**ASSIGNMENT 8 (Greedy Programming)**

**AP LAB**

**Lavanya Saini – 22BCS15497**

**22BCS IOT 614-B**

1. **Maximum Units on a Truck**

class Solution {

public:

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

return a[1] > b[1];

}

int maximumUnits(vector<vector<int>>& boxes, int size) {

sort(boxes.begin(), boxes.end(), cmp);

int units = 0;

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

int count = min(size, boxes[i][0]);

units += count \* boxes[i][1];

size -= count;

