Date: 13/11/2024

**DSA Practice Problems**

1. **Kth Smallest Element**

class Solution {

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

int N=arr.length;

PriorityQueue<Integer> pq = new PriorityQueue<>((a, b) -> b - a);

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

pq.add(arr[i]);

if (pq.size() > k){

pq.poll();

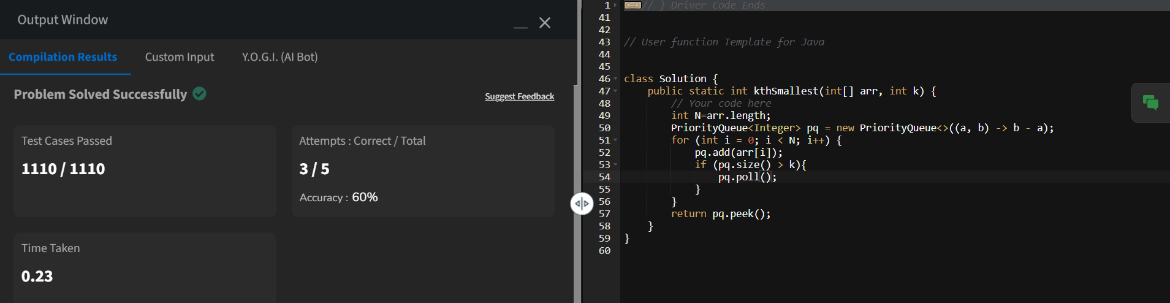
}

}

return pq.peek();

}

}



Time Complexity: O (N log N)

1. **Minimize the Height ll**

class Solution {

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

int n = arr.length;

Arrays.sort(arr);

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

int smallest = arr[0] + k;

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

if (smallest > largest) {

int temp = smallest;

smallest = largest;

largest = temp;

}

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

int min\_value = Math.min(smallest, arr[i] - k);

int max\_value = Math.max(largest, arr[i - 1] + k);

if (min\_value < 0) {

continue;

}

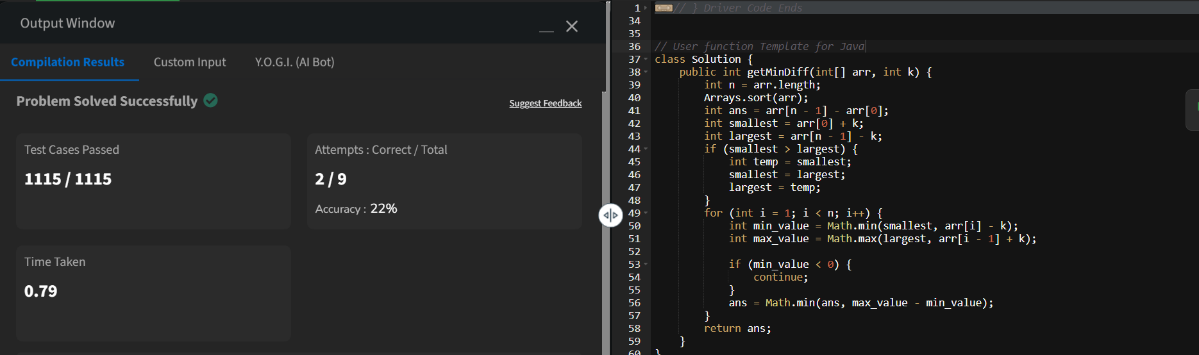
ans = Math.min(ans, max\_value - min\_value);

}

return ans;

}

}



Time Complexity: O(n log n)

1. **Paranthesis checker**

class Solution {

static boolean isParenthesisBalanced(String s) {

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

for (int i = 0; i < s.length(); i++) {

char c = s.charAt(i);

if (c == '(' || c == '{' || c == '[') {

stack.push(c);

} else {

if (stack.isEmpty()) return false;

char top = stack.peek();

if ((top == '(' && c == ')') ||

(top == '{' && c == '}') ||

(top == '[' && c == ']')) {

stack.pop();

} else {

return false;

}

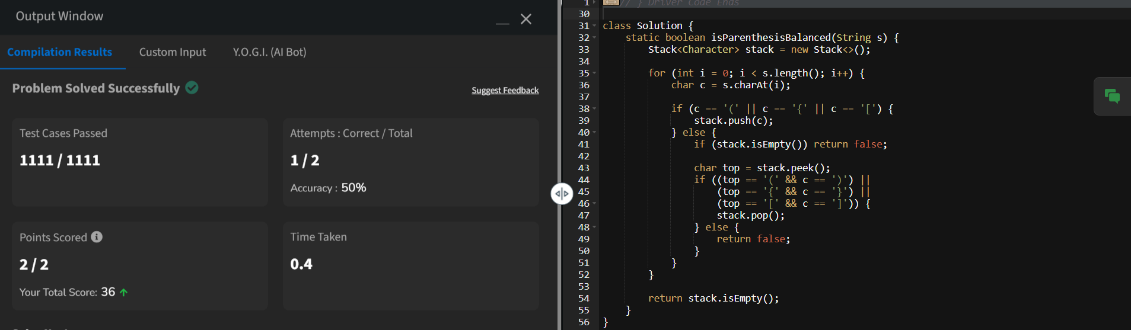
}

}

return stack.isEmpty();

}

}



Time Complexity: O(n)

1. **Equilibrium Point**

class Solution {

public int equilibriumPoint(int[] arr) {

int n = arr.length;

if (n == 1) {

return 1;

}

int total = 0;

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

total += arr[i];

}

int ls = 0;

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

int rs = total - ls - arr[i];

if (ls == rs) {

return i + 1;

}

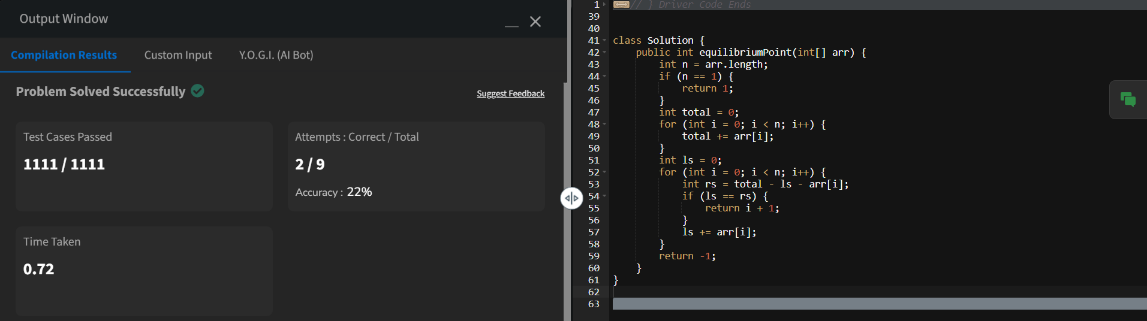
ls += arr[i];

}

return -1;

}

}



Time Complexity: O(n)

1. **Binary Search**

class Solution {

public int binarysearch(int[] arr, int k) {

int left = 0;

int right = arr.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == k) {

return mid;

}

if (arr[mid] < k) {

left = mid + 1;

} else {

right = mid - 1;

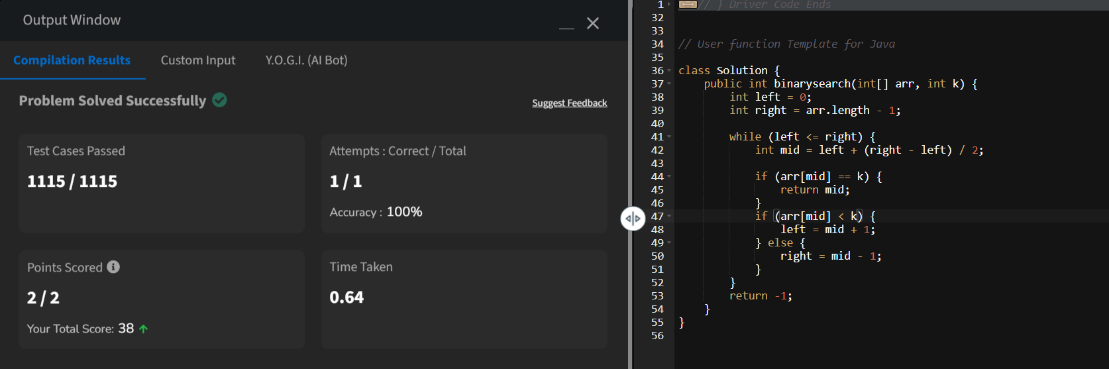
}

}

return -1;

}

}



Time Complexity: O(log n)

1. **Next Greater Element**

class Solution {

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

int n = arr.length;

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

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

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

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

stack.pop();

}

if (stack.isEmpty()) {

res.add(-1);

} else {

res.add(stack.peek());

}

stack.push(arr[i]);

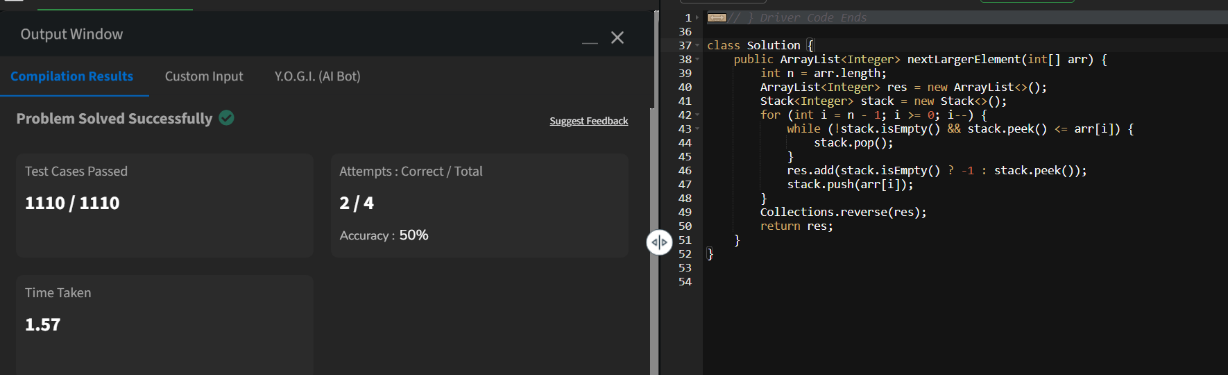
}

Collections.reverse(res);

return res;

}

}



Time Complexity: O(n)

1. **Union of two arrays**

class Solution {

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

// code here

HashSet<Integer> set = new HashSet<>();

for (int num : a) {

set.add(num);

}

for (int num : b) {

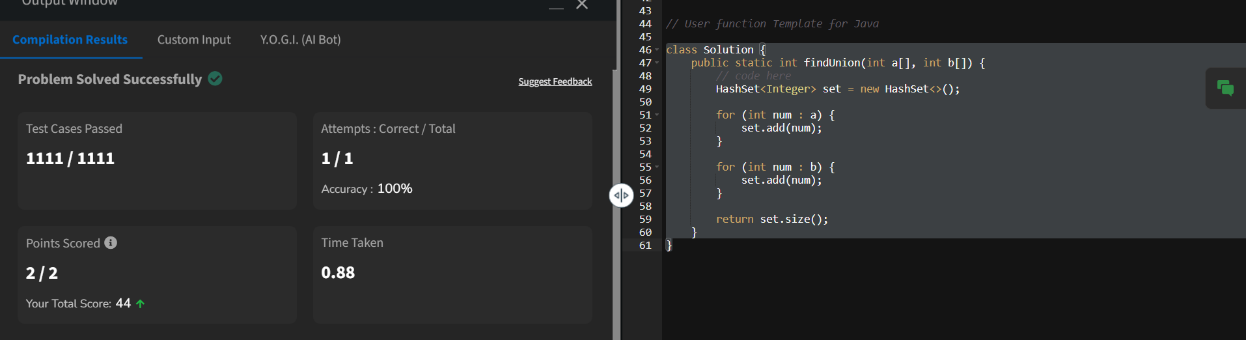
set.add(num);

}

return set.size();

}

}



Time Complexity: O(n+m)