**Name:** Aditya Singh

**UID:** 22BCS15063

**Section:** FL\_IOT\_601 - A

**Experiment – 8 Solutions:-**

1. **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 ans = 0;

for(auto box : boxTypes){

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

ans += val \* box[1];

truckSize -= val;

if(truckSize == 0) break;

}

return ans;

}

};

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **Minimum Operations to Make Array Increasing:-**

class Solution {

public:

int minOperations(vector<int>& nums) {

if(nums.size() == 1) return 0;

int max\_val = nums[0], ops = 0;

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

if(nums[i] > max\_val){

max\_val = nums[i];

continue;

}

ops += max\_val - nums[i] + 1;

max\_val++;

}

return ops;

}

};

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **Remove Stones to Maximize Total:-**

class Solution {

public:

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

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

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

while(k--){

int a = pq.top();

pq.pop();

pq.push(a - (a / 2));

res -= (a / 2);

}

return res;

}

};

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **Maximum Score from Removing Substrings:-**

class Solution {

public:

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

int aCount = 0;

int bCount = 0;

int lesser = min(x, y);

int result = 0;

for (char c : s) {

if (c > 'b') {

result += min(aCount, bCount) \* lesser;

aCount = 0;

bCount = 0;

}

else if (c == 'a') {

if (x < y && bCount > 0) {

bCount--;

result += y;

} else {

aCount++;

}

}

else {

if (x > y && aCount > 0) {

aCount--;

result += x;

}

else {

bCount++;

}

}

}

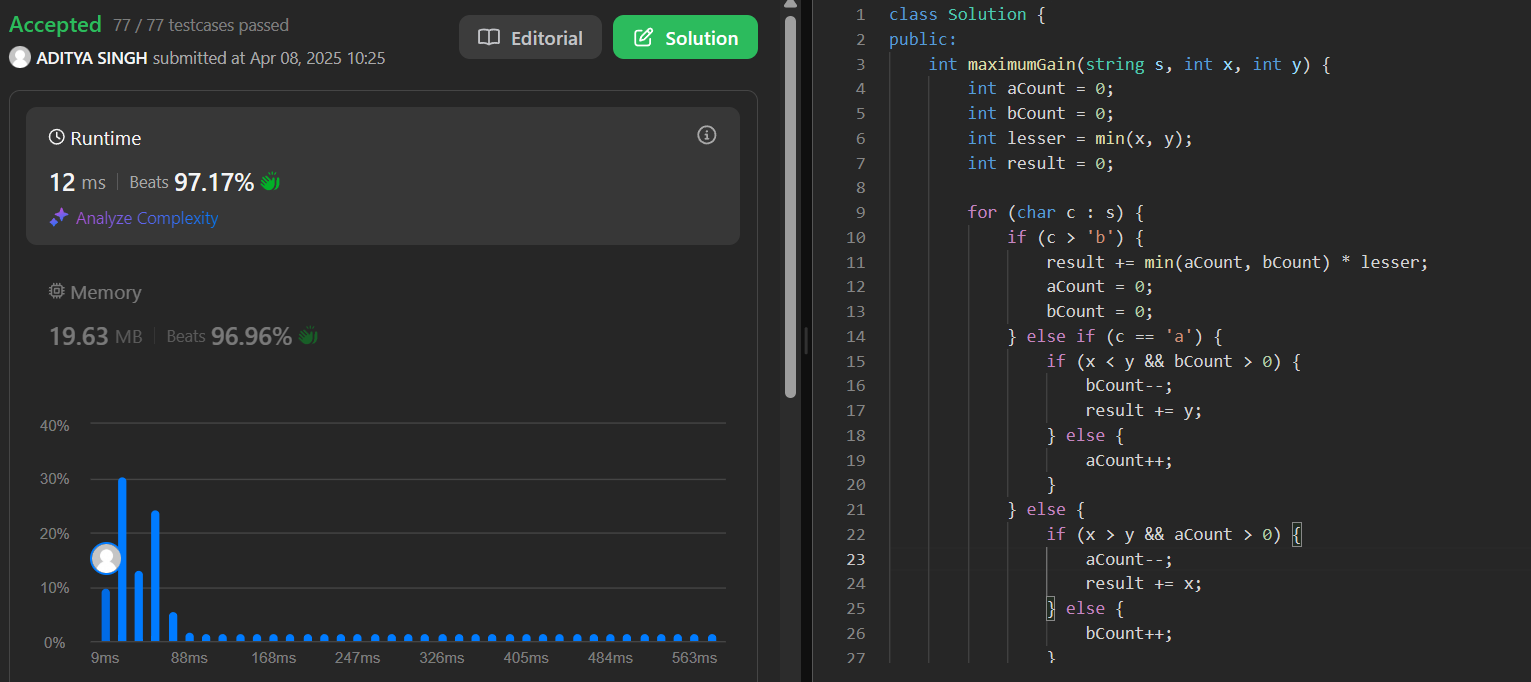
result += min(aCount, bCount) \* lesser;

return result;

