**ASSIGNMENT 8**

**Max Units on a Truck**

**CODE:**

class Solution {

public:

    int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {

        for (int i=0;i<boxTypes.size();i++){

            reverse(boxTypes[i].begin(),boxTypes[i].end());

        }

        sort(boxTypes.begin(),boxTypes.end());

        reverse(boxTypes.begin(),boxTypes.end());

        int ans=0,i=0;

        while (truckSize>0 && i<boxTypes.size()){

            while (truckSize>0 && boxTypes[i][1]>0){

                ans+=boxTypes[i][0];

                truckSize--;

                boxTypes[i][1]--;

            }

            i++;

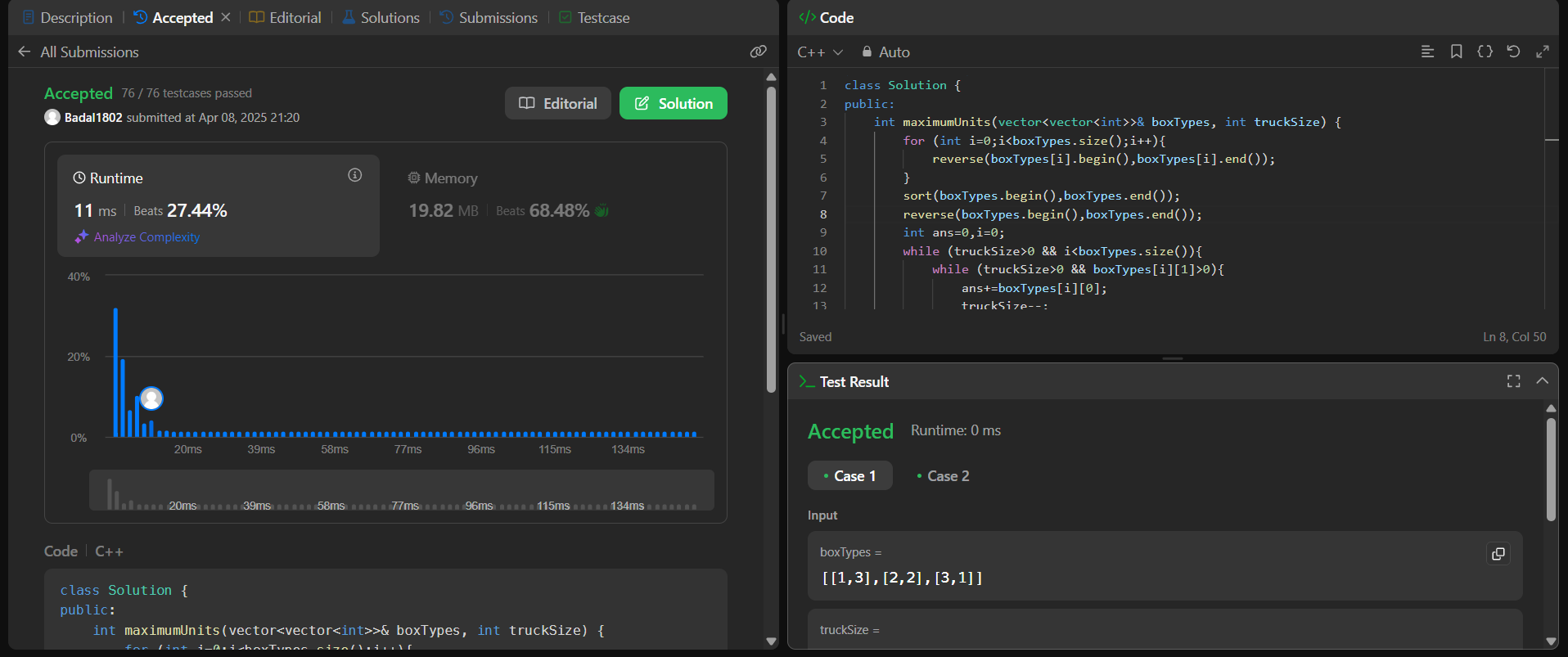
        }

        return ans;

    }

};

**OUTPUT:**

****

**Min Operations to Make Array Increasing**

**CODE:**

class Solution {

public:

    int minOperations(vector<int>& nums) {

         int counter = 0;

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

        {

            while(nums[i] >= nums[i+1])

            {

                nums[i+1]++;

                counter++;

            }

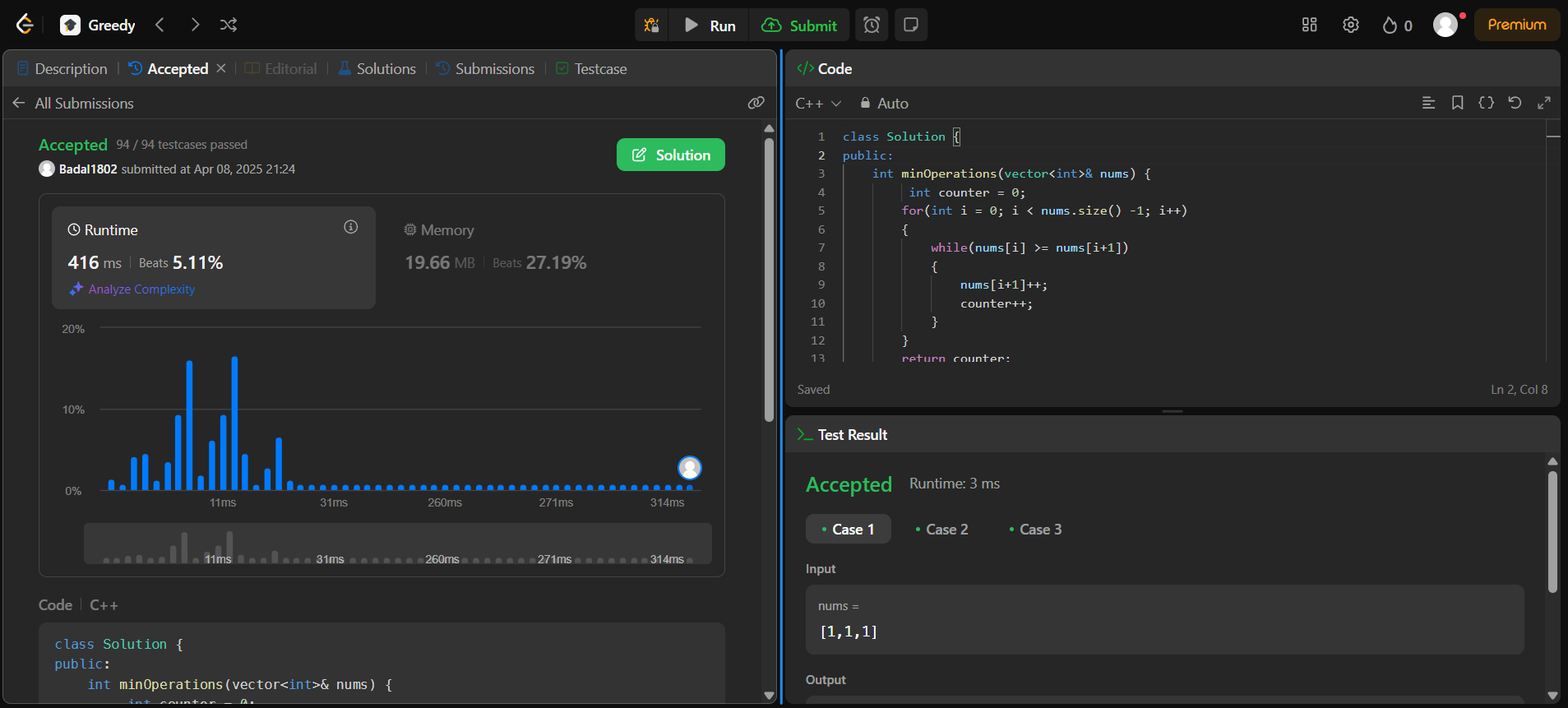
        }

        return counter;

    }

};

**OUTPUT:**

****

**Remove Stones to Maximize Total**

**CODE:**

class Solution {

public:

    int minStoneSum(vector<int>& piles, int k) {

      int n=piles.size();

        priority\_queue<int,vector<int>>pq(piles.begin(),piles.end());

        int ans=accumulate(piles.begin(),piles.end(),0);

        int i=0;

        while(k>0 && !pq.empty())

