**Experiment - 8**

**Name: Armaan UID: 22BCS12007**

**Aim:** To solve leet code problems

1. Problem : Max Units on a Truck

Code:

class Solution {

        public int maximumUnits(int[][] boxTypes, int truckSize) {

        Arrays.sort(boxTypes, (a, b) -> Integer.compare(b[1], a[1]));

        int boxes = 0;

        for (int[] box : boxTypes) {

            if (truckSize >= box[0]) {

                boxes += box[0] \* box[1];

                truckSize -= box[0];

            } else {

                boxes += truckSize \* box[1];

                return boxes;

            }

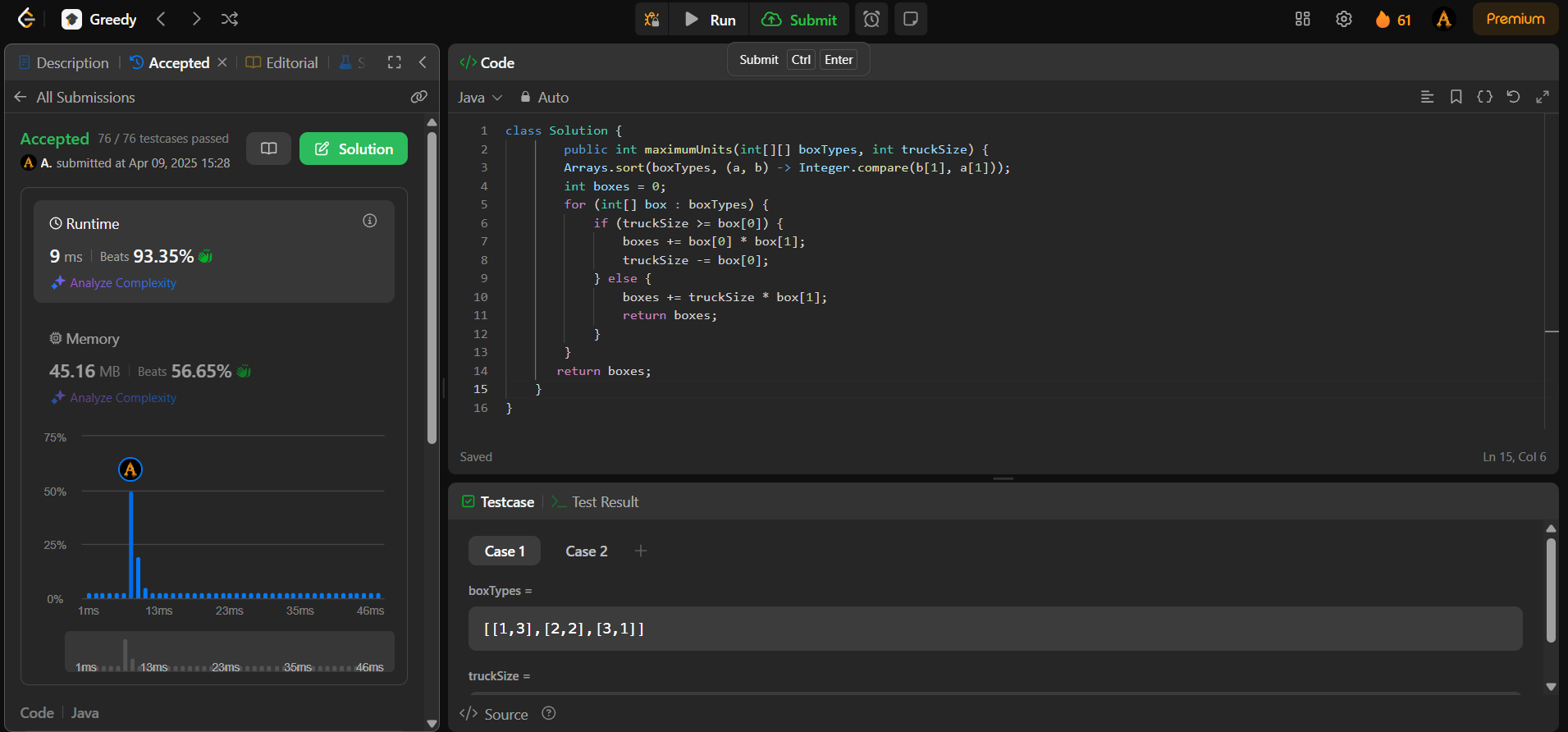
        }

       return boxes;

    }

}

Output :



1. Problem: Min Operations to Make Array Increasing

Code:

class Solution {

    public int minOperations(int[] nums) {

        int count = 0;

        for(int i = 0; i < nums.length - 1; i++) {

            if(nums[i] >= nums[i + 1]) {

                int diff = nums[i] - nums[i + 1] + 1;

                count = count + diff;

                nums[i + 1] = nums[i + 1] + diff;

            }

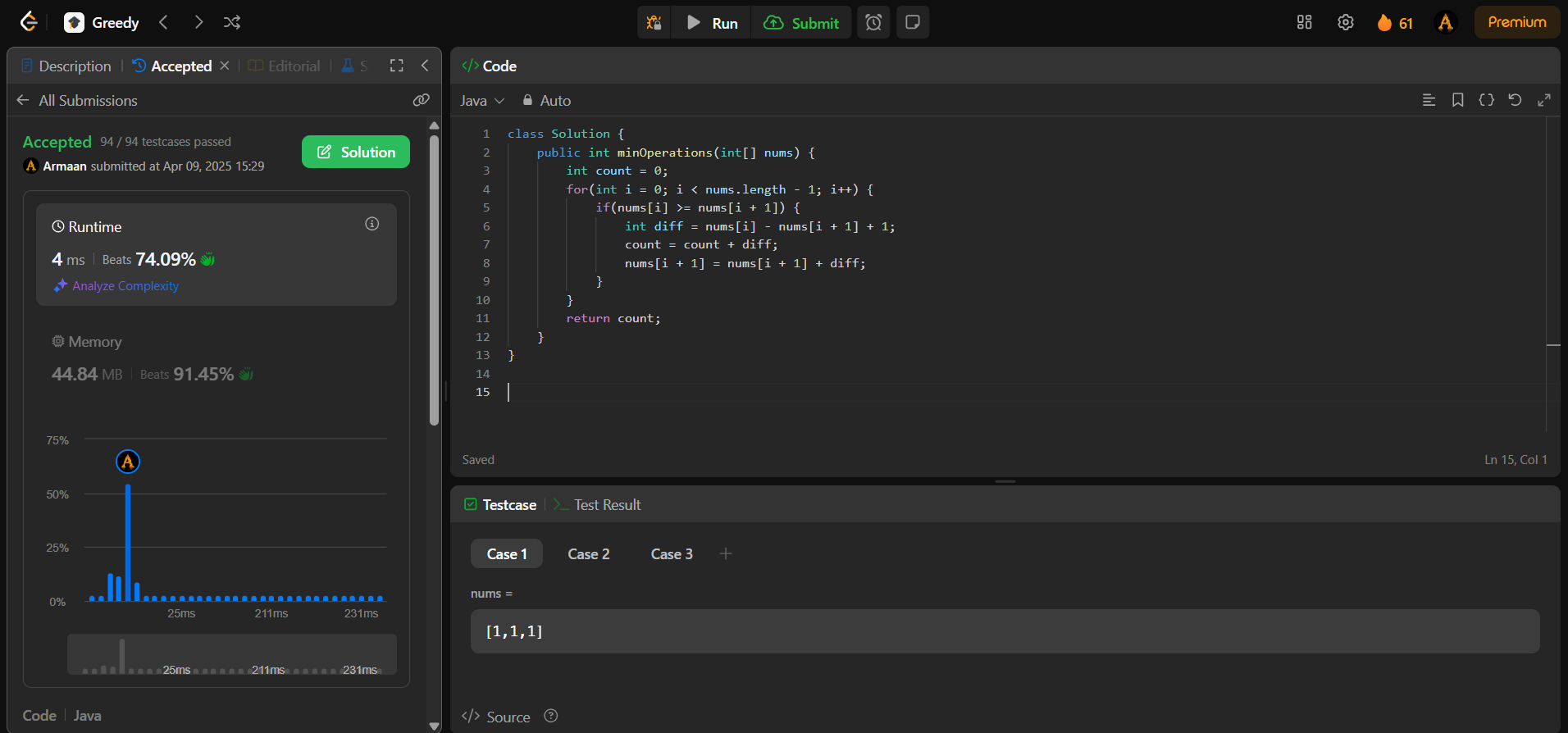
        }

        return count;

    }

}

Output:



1. Problem: Remove Stones to Maximize Total

Code:

class Solution {

public int minStoneSum(int[] piles, int k) {

Queue <Integer> heap = new PriorityQueue (new Comparator <Integer> () {

public int compare (Integer a, Integer b) {

if (a < b)

return 1;

else if (a > b)

return -1;

else

return 0;

}

});

for (int val : piles)

heap.offer (val);

while (k-- > 0) {

int stones = heap.poll ();

stones -= (int) (Math.floor (stones / 2));

heap.offer (stones);

}

int sum = 0;

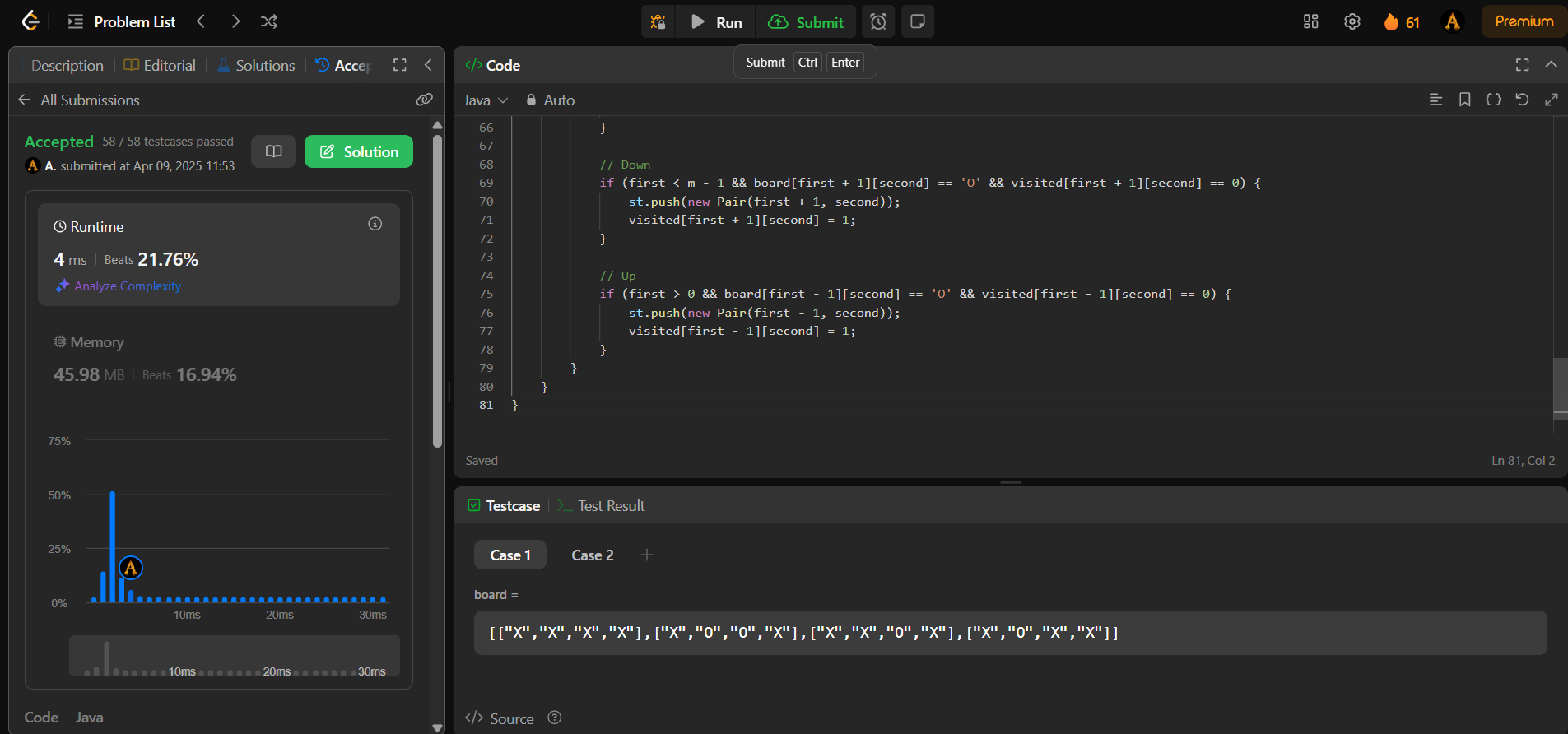
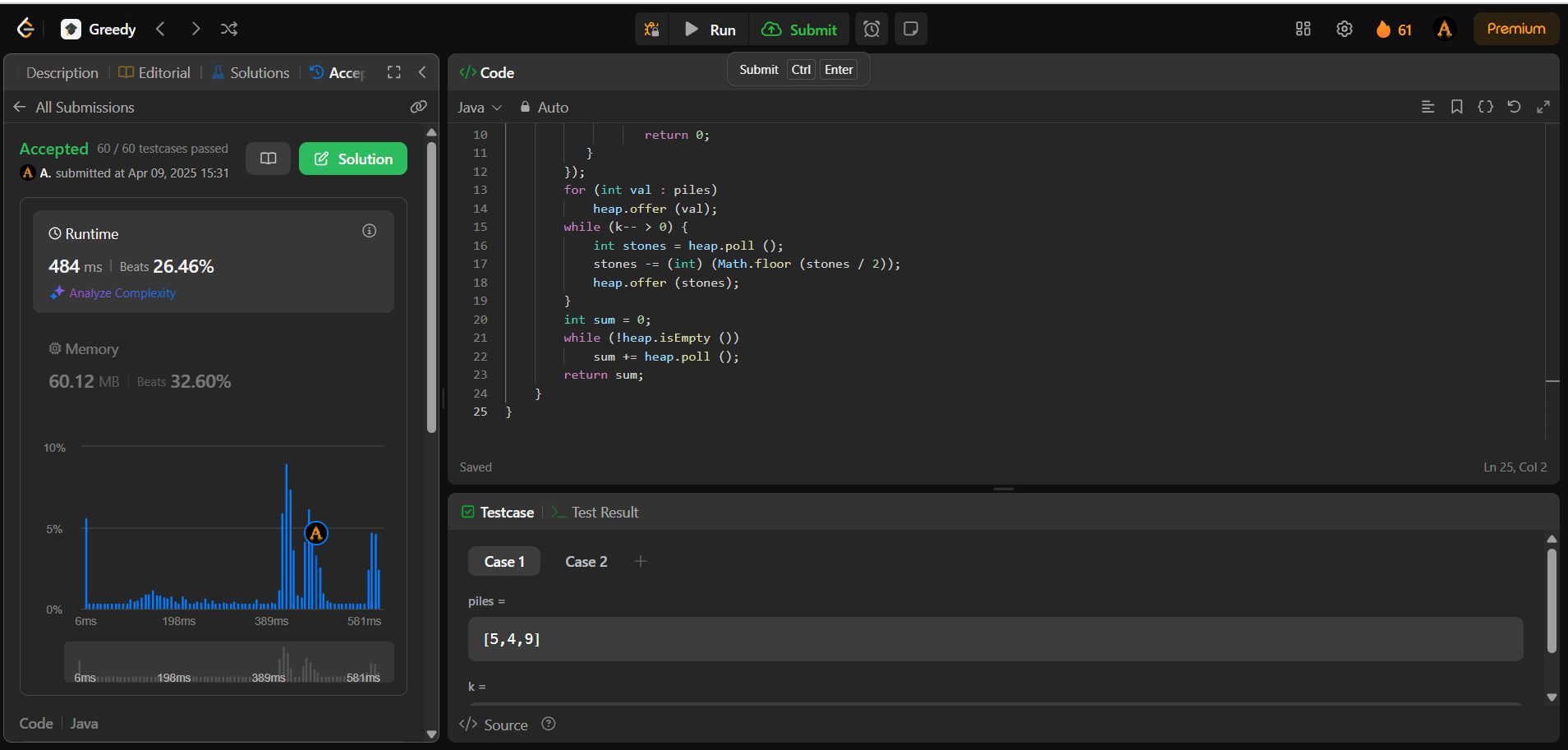
while (!heap.isEmpty ())

sum += heap.poll ();

return sum;

}

}

Output:

1. Problem: Max Score from Removing Substrings

Code:

class Solution {

public int maximumGain(String s, int x, int y) {

int res = 0;

String top, bot;

int top\_score, bot\_score;

if (y > x) {

top = "ba";

top\_score = y;

bot = "ab";

bot\_score = x;

} else {

top = "ab";

top\_score = x;

bot = "ba";

bot\_score = y;

}

// Removing first top substrings cause they give more points

StringBuilder stack = new StringBuilder();

for (char ch : s.toCharArray()) { // Changed 'char' to 'ch'

if (ch == top.charAt(1) && stack.length() > 0 && stack.charAt(stack.length() - 1) == top.charAt(0)) {

res += top\_score;

stack.setLength(stack.length() - 1);

} else {

stack.append(ch);

}

}

// Removing bot substrings cause they give less or equal amount of scores

StringBuilder new\_stack = new StringBuilder();

for (char ch : stack.toString().toCharArray()) { // Changed 'char' to 'ch'

if (ch == bot.charAt(1) && new\_stack.length() > 0 && new\_stack.charAt(new\_stack.length() - 1) == bot.charAt(0)) {

res += bot\_score;

new\_stack.setLength(new\_stack.length() - 1);

} else {

new\_stack.append(ch);

}

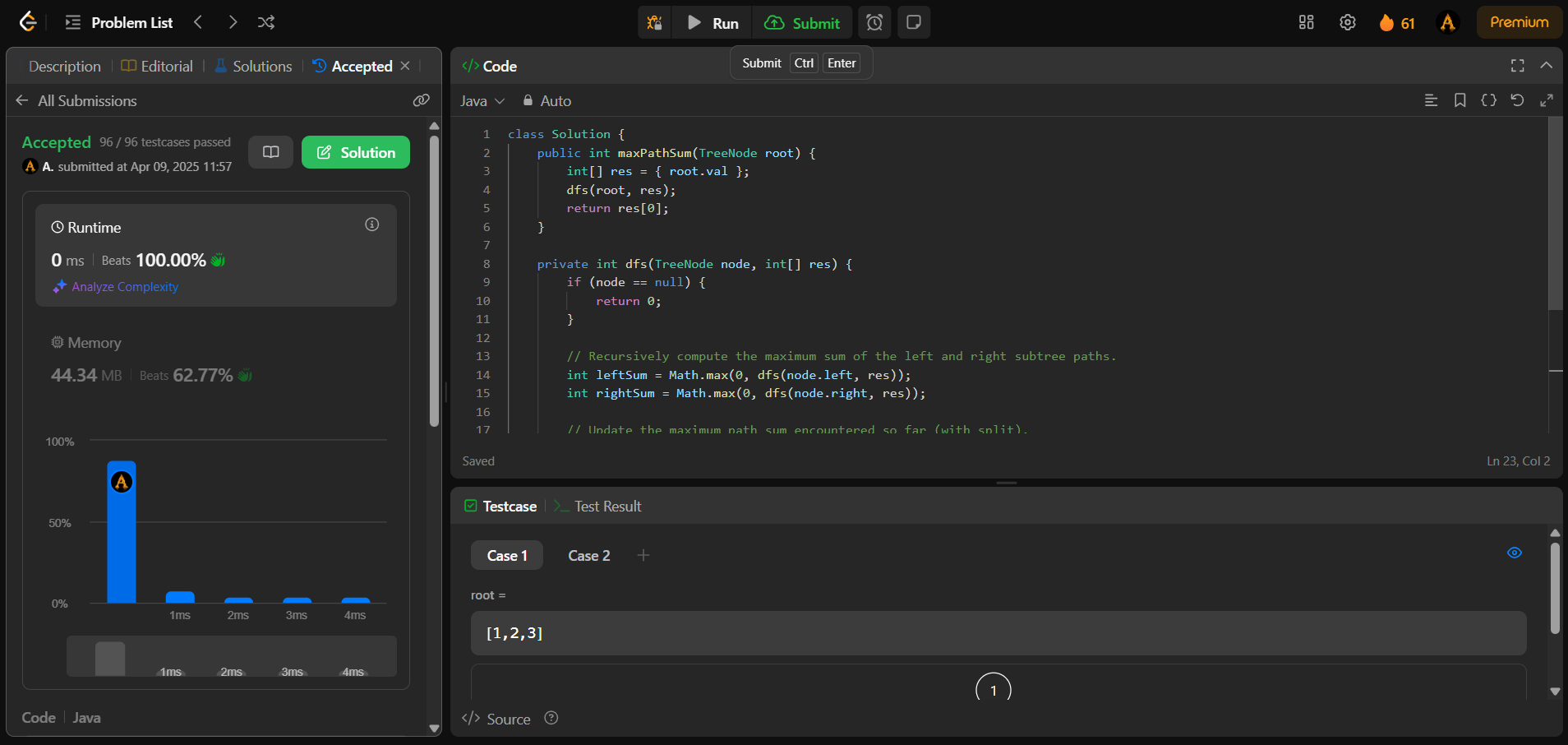
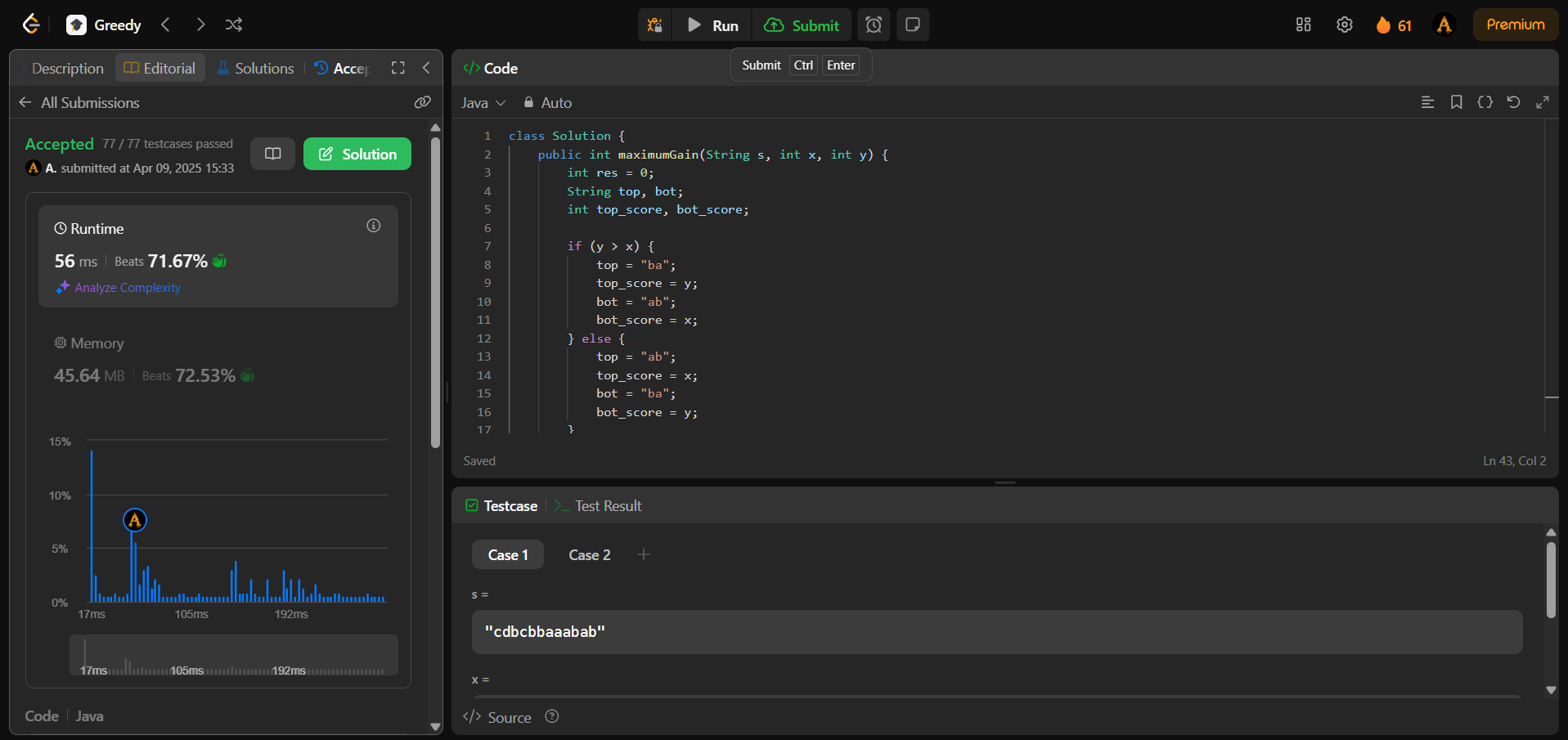
}

return res;

}

}

Output:



1. Problem: Min Operations to Make a Subsequence

Code:

import java.util.\*;

class Solution {

public int minOperations(int[] target, int[] arr) {

Map<Integer, Integer> map = new HashMap<>();

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

map.put(target[i], i);

}

List<Integer> sequence = new ArrayList<>();

for (int num : arr) {

if (map.containsKey(num)) {

sequence.add(map.get(num));

}

}

int maxSubsequence = lis(sequence);

return target.length - maxSubsequence;

}

private int lis(List<Integer> list) {

List<Integer> max = new ArrayList<>();

for (int num : list) {

if (max.isEmpty() || num > max.get(max.size() - 1)) {

max.add(num);

} else {

int pos = Collections.binarySearch(max, num);

if (pos >= 0) {

max.set(pos, num);

} else {

max.set(-(pos + 1), num);

}

}

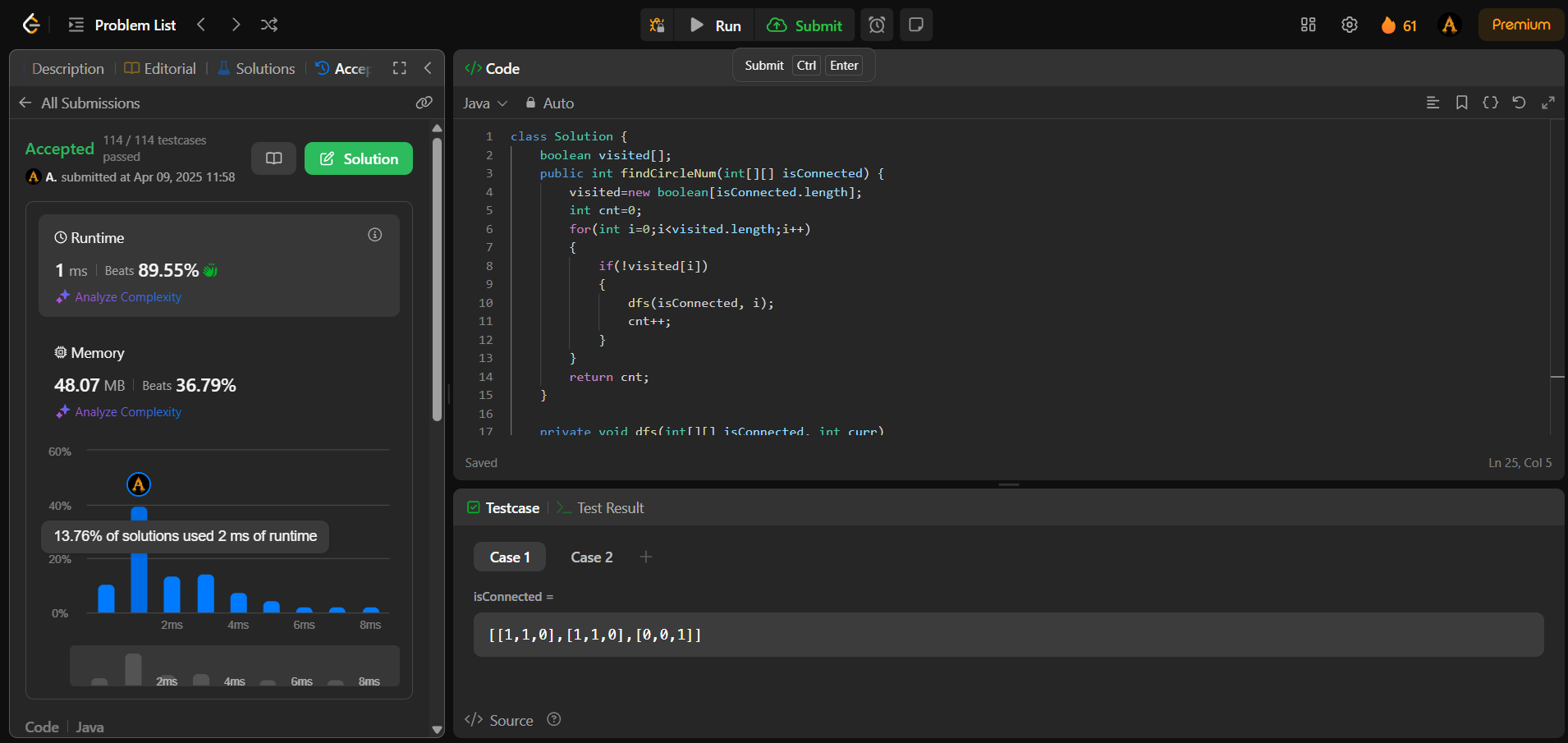
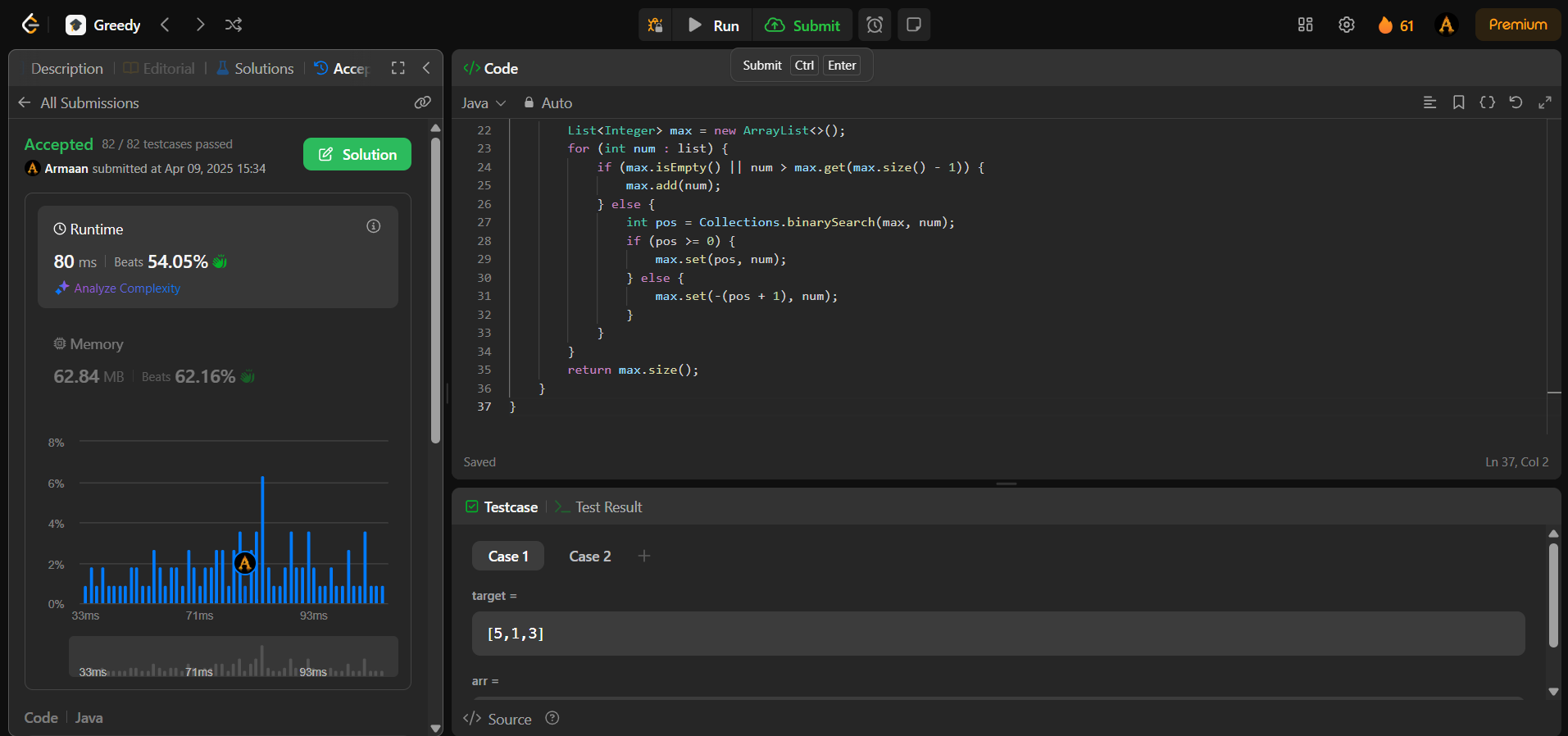
}

return max.size();

}

}

Output:



1. Problem : Max Number of Tasks You Can Assign

Code:

class Solution {

public int maxTaskAssign(int[] tasks, int[] workers, int pills, int strength) {

int left = 0, right = Math.min(tasks.length, workers.length);

Arrays.sort(tasks);

Arrays.sort(workers);

while(left+1<right)

{

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

if(canAssign(mid, tasks, workers, pills, strength))

{

left = mid;

}

else

{

right = mid;

}

}

if(canAssign(right, tasks, workers, pills, strength))

{

return right;

}

else return left;

}

public boolean canAssign(int count, int[] tasks, int[] workers, int pills, int strength){

Deque<Integer> dq = new ArrayDeque<>();

int ind = workers.length - 1;

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

while(ind>=workers.length-count && workers[ind]+strength>=tasks[i])

{

dq.offerLast(workers[ind]);

ind--;

}

if(dq.isEmpty())return false;

if(dq.peekFirst()>=tasks[i])

{

dq.pollFirst();

}

else

{

dq.pollLast();

pills--;

if(pills<0)return false;

}

}

return true;

}

}

Output:

