Name: Anish Patial

Uid: 22BCS15029

Section: FL\_Iot 601 ‘A’

# Max Units on a Truck

class Solution {

public:

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

sort(boxTypes.begin(), boxTypes.end(), [](const vector<int>& a, const vector<int>& b) {

return a[1] > b[1];

});

int total = 0;

for (auto& box : boxTypes) {

int count = min(box[0], truckSize);

total += count \* box[1];

truckSize -= count;

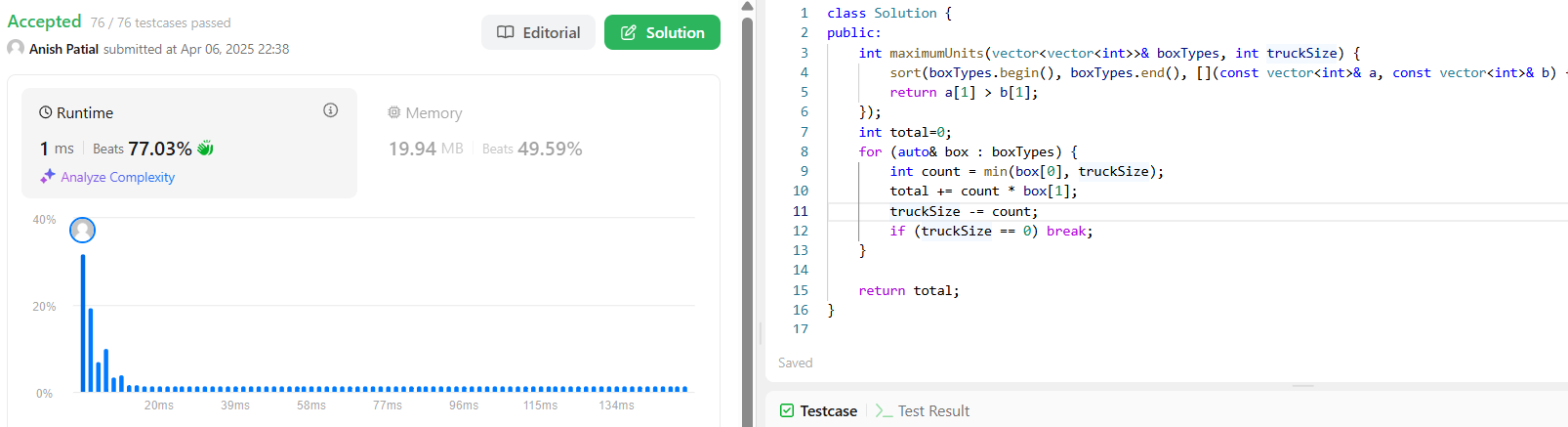
if (truckSize == 0) break;

}

return total;

}

};



# [Minimum Operations to Make the Array Increasing](https://leetcode.com/problems/minimum-operations-to-make-the-array-increasing/)

class Solution {

public:

    int minOperations(vector<int>& nums) {

        int count=0;

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

            if(nums[i+1]<=nums[i]){

                count+=(nums[i]-nums[i+1]+1);

                nums[i+1]=nums[i]+1;

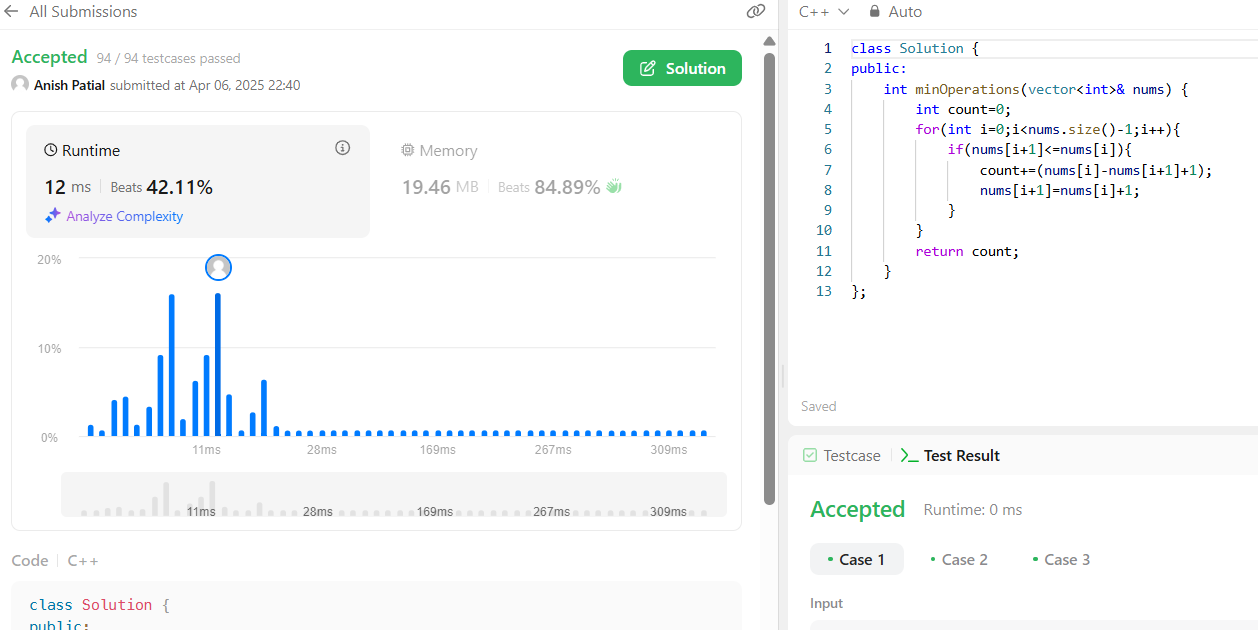
            }

        }

        return count;

    }

};



# [Remove Stones to Minimize the Total](https://leetcode.com/problems/remove-stones-to-minimize-the-total/)

class Solution { public:

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

int ans = accumulate(piles.begin(), piles.end(), 0); priority\_queue<int> maxHeap;

for (const int pile : piles) maxHeap.push(pile);

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

const int maxPile = maxHeap.top(); maxHeap.pop();

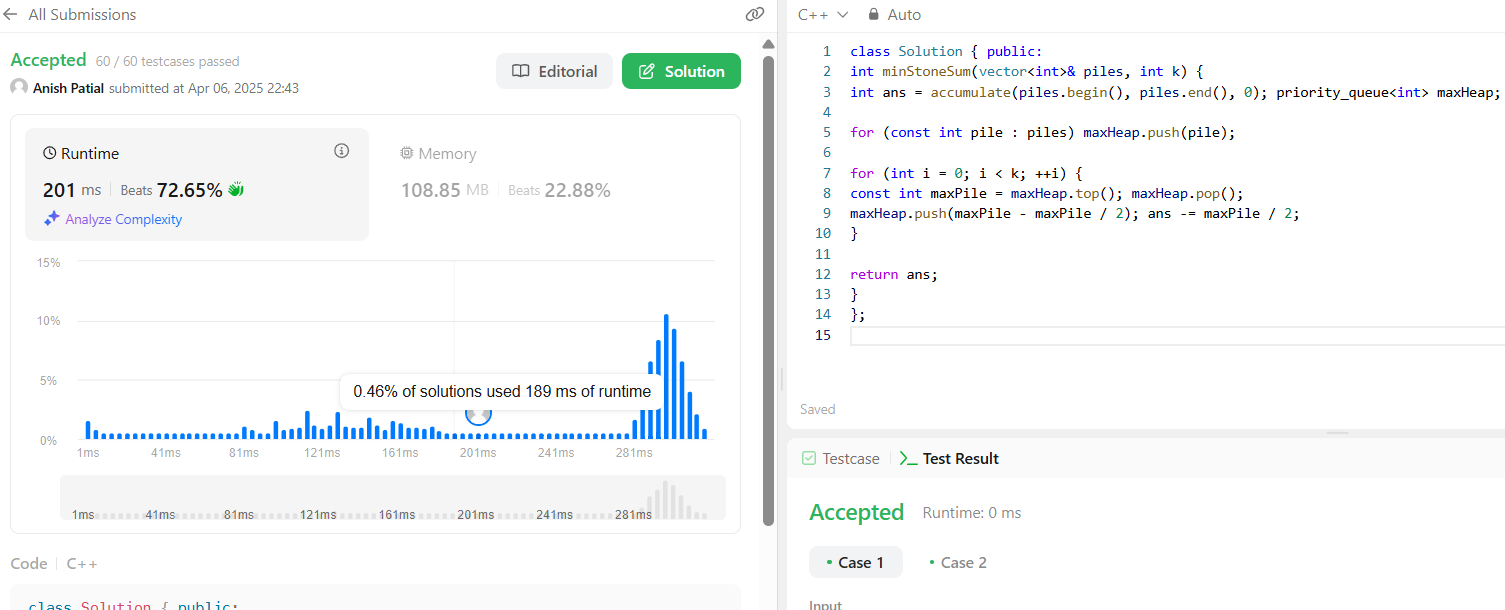
maxHeap.push(maxPile - maxPile / 2); ans -= maxPile / 2;

}

return ans;

}

};



# [Maximum Score From Removing Substrings](https://leetcode.com/problems/maximum-score-from-removing-substrings/)

class Solution { public:

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

return x > y ? gain(s, "ab", x, "ba", y) : gain(s, "ba", y, "ab", x);

}

private:

int gain(const string& s, const string& sub1, int point1, const string& sub2, int point2) {

int points = 0;

vector<char> stack1; vector<char> stack2;

for (const char c : s)

if (!stack1.empty() && stack1.back() == sub1[0] && c == sub1[1]) { stack1.pop\_back();

points += point1;

} else {

stack1.push\_back(c);

}

for (const char c : stack1)

if (!stack2.empty() && stack2.back() == sub2[0] && c == sub2[1]) { stack2.pop\_back();

points += point2;

} else {

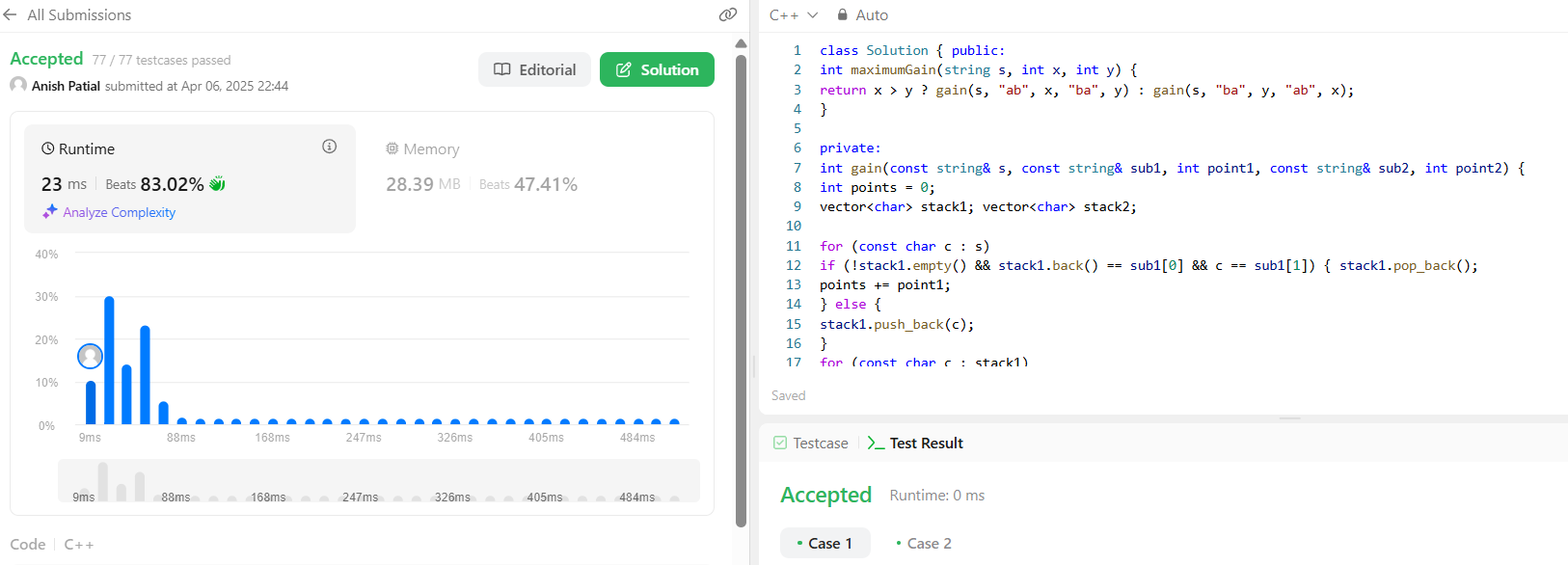
stack2.push\_back(c);

}

return points;

}

};



# [Minimum Operations to Make a Subsequence](https://leetcode.com/problems/minimum-operations-to-make-a-subsequence/)

class Solution { public:

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

unordered\_map<int, int> numToIndex;

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

for (const int a : arr)

if (const auto it = numToIndex.find(a); it != numToIndex.end()) indices.push\_back(it->second);

return target.size() - lengthOfLIS(indices);

}

private:

int lengthOfLIS(vector<int>& nums) { vector<int> tails;

for (const int num : nums)

if (tails.empty() || num > tails.back()) tails.push\_back(num);

else

tails[firstGreaterEqual(tails, num)] = num; return tails.size();

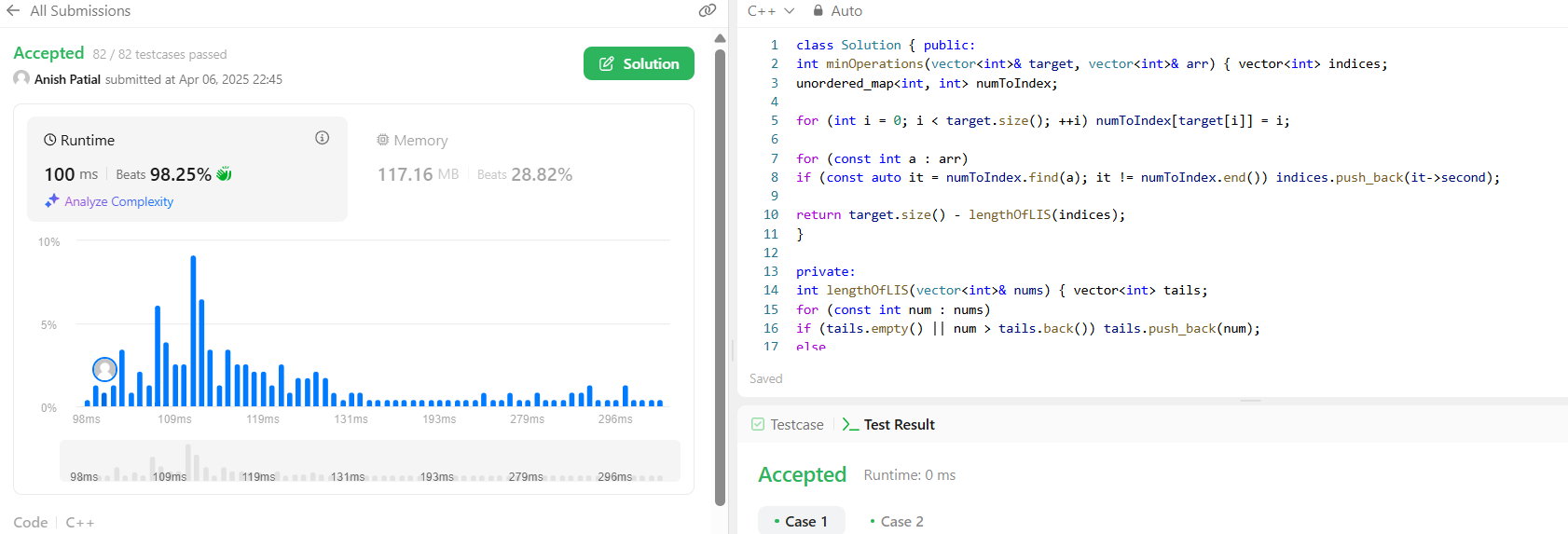
}

private:

int firstGreaterEqual(const vector<int>& arr, int target) { return ranges::lower\_bound(arr, target) - arr.begin();

}

};



# [Maximum Number of Tasks You Can Assign](https://leetcode.com/problems/maximum-number-of-tasks-you-can-assign/)

class Solution { public:

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

int ans = 0; int l = 0;

int r = min(tasks.size(), workers.size()); ranges::sort(tasks);

ranges::sort(workers);

auto canComplete = [&](int k, int pillsLeft) { map<int, int> sortedWorkers;

for (int i = workers.size() - k; i < workers.size(); ++i)

++sortedWorkers[workers[i]]; for (int i = k - 1; i >= 0; --i) {

auto it = sortedWorkers.lower\_bound(tasks[i]); if (it != sortedWorkers.end()) {

if (--(it->second) == 0)

sortedWorkers.erase(it);

} else if (pillsLeft > 0) {

it = sortedWorkers.lower\_bound(tasks[i] - strength); if (it != sortedWorkers.end()) {

if (--(it->second) == 0)

sortedWorkers.erase(it);

--pillsLeft;

} else {

return false;

}

} else {

return false;

}

}

return true;

};

while (l <= r) {

const int m = (l + r) / 2;

if (canComplete(m, pills)) { ans = m;

l = m + 1;

} else {

r = m - 1;

}

}

return ans;

}

};