        {

            int temp=pq.top();

            pq.pop();

            ans-=(temp/2);

            pq.push(temp-temp/2);

            k--;

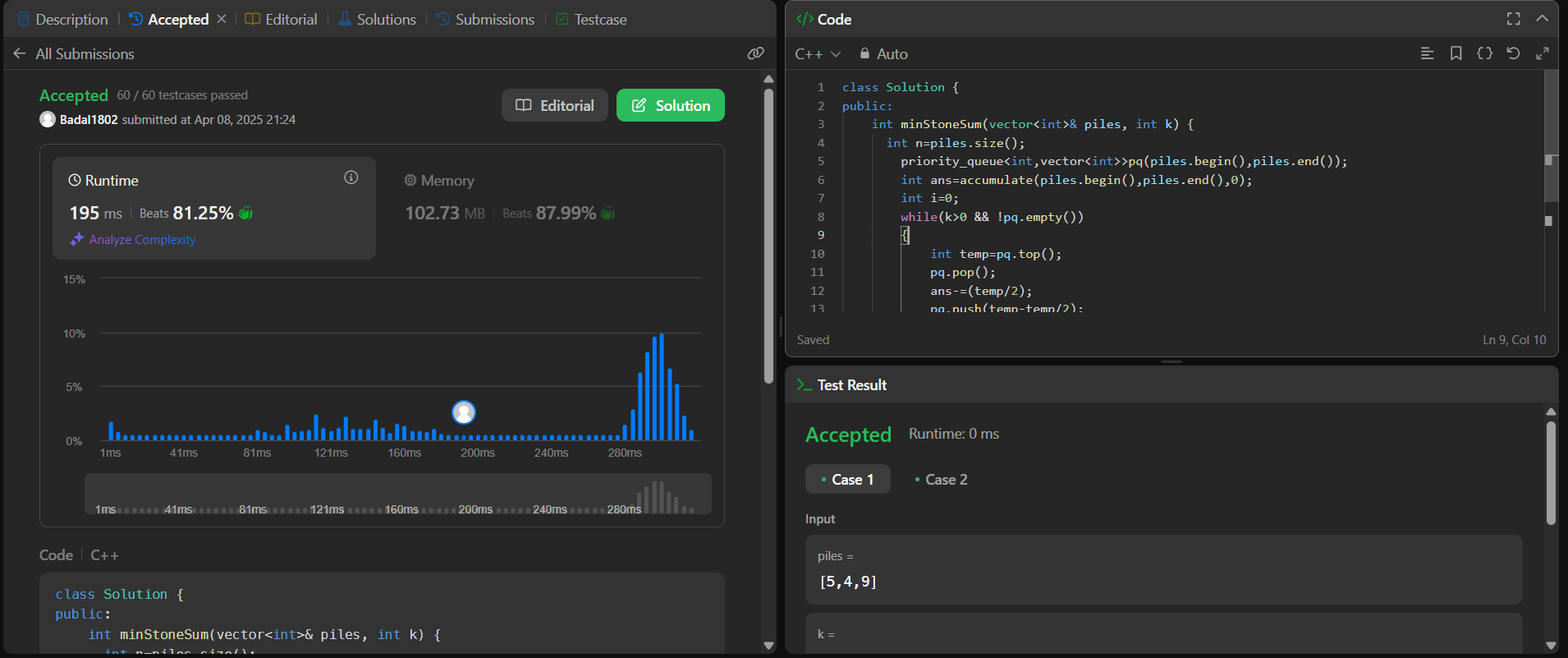
        }

        return ans;

    }

};

**OUTPUT:**

****

**Max Score from Removing Substrings**

**CODE:**

class Solution {

void getCount(string str, string sub, int& cnt1, int& cnt2) {

char first = sub[0], second = sub[1];

int i = 1;

while(i < str.length()) {

if(i > 0 && str[i-1] == first && str[i] == second) {

cnt1++;

str.erase(i-1, 2);

i--;

continue;

}

i++;

}

i = 1;

while(i < str.length()) {

if(i > 0 && str[i-1] == second && str[i] == first) {

cnt2++;

str.erase(i-1, 2);

i--;

continue;

}

i++;

}

return;

}

public:

int maximumGain(string s, int x, int y) {

int mxABcnt = 0;

int mxBAcnt = 0;

int minBAcnt = 0;

int minABcnt= 0;

getCount(s, "ab", mxABcnt, minBAcnt);

getCount(s, "ba", mxBAcnt, minABcnt);

int operation1 = mxABcnt \* x + minBAcnt \* y;

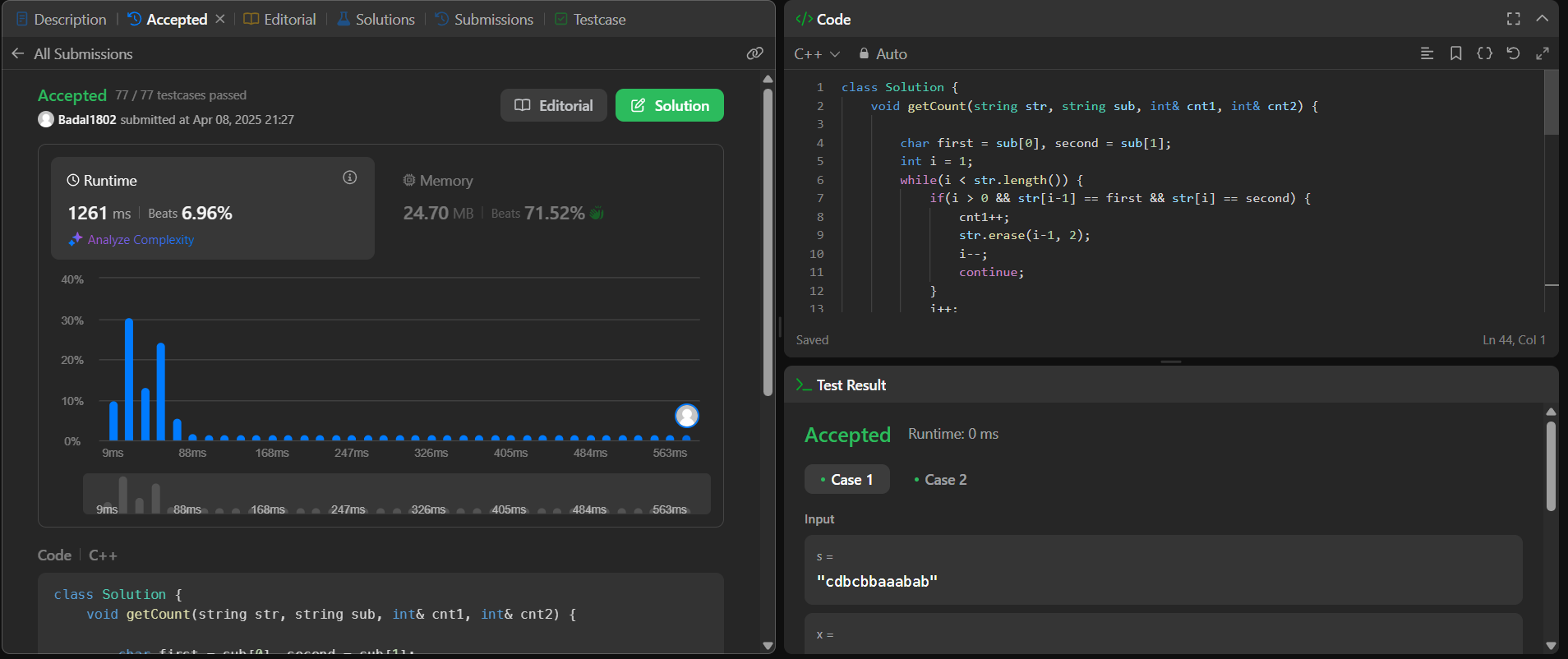
int operation2 = mxBAcnt \* y + minABcnt \* x;

return max(operation1, operation2);

}

};

**OUTPUT:**

****

**Min Operations to Make a Subsequence  
CODE:**

class Solution {

public:

int minOperations(vector<int>& target, vector<int>& arr) {

unordered\_map<int, int> mp;

for(int i = 0; i < target.size(); i++) mp[target[i]] = i;

vector<int> v;

for(int a: arr) if (mp.count(a)) v.push\_back(mp[a]);

int n = v.size(), ans = 0;

vector<int> tail(n + 1, INT\_MAX);

tail[0] = INT\_MIN;

for(int a: v) {

int b = upper\_bound(tail.begin(), tail.begin() + min(ans + 1, n), a) - tail.begin();

if (b == 0 || (tail[b - 1] < a && tail[b] > a)) {

tail[b] = a;

ans = max(ans, b);

}

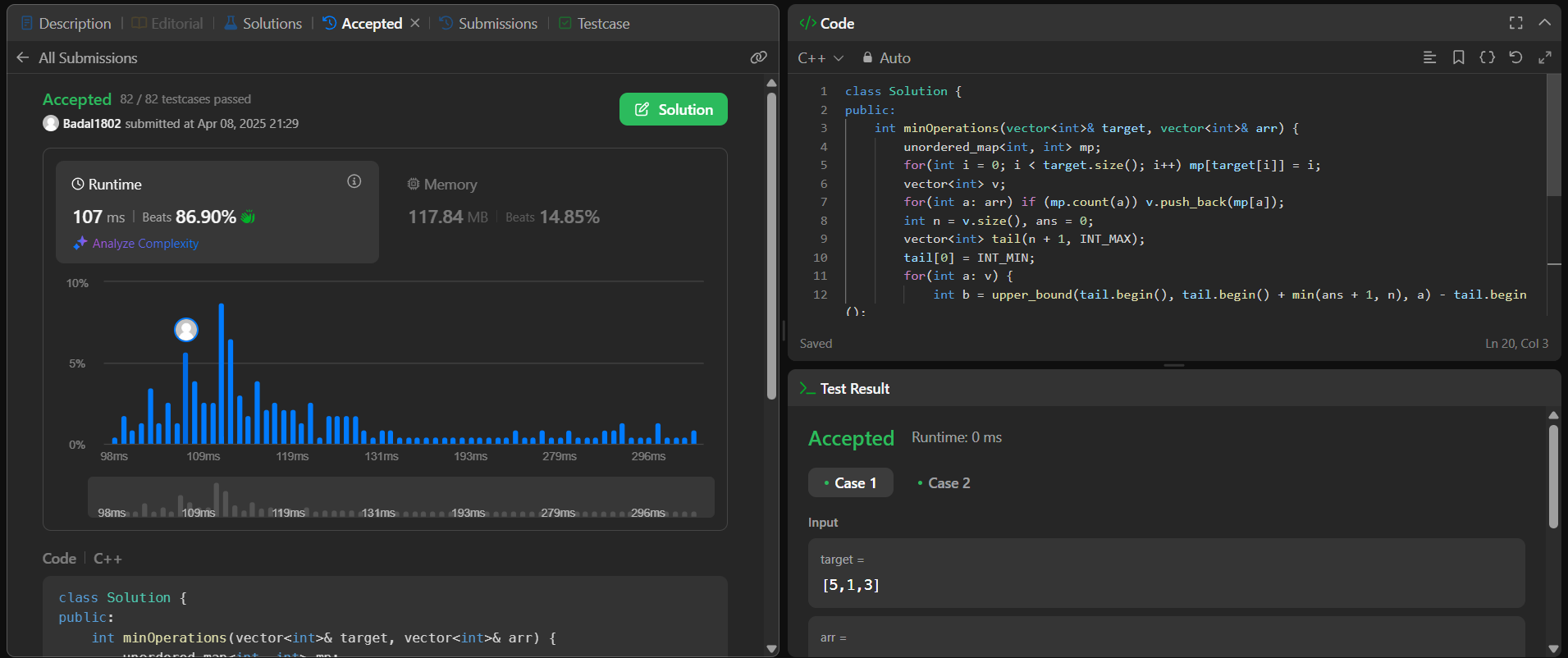
}

return target.size() - ans;

}

};

**OUTPUT:**

****

**Max Number of Tasks You Can Assign  
CODE:**

class Solution {

private:

bool isPossible(int numTask, vector<int>& tasks, vector<int>& workers, int pills, int strength){

multiset<int> ms(workers.end()-numTask,workers.end());

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

auto it = ms.end();

it--;

if(\*it < tasks[i]){

if(!pills) return false;

it = ms.lower\_bound(tasks[i]-strength);

if(it == ms.end()) return false;

pills--;

}

ms.erase(it);

}

return true;

}

public:

int maxTaskAssign(vector<int>& tasks, vector<int>& workers, int pills, int strength) {

sort(tasks.begin(), tasks.end());

sort(workers.begin(),workers.end());

int n = tasks.size();

int m = workers.size();

int low = 0;

int high = min(n,m);

int ans = 0;

while(low <= high){

int mid = (low + high) >> 1;

if(isPossible(mid,tasks,workers,pills,strength)){

ans = mid;

low = mid + 1;

}

else high = mid - 1;

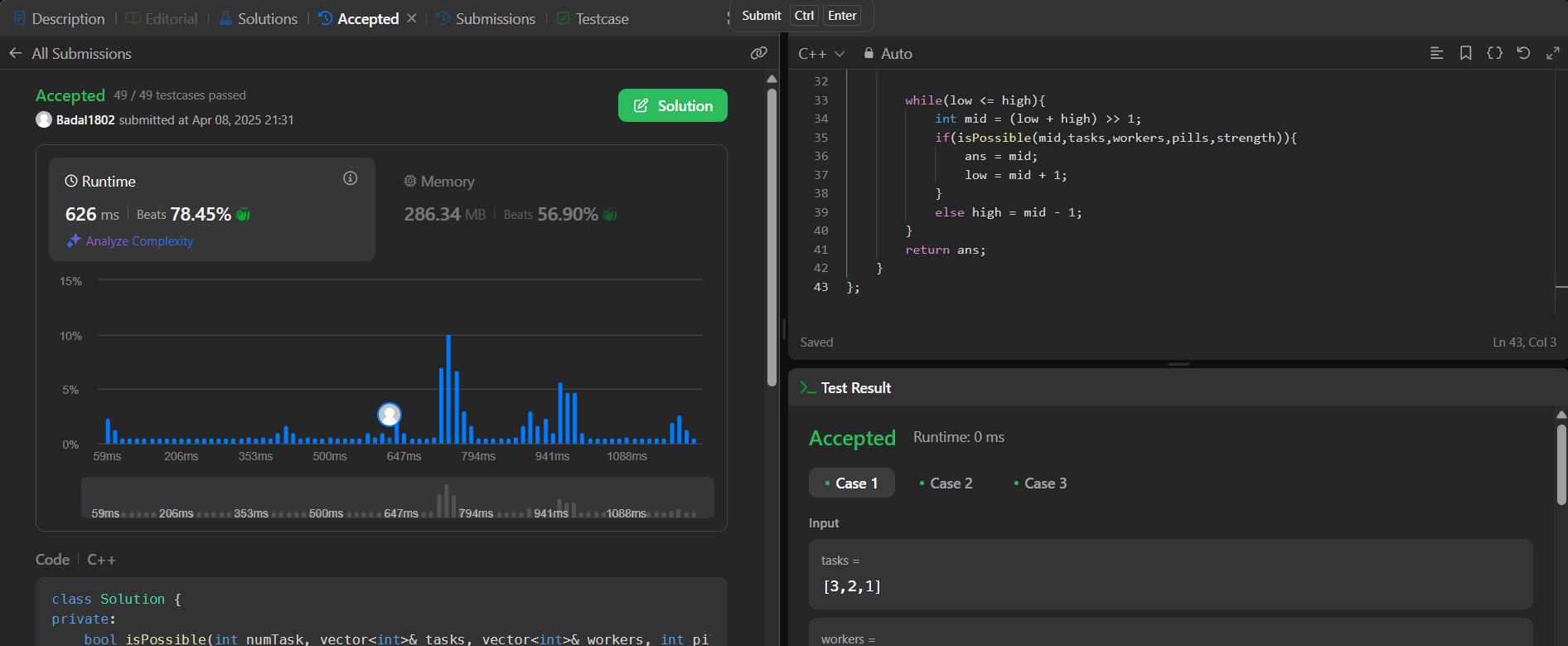
}

return ans;

}

};

**OUTPUT:**

****