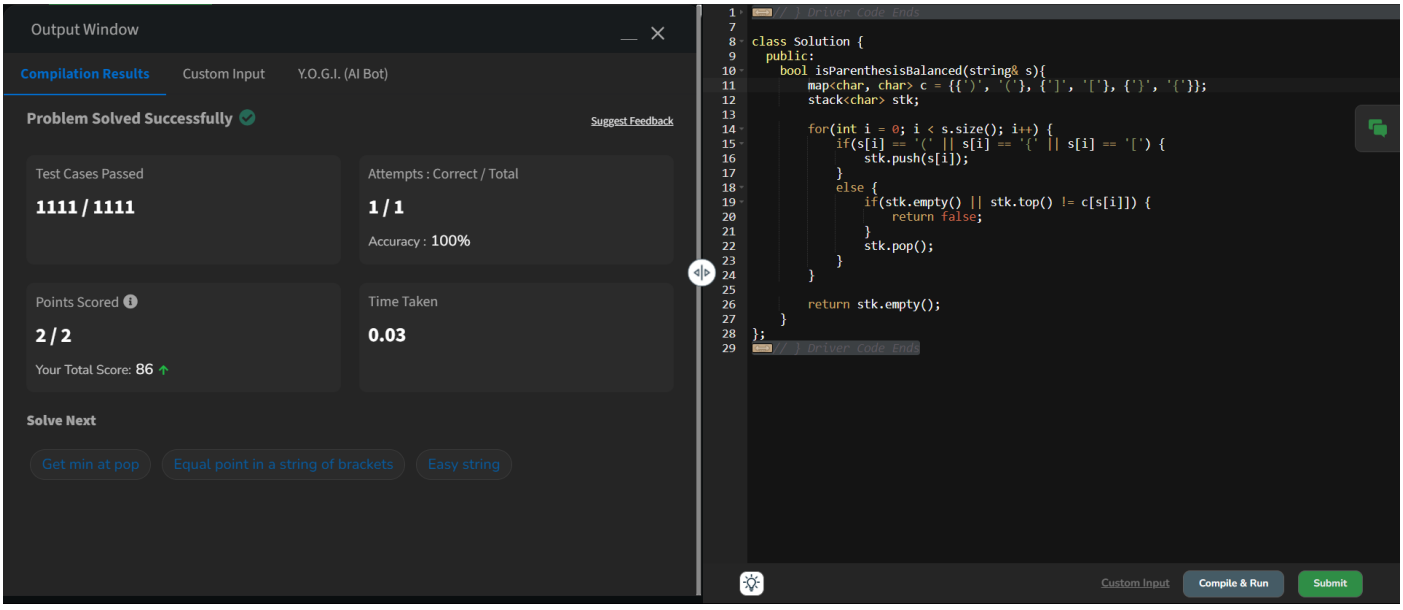
Constraints:
$1 \le s.size() \le 10^6$
s[i] ∈ {'{', '}', '(', ')', '[', ']'}

Code:

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



// } Driver Code Ends

class Solution {
  public:
    bool isParenthesisBalanced(string& s){
        map<char, char> c = {{')', '('}, {']', '['}, {'}', '{'}};
        stack<char> stk;

        for(int i = 0; i < s.size(); i++) {
            if(s[i] == '(' || s[i] == '{' || s[i] == '[') {
                stk.push(s[i]);
            }
            else {
                if(stk.empty() || stk.top() != c[s[i]]) {
                    return false;
                }
                stk.pop();
            }
        }

        return stk.empty();
    }
};

//{ Driver Code Starts.

int main() {
```

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

        cout << "~"
             << "\n";
    }
}
// } Driver Code Ends
```

Output:



4. **Equilibrium Point**

Difficulty: **Easy**Accuracy: **28.13%**Submissions: **593K+**Points: **2**

Given an array **arr** of non-negative numbers. The task is to find the first **equilibrium point** in an array. The equilibrium point in an array is an index (or position) such that the sum of all elements before that index is the same as the sum of elements after it.

Note: Return equilibrium point in 1-based indexing. Return -1 if no such point exists.

Examples:

Input: arr[] = [1, 3, 5, 2, 2]
Output: 3

Explanation: The equilibrium point is at position 3 as the sum of elements before it (1+3) = sum of elements after it (2+2).

Input: arr[] = [1]
Output: 1

Explanation: Since there's only one element hence it's only the equilibrium point.

Input: arr[] = [1, 2, 3]
Output: -1

Explanation: There is no equilibrium point in the given array.

Expected Time Complexity: O(n)
Expected Auxiliary Space: O(1)

Constraints:
1 <= arr.size <= 10^6
0 <= arr[i] <= 10^9

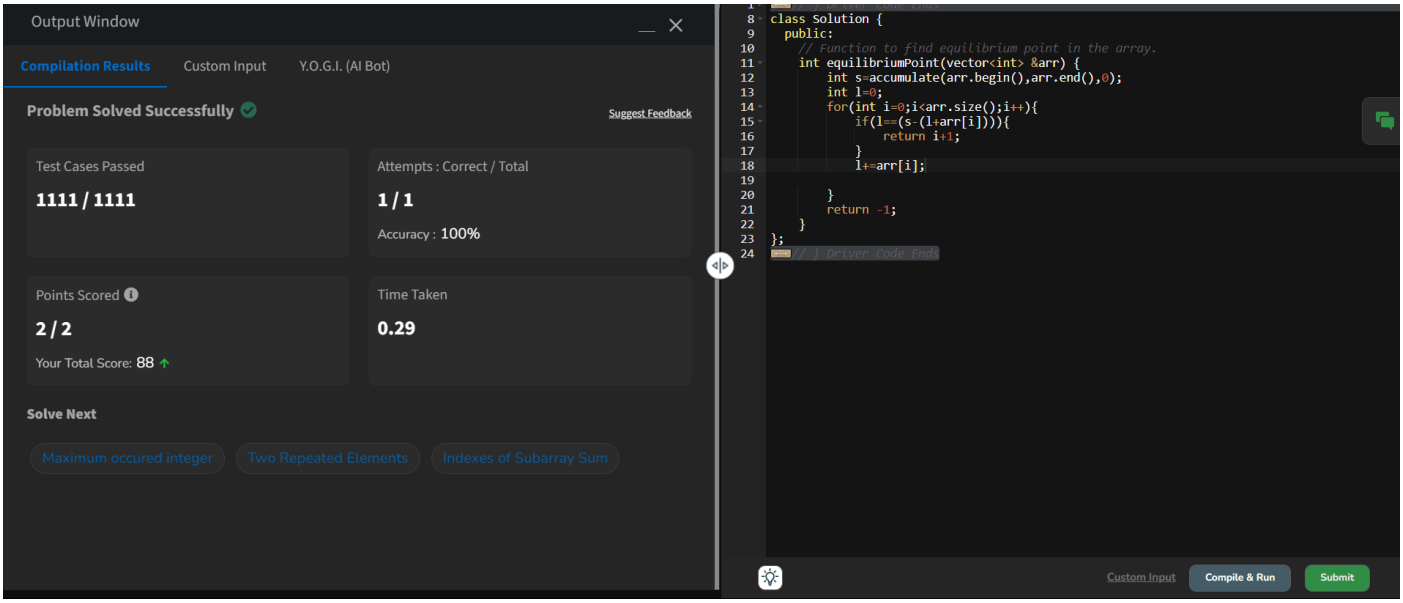
Try more examples

Code:

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



// } Driver Code Ends
class Solution {
  public:
    // Function to find equilibrium point in the array.
    int equilibriumPoint(vector<int> &arr) {
        int s=accumulate(arr.begin(),arr.end(),0);
        int l=0;
        for(int i=0;i<arr.size();i++){
            if(l==(s-(l+arr[i]))){
                return i+1;
            }
            l+=arr[i];

        }
```

```
        return -1;
    }
};

//{ Driver Code Starts.

