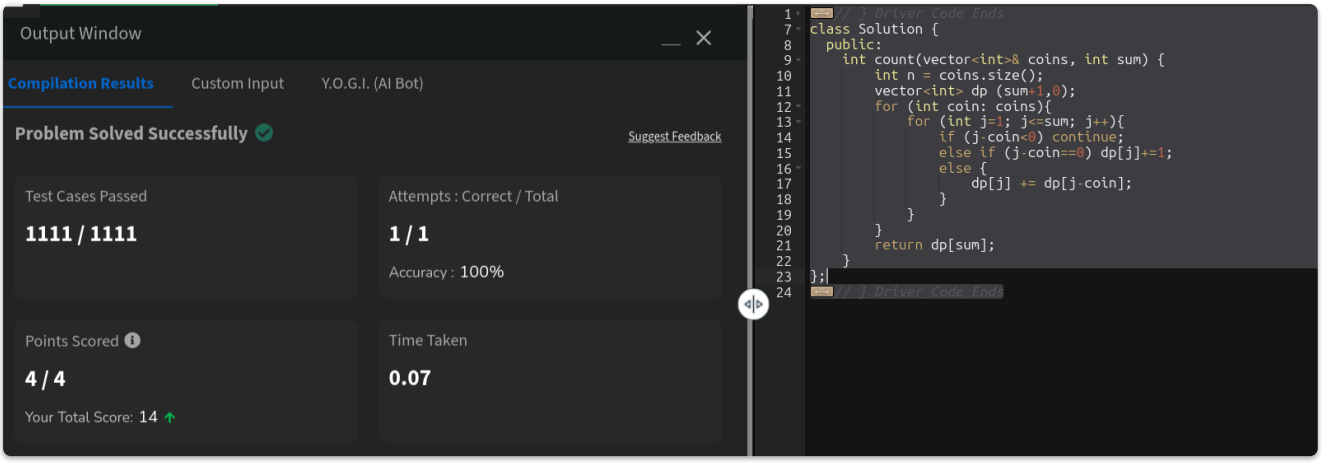
Coin Change (Counting ways)

```
class Solution {
  public:
    int count(vector<int>& coins, int sum) {
        int n = coins.size();
        vector<int> dp (sum+1,0);
        for (int coin: coins){
            for (int j=1; j<=sum; j++){
                if (j-coin<0) continue;
                else if (j-coin==0) dp[j]+=1;
                else {
                    dp[j] += dp[j-coin];
                }
            }
        }
        return dp[sum];
    }
};
```

2D DP: `dp[i][j]=dp[i-1][j]+dp[i][j-coin]`
1D DP: Replacing on same for every coin `dp[i]+=dp[i-coin]`

