**ADVANCED PROGRAMMING-II**

**ASSIGNMENT-08**

**UID-22BCS11558 NAME-NIKHIL**

**Q1.** **[Maximum Units on a Truck](https://leetcode.com/problems/maximum-units-on-a-truck/):**

**Code:**

class Solution {

public:

 static bool compare(vector<int> &a ,vector<int> &b){

        return a[1]>b[1];

    }

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

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

       int n=boxTypes.size();

       int unit=0;

       for(int i=0;i<n;i++){

        int box=boxTypes[i][0];

        int profit=boxTypes[i][1];

         if (truckSize >= box) {

                unit += box \* profit;

                truckSize -= box;

            } else {

                unit += truckSize \* profit;

                break;

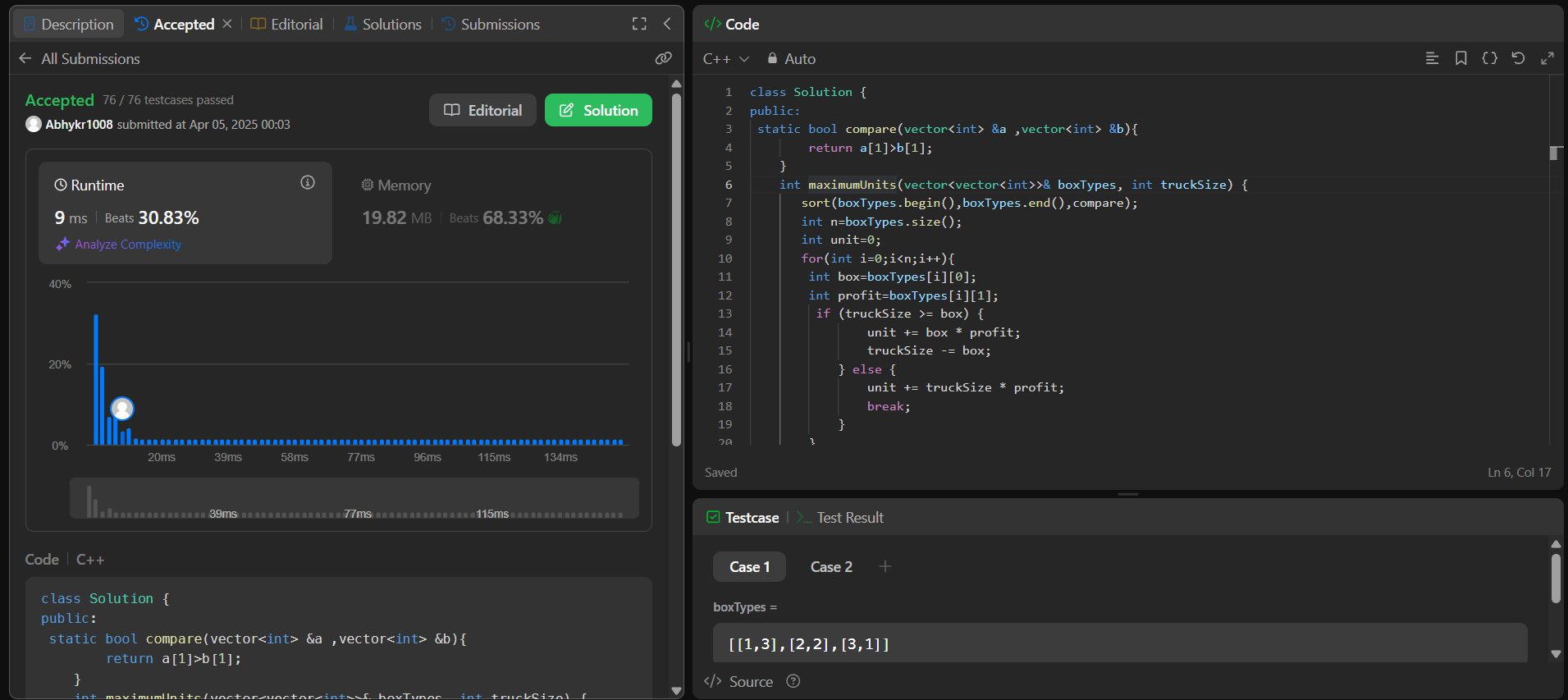
            }

        }

        return unit;

    }

};

**Screenshot:**

**Q2. Min Operations to Make Array Increasing:**

**Code:**

class Solution {

public:

    int minOperations(vector<int>& nums) {

        int n=nums.size();

        int operations=0;

        int current\_max=0;

        for(int i=0;i<n;i++){

            if(nums[i]<=current\_max){

                operations=operations+(current\_max+1-nums[i]);

                    nums[i]=current\_max +1;

                }

                current\_max=nums[i];

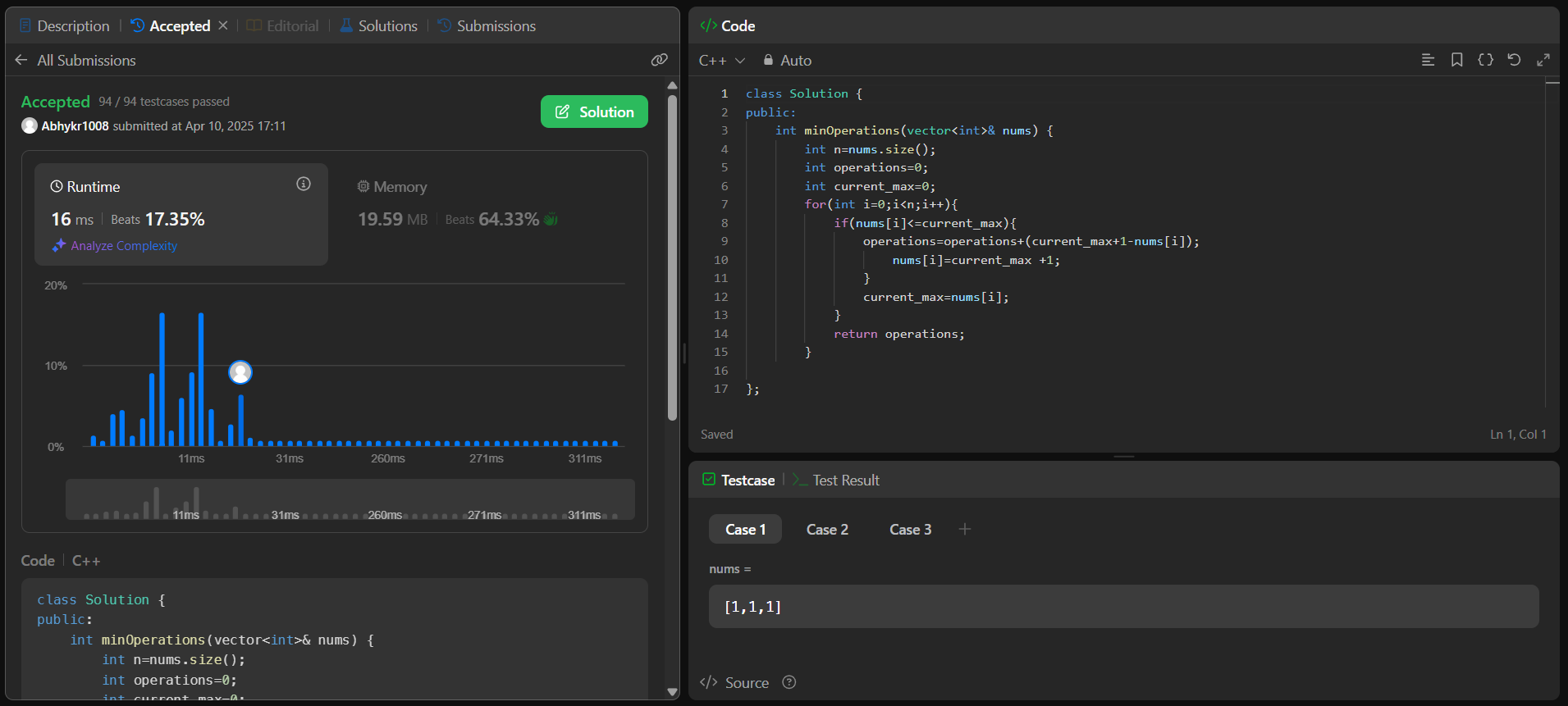
            }

            return operations;

        }

};

**Screenshot:**

****

**Q3. Remove Stones to Maximize Total:**

**Code:**

class Solution {

public:

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

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

    while (k--) {

        int maxPile = pq.top();

        pq.pop();

        maxPile -= maxPile / 2;

        pq.push(maxPile);

    }

    int total = 0;

    while (!pq.empty()) {

        total += pq.top();

        pq.pop();

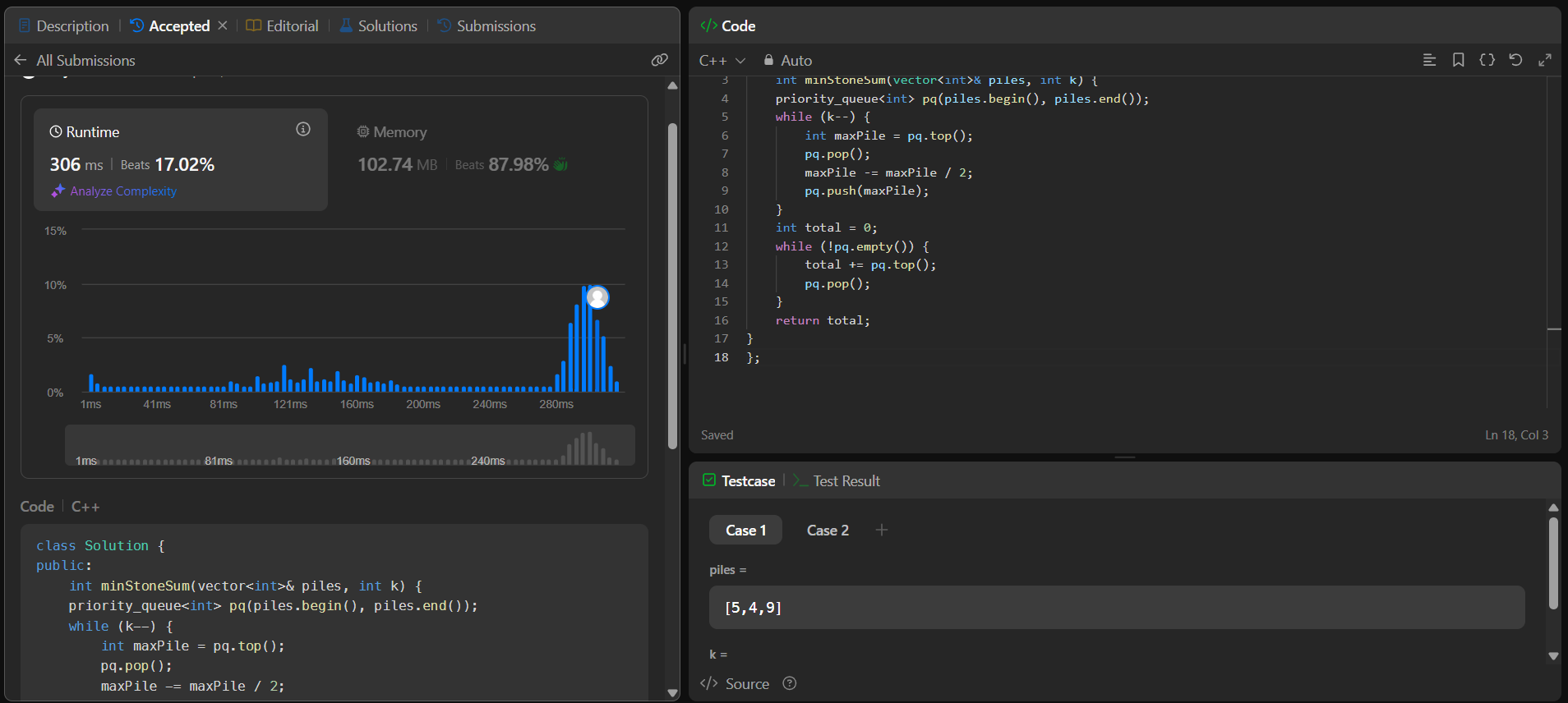
    }

    return total;

}

};

**Screenshot:**



**Q4. Max Score from Removing Substrings:**

**Code:**

class Solution {

public:

    int removePairs(string& s, char first, char second, int points) {

    stack<char> st;

    int score = 0;

    for (char c : s) {

        if (!st.empty() && st.top() == first && c == second) {

            st.pop();

            score += points;

        } else {

            st.push(c);

        }

    }

    string rebuilt;

    while (!st.empty()) {

        rebuilt += st.top();

        st.pop();

    }

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

    s = rebuilt;

    return score;

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

    int total = 0;

    if (x > y) {

        total += removePairs(s, 'a', 'b', x);

        total += removePairs(s, 'b', 'a', y);

    } else {

        total += removePairs(s, 'b', 'a', y);

        total += removePairs(s, 'a', 'b', x);

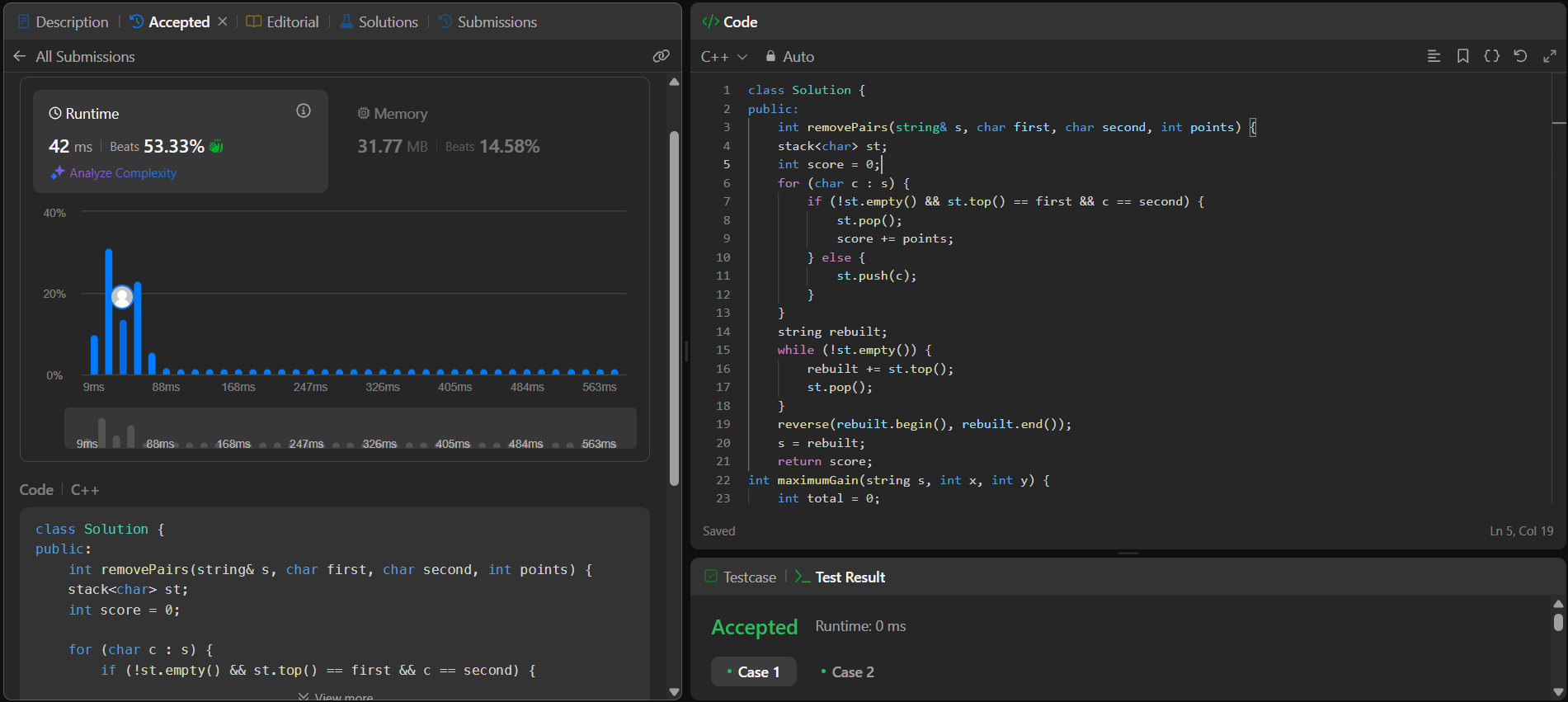
    }

    return total;

}

};

**Screenshot:**



**Q5. Min Operations to Make a Subsequence:**

**Code:**

class Solution {

public:

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

    unordered\_map<int, int> pos;

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

        pos[target[i]] = i;

    }

    vector<int> seq;

    for (int val : arr) {

        if (pos.find(val) != pos.end()) {

            seq.push\_back(pos[val]);

        }

    }

    vector<int> lis;

    for (int index : seq) {

        auto it = lower\_bound(lis.begin(), lis.end(), index);

        if (it == lis.end()) {

            lis.push\_back(index);

        } else {

            \*it = index;

        }

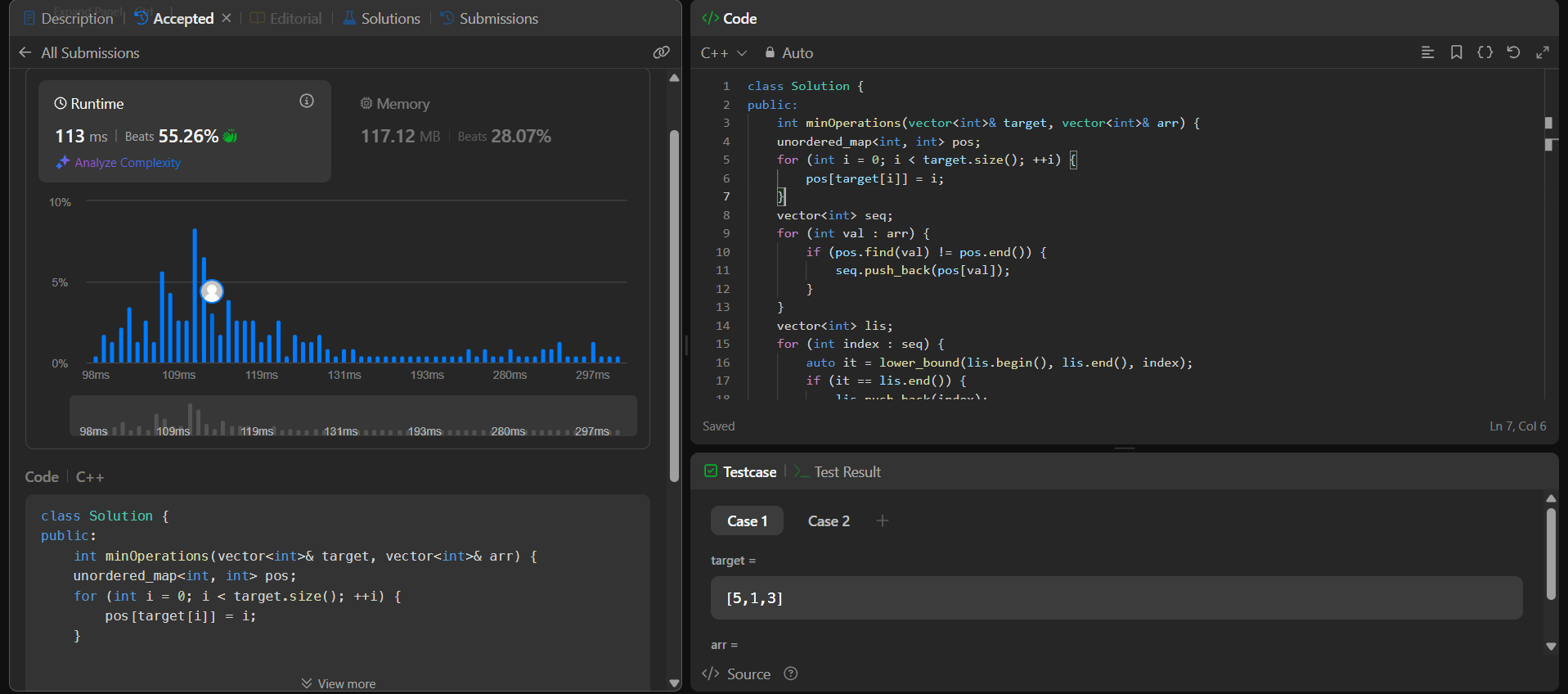
    }

    return target.size() - lis.size();

}

};

**Screenshot:**



**Q6. Max Number of Tasks You Can Assign:**

**Code:**

class Solution {

public:

    bool canAssign(int k, vector<int>& tasks, vector<int>& workers, int pills, int strength) {

        multiset<int> availableWorkers(workers.end() - k, workers.end());

        int usedPills = 0;

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

            auto it = availableWorkers.lower\_bound(tasks[i]);

            if (it != availableWorkers.end()) {

                availableWorkers.erase(it);

            } else {

                if (usedPills == pills) return false;

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

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

                availableWorkers.erase(it);

                ++usedPills;

            }

        }

        return true

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

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

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

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

        int result = 0;

        while (left <= right) {

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

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

                result = mid;

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return result;

    }s

};

**Screenshot:**