int main() {
    int t;
    cin >> t;
    cin.ignore(); // To discard any leftover newline characters
    while (t--)   // while testcases exist
    {
        vector<int> arr;
        string input;
        getline(cin, input); // Read the entire line for the array elements
        stringstream ss(input);
        int number;
        while (ss >> number) {
            arr.push_back(number);
        }

        Solution ob;
        cout << ob.equilibriumPoint(arr) << endl;
        cout << "~" << endl;
    }
}
// } Driver Code Ends
```

Output:



5.

Binary Search

Difficulty: **Easy**Accuracy: **44.32%**Submissions: **530K+**Points: **2**

Given a sorted array **arr** and an integer **k**, find the position(0-based indexing) at which k is present in the array using binary search.

Note: If multiple occurrences are there, please return the smallest index.

Examples:

Input: arr[] = [1, 2, 3, 4, 5], k = 4

Output: 3

Explanation: 4 appears at index 3.

Input: arr[] = [11, 22, 33, 44, 55], k = 445

Output: -1

Explanation: 445 is not present.

Note: Try to solve this problem in constant space i.e O(1)

Constraints:
1 <= arr.size() <= 10^5
1 <= arr[i] <= 10^6
1 <= k <= 10^6

//{ Driver Code Starts

// Initial Template for C++

#include <bits/stdc++.h>

using namespace std;

Code:

//{ Driver Code Starts

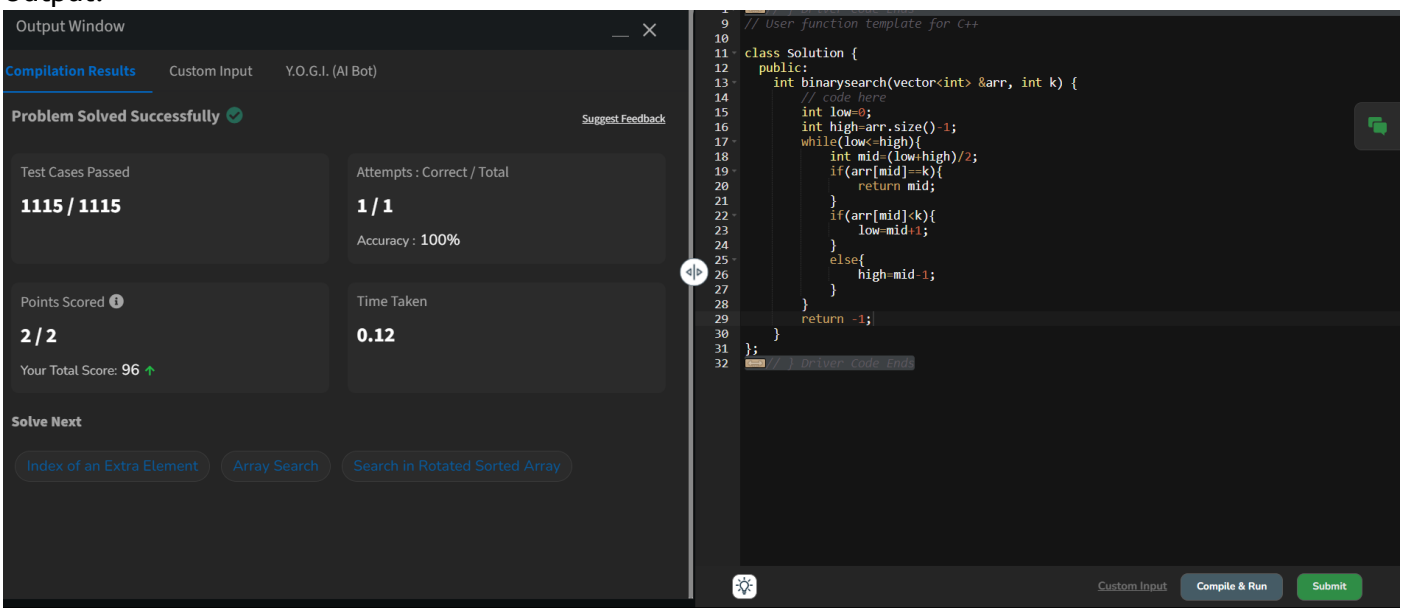
// Initial template for C++

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



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

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

//{ Driver Code Starts.
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;
        cout << "~" << endl;
    }
    return 0;
}
```