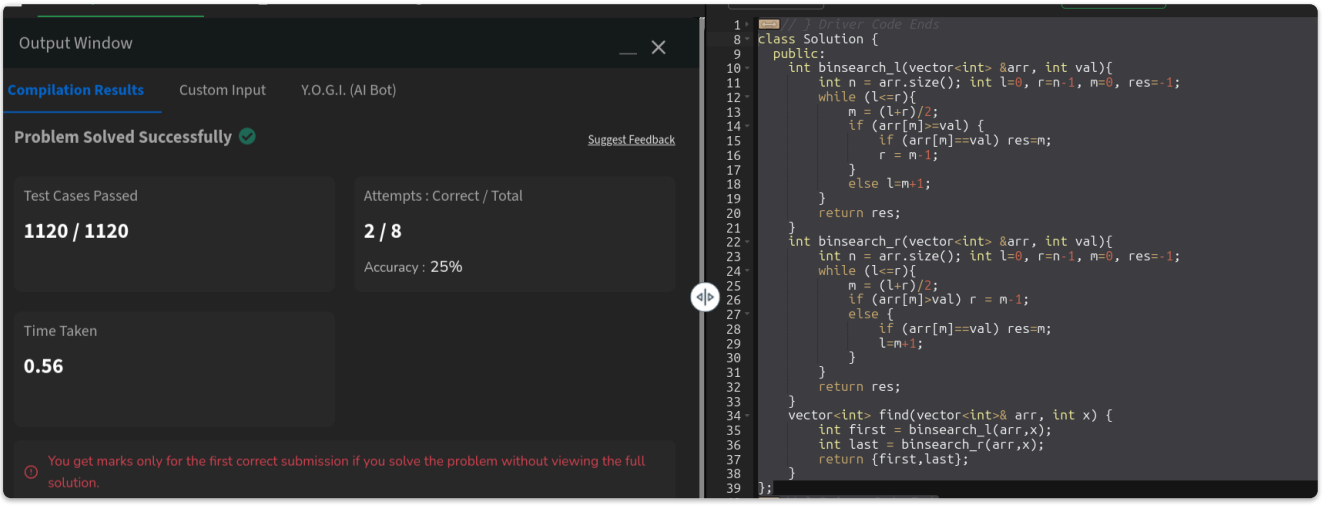


Time Complexity: $O(n * sum)$
Space Complexity: $O(sum)$

First and Last Occurences

```
class Solution {
  public:
    int binsearch_l(vector<int> &arr, int val){
        int n = arr.size(); int l=0, r=n-1, m=0, res=-1;
        while (l<=r){
            m = (l+r)/2;
```

```
            if (arr[m]>=val) {
                if (arr[m]==val) res=m;
                r = m-1;
            }
            else l=m+1;
        }
        return res;
    }
    int binsearch_r(vector<int> &arr, int val){
        int n = arr.size(); int l=0, r=n-1, m=0, res=-1;
        while (l<=r){
            m = (l+r)/2;
            if (arr[m]>val) r = m-1;
            else {
                if (arr[m]==val) res=m;
                l=m+1;
            }
        }
        return res;
    }
    vector<int> find(vector<int>& arr, int x) {
        int first = binsearch_l(arr,x);
        int last = binsearch_r(arr,x);
        return {first,last};
    }
};
```

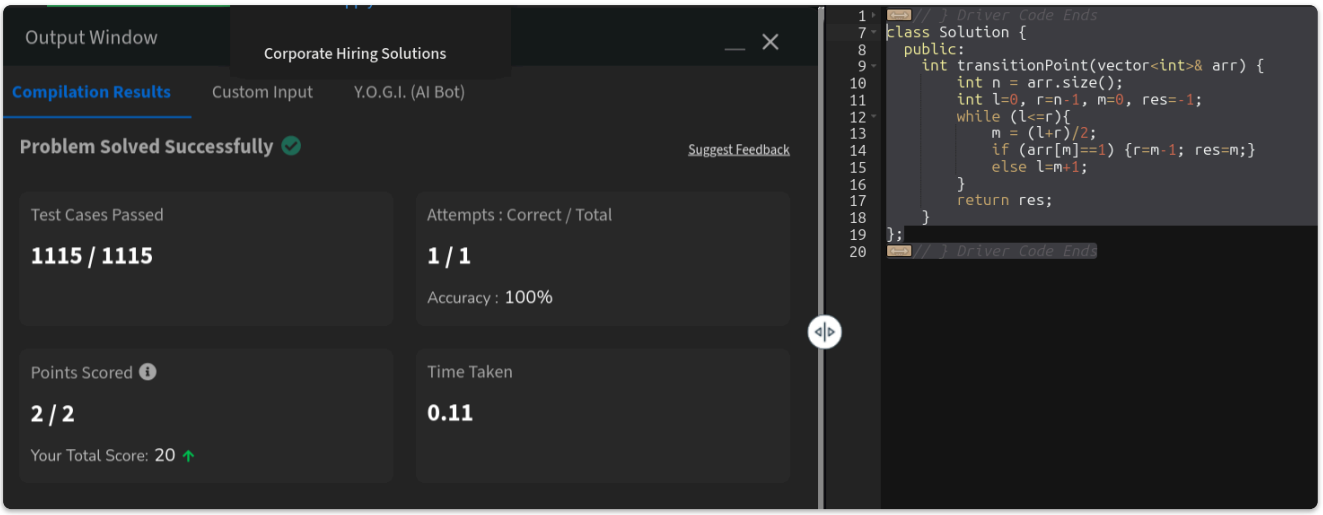


Time Complexity: $O(log(N))$
Space Complexity: $O(1)$

Find Transition Point

```
class Solution {
  public:
    int transitionPoint(vector<int>& arr) {
        int n = arr.size();
```

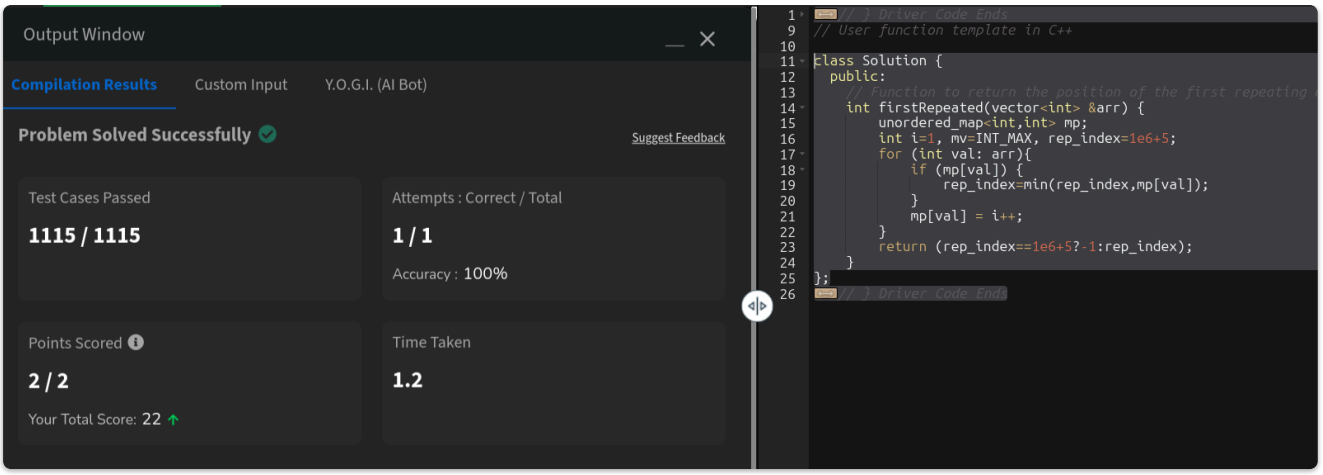
```
        int l=0, r=n-1, m=0, res=-1;
        while (l<=r){
            m = (l+r)/2;
            if (arr[m]==1) {r=m-1; res=m;}
            else l=m+1;
        }
        return res;
    }
};
```



Time Complexity: $O(log(N))$
Space Complexity: $O(1)$

First Repeating Element

```
class Solution {
  public:
    // Function to return the position of the first repeating element.
    int firstRepeated(vector<int> &arr) {
        unordered_map<int,int> mp;
        int i=1, mv=INT_MAX, rep_index=1e6+5;
        for (int val: arr){
            if (mp[val]) {
                rep_index=min(rep_index,mp[val]);
            }
            mp[val] = i++;
        }
        return (rep_index==1e6+5?-1:rep_index);
    }
};
```



Time Complexity: $O(N)$
Space Complexity: $O(N)$

Remove Duplicates sorted array

```
class Solution {
  public:
    int remove_duplicate(vector<int> &arr) {
        auto i = arr.begin();
        int n = arr.size(), el = -1, rep=0;
        while (i!=arr.end()){
            if (el==arr[i-arr.begin()]) {i=arr.erase(i);rep++;}
            else {el=arr[i-arr.begin()];i++;}
        }
        return n-rep;
    }
};
```

Time Complexity: $O(N)$
Space Complexity: $O(1)$

--- Java

Maximum Index

```
int maxIndexDiff(int[] arr) {

        int[] Lmin = new int[arr.length];
        int[] Lmax = new int[arr.length];

        Lmin[0] = arr[0];
        Lmax[0] = arr[0];
        for(int i = 1; i < arr.length; i++){
            Lmin[i] = Math.min(arr[i], Lmin[i-1]);
        }
```

```
        Lmax[arr.length - 1] = arr[arr.length - 1];
        for(int i = arr.length-2 ; i >= 0; i--){
            Lmax[i] = Math.max(arr[i], Lmax[i+1]);
        }

        int i = 0, j = 0, maxLength = -1;
        while(i < arr.length && j < arr.length){
            if(Lmin[i] <= Lmax[j]){
                maxLength = Math.max(maxLength, j-i);
                j++;
            }
            else{
                i++;
            }
        }

        return maxLength;
    }
```

Wave Array

```
public static void convertToWave(int[] arr) {
        int n = arr.length;
        int i = 0, j = 1;
        while(n > 1){
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            n-=2;
            i+=2;
            j+=2;
        }
    }
```