}

};

Result:-

****

1. **Minimum Operations to Make a Subsequence:-**

class Solution {

public:

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

unordered\_map<int, int> mapping;

int i = 0;

for (auto& num : target) mapping[num] = ++i;

vector<int> A;

for (int& num : arr){

if (mapping.find(num) != mapping.end()){

A.push\_back(mapping[num]);

}

}

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

}

int lengthOfLIS(vector<int>& nums) {

if (nums.empty()) return 0;

vector<int> piles;

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

auto it = std::lower\_bound(piles.begin(), piles.end(), nums[i]);

if (it == piles.end()) piles.push\_back(nums[i]);

else \*it = nums[i];

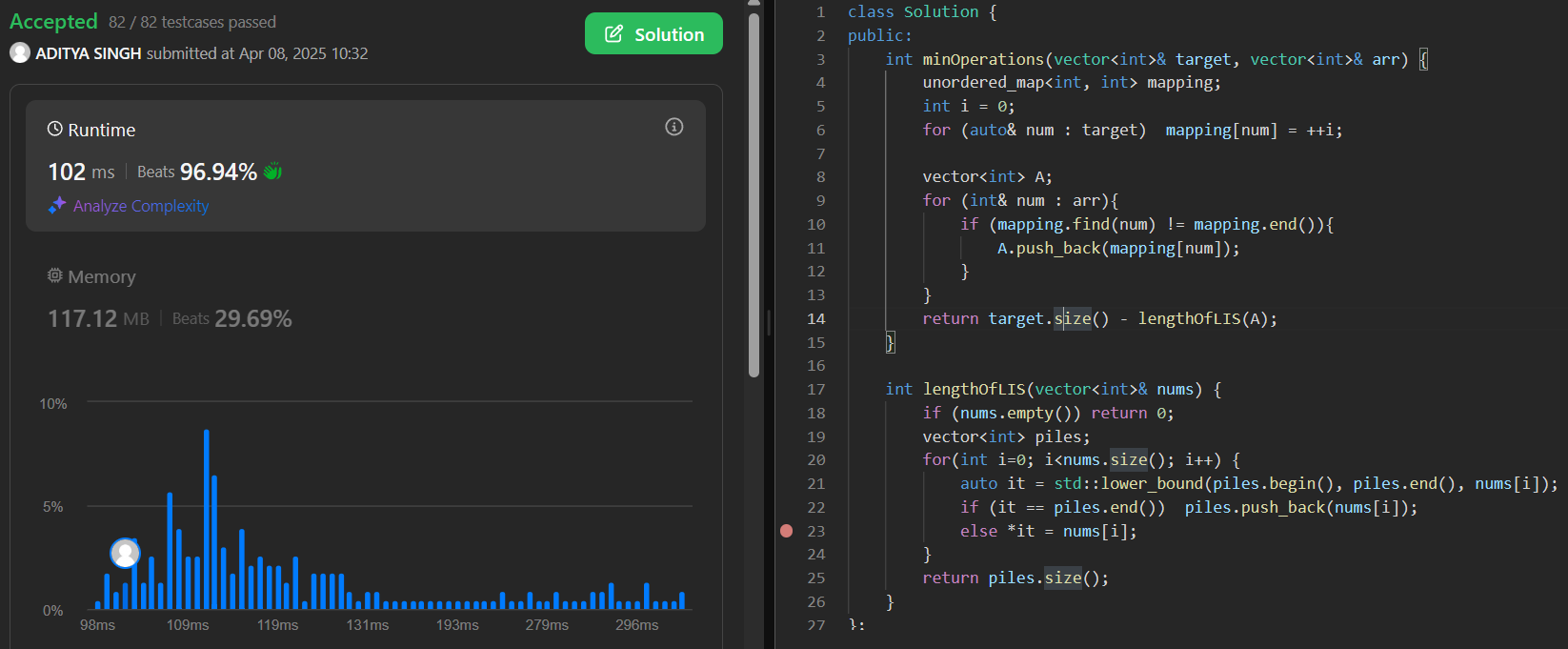
}

return piles.size();

}

};

Result:-



1. **Maximum Number of Tasks You Can Assign:-**

class Solution {

int s;

vector<int> ts, ws;

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 left = 0;

int right = min(n, m);

int answer = 0;

while (left <= right) {

int mid = (left + right)/2;

multiset<int> workersSet(workers.end() - mid, workers.end());

int pillCountRemaining = pills;

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

auto it = prev(workersSet.end());

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

if (pillCountRemaining == 0) break;

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

if (it == workersSet.end()) break;

pillCountRemaining--;

}

workersSet.erase(it);

}

if (workersSet.empty()) {

answer = mid;

left = mid + 1;

} else {

right = mid - 1;

}

}

return answer;

}

};

Result:-