// } Driver Code Ends

Output:



6. **Next Greater Element**

Difficulty: **Medium**Accuracy: **32.95%**Submissions: **410K+**Points: **4**

Given an array **arr[ ]** of integers, the task is to find the next greater element for each element of the array in order of their appearance in the array. Next greater element of an element in the array is the nearest element on the right which is greater than the current element.

If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

Examples

Input: arr[] = [1, 3, 2, 4]

Output: [3, 4, 4, -1]

Explanation: The next larger element to 1 is 3, 3 is 4, 2 is 4 and for 4, since it doesn't exist, it is -1.

Input: arr[] = [6, 8, 0, 1, 3]

Output: [8, -1, 1, 3, -1]

Explanation: The next larger element to 6 is 8, for 8 there is no larger elements hence it is -1, for 0 it is 1 , for 1 it is 3 and then for 3 there is no larger element on right and hence -1.

Input: arr[] = [10, 20, 30, 50]

Output: [20, 30, 50, -1]

Explanation: For a sorted array, the next element is next greater element also exxept for the last element.

Input: arr[] = [50, 40, 30, 10]

Output: [-1, -1, -1, -1]

Explanation: There is no greater element for any of the elements in the array, so all are -1.

Constraints:
$1 \leq arr.size() \leq 10^6$
$0 \leq arr[i] \leq 10^9$

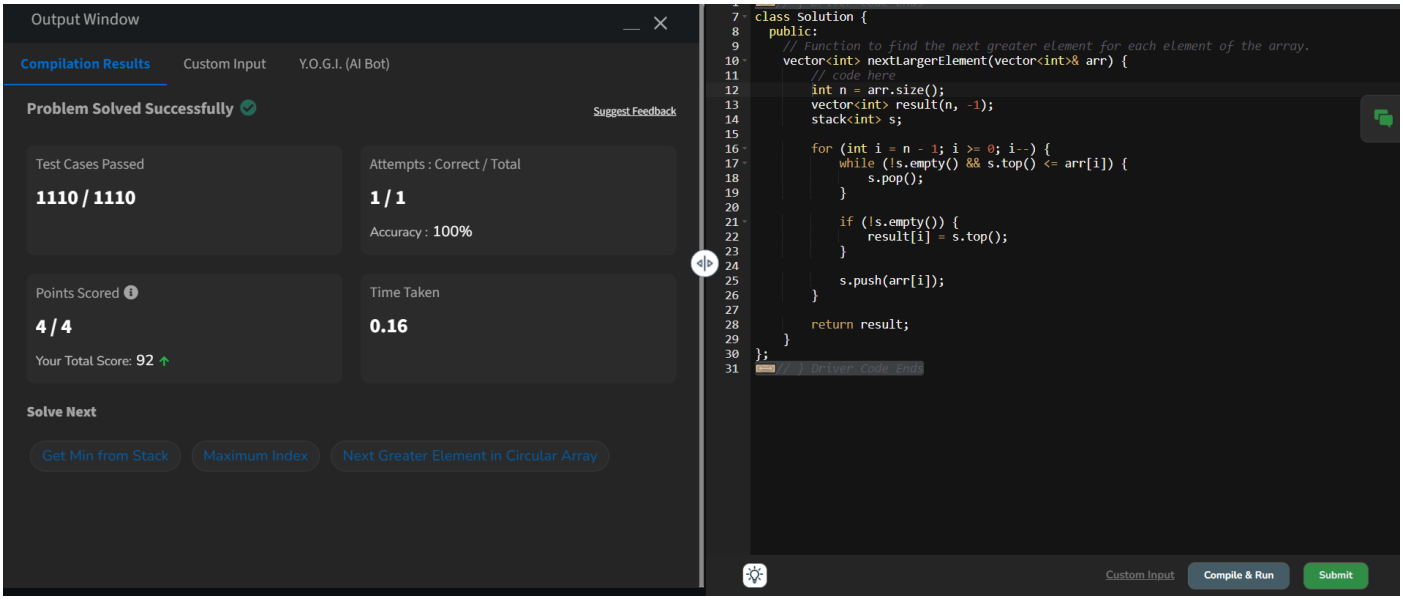
Code:

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



// } Driver Code Ends
class Solution {
  public:
    // Function to find the next greater element for each element of the array.
    vector<int> nextLargerElement(vector<int>& arr) {
        // code here
        int n = arr.size();
        vector<int> result(n, -1);
        stack<int> s;
```

```
        for (int i = n - 1; i >= 0; i--) {
            while (!s.empty() && s.top() <= arr[i]) {
                s.pop();
            }

            if (!s.empty()) {
                result[i] = s.top();
            }

            s.push(arr[i]);
        }

        return result;
    }
};

//{ Driver Code Starts.

int main() {
    int t; // Number of test cases
    cin >> t;
    cin.ignore(); // Ignore the newline after reading t
    while (t--) {
        vector<int> a;
        string input;

        // Reading the entire input line for the array
        getline(cin, input);
        stringstream ss(input);
        int num;
        while (ss >> num)
            a.push_back(num); // Read the array elements from input string

        Solution obj;
        vector<int> result = obj.nextLargerElement(a);
```

```
        // Print the result in the required format
        for (int i = 0; i < result.size(); i++) {
            if (i != 0)
                cout << " ";
            cout << result[i];
        }
        cout << endl;        // Ensure new line after each test case output
        cout << "~" << endl; // Ensure new line after each test case output
    }

    return 0;
}

// } Driver Code Ends
```

Output:



7. **Union of Two Arrays with Duplicate Elements**

Difficulty: **Easy**Accuracy: **42.22%**Submissions: **387K+**Points: **2**

Given two arrays **a[]** and **b[]**, the task is to find the number of elements in the union between these two arrays.

The Union of the two arrays can be defined as the set containing distinct elements from both arrays. If there are repetitions, then only one element occurrence should be there in the union.

Note: Elements are not necessarily distinct.

Examples

Input: a[] = [1, 2, 3, 4, 5], b[] = [1, 2, 3]

Output: 5

Explanation: 1, 2, 3, 4 and 5 are the elements which comes in the union setof both arrays. So count is 5.

Input: a[] = [85, 25, 1, 32, 54, 6], b[] = [85, 2]
Output: 7

Explanation: 85, 25, 1, 32, 54, 6, and 2 are the elements which comes in the union set of both arrays. So count is 7.

Input: a[] = [1, 2, 1, 1, 2], b[] = [2, 2, 1, 2, 1]
Output: 2

Explanation: We need to consider only distinct. So count is 2.

Constraints:
$1 \leq a.size(), b.size() \leq 10^6$
$0 \leq a[i], b[i] < 10^5$

Code:

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


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



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


class Solution {
  public:
    // Function to return the count of number of elements in union of two arrays.
    int findUnion(vector<int>& a, vector<int>& b) {
        // code here
        set<int> s;
        int i=0;
        int n1=a.size();
        int n2=b.size();
        while(i<n1 || i<n2){
            if(i<n1){
                s.insert(a[i]);
            }
            if(i<n2){
```

```
            s.insert(b[i]);
        }
        i++;
    }
    return s.size();
  }
};

//{ Driver Code Starts.

int main() {
  int t;
  cin >> t;
  cin.ignore(); // Ignore the newline character after reading t

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

    string input;
    // For a
    getline(cin, input); // Read the entire line for the array elements
    stringstream ss(input);
    int number;
    while (ss >> number) {
      a.push_back(number);
    }

    // For b
    getline(cin, input); // Read the entire line for the array elements
    stringstream ss2(input);
    while (ss2 >> number) {
      b.push_back(number);
    }
```

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

    return 0;
}
// } Driver Code Ends
```

Output: