**Kth Smallest element**

**CODE :**

class Solution {

public static int kthSmallest(int[] arr, int k) {

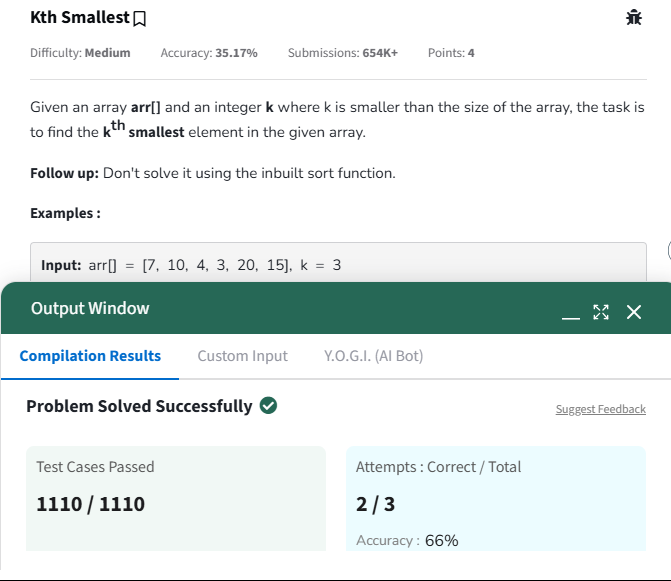
Arrays.sort(arr);

return arr[k-1];

}

}

OUTPUT :



TIME COMPLEXITY : O(nlogn)

SPACE COMPLEXITY : O(1)

**MINIMIZE THE HEIGHTS II**

**CODE :**

class Solution {

int getMinDiff(int[] arr, int k){

int n =arr.length;

if( n == 1){

return 0;

}

Arrays.sort(arr);

int ans = arr[n-1] - arr[0];

int min = arr[0] + k;

int max = arr[n-1] - k;

for(int i=0; i<n-1; i++){

int min\_ele = Math.min(min, arr[i+1]-k);

int max\_ele = Math.max(max, arr[i]+k);

if(min\_ele < 0)

continue;

ans = Math.min(ans, max\_ele - min\_ele);

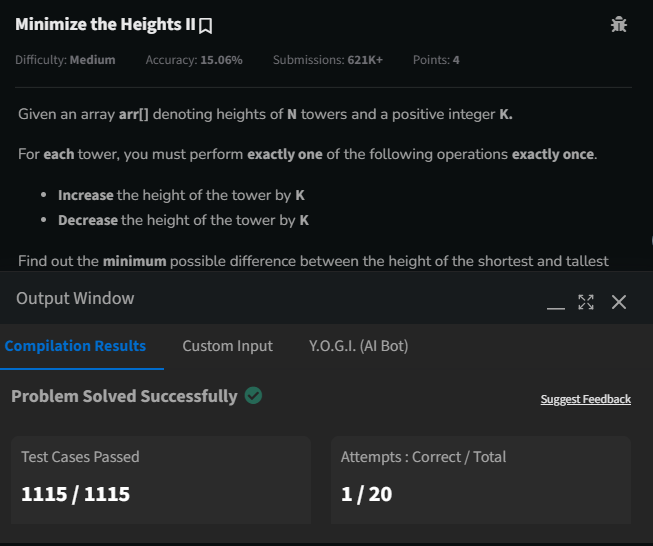
}

return ans;

}

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

**PARENTHESIS CHECKER :**

**CODE :**

**class Solution {**

// Function to check if brackets are balanced or not.

static boolean isParenthesisBalanced(String s) {

Stack<Character> st = new Stack<>();

for(char ch : s.toCharArray()) {

if(ch == '(') {

st.push(')');

} else if(ch == '{') {

st.push('}');

} else if(ch == '[') {

st.push(']');

} else {

if(st.isEmpty() || st.pop() != ch) {

return false;

}

}

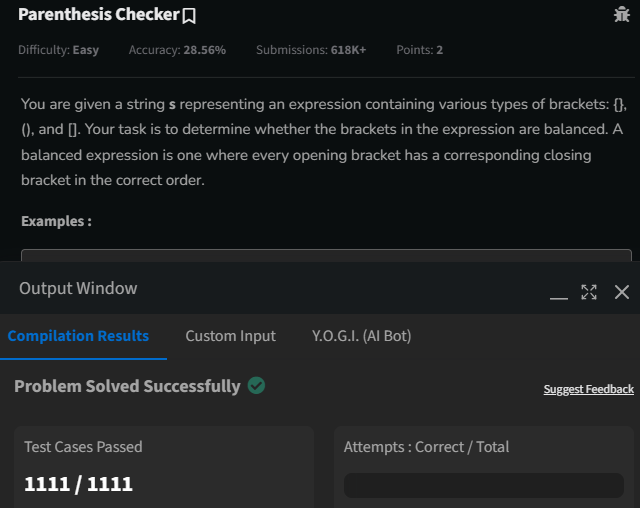
}

return st.isEmpty();

}

}

OUTPUT :



TIME COMPLEXITY : O(N)

SPACE COMEPLEXITY : O(N)

**EQUILLIBRIUM POINT :**

**CODE :**

public class equilibrium {

    public static void main(String[] args) {

        System.out.println(find(new int[]{1, 3, 5, 2, 2}));

        System.out.println(find(new int[]{1, 2, 3}));

    }

    public static int find(int arr[]) {

        int lsum = 0 , rsum = 0 ;

        int i = 0 , j = arr.length - 1;

        lsum += arr[i];

        rsum += arr[j];

        while(i+1<j-1){

            if(lsum<rsum){

                i++;

                lsum += arr[i];

            }

            if(rsum<lsum){

                j--;

                rsum += arr[j];

            }

            if(rsum==lsum){

                i++;

                lsum += arr[i];

                j--;

                rsum += arr[j];

            }

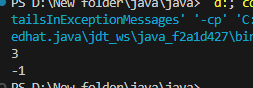
        }

        return lsum==rsum ? i+1 : -1;

    }

}

**OUTPUT :**

****

TIME COMPLEXITY : O(N)

SPACE COMPLEXITY : O(N)

**BINARY SEARCH :**

**CODE :**

public class binarysearch {

    public static void main(String[] args) {

        System.out.println(bs(new int[]{ 2, 3, 4, 10, 40 },10));

    }

    static int bs(int arr[], int x)

    {

        int l = 0, r = arr.length - 1;

        while (l <= r) {

            int mid = l + (r - l) / 2;

            if (arr[mid] == x)

                return mid;

            if (arr[mid] < x)

                l = mid + 1;

            else

                r = mid - 1;

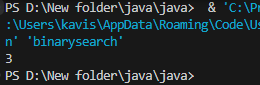
        }

        return -1;

    }

}

**OUTPUT :**

****

TIME COMPLEXITY : O(N)

SPACE COMPLEXITY : O(N)

**NEXT GREATER ELEMENT :**

**CODE :**

class Solution {

// Function to find the next greater element for each element of the array.

public ArrayList<Integer> nextLargerElement(int[] arr) {

int n = arr.length;

ArrayList<Integer> res = new ArrayList<>();

Stack<Integer> stack = new Stack<>();

for(int i=0;i<n;i++){

res.add(-1);

}

for(int i=0;i<n;i++){

while(!stack.isEmpty() && arr[stack.peek()]<arr[i]){

int ind=stack.pop();

res.set(ind,arr[i]);

}

stack.push(i);

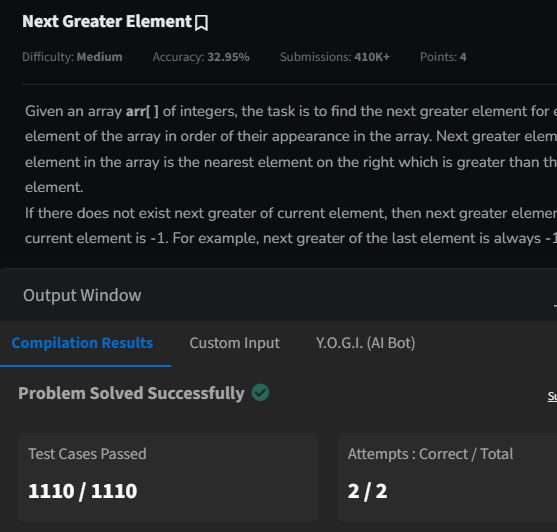
}

return res;

}

}

OUTPUT :



TIME COMPLEXITY : O(N)

SPACE COMPLEXITY : O(N)

**UINON OF TWO ARRAYS WITH DUPLICATE ELELMENTS :**

**CODE :**

import java.util.HashSet;

import java.util.Set;

public class unionarrays {

    public static void main(String[] args) {

        System.out.println(findUnion(new int[]{1, 2, 3}, new int[]{2, 3, 4}));

    }

    public static int findUnion(int a[], int b[]) {

        Set<Integer> s = new HashSet<>();

        for(int i : a){

            s.add(i);

        }

        for(int j : b){

            s.add(j);

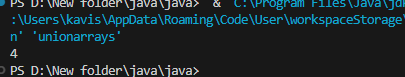
        }

        return s.size();

    }

}

**OUTPUT :**

****

**TIME COMPLEXTY :O(n)**

**SPACE COMPLEXITY :O(N);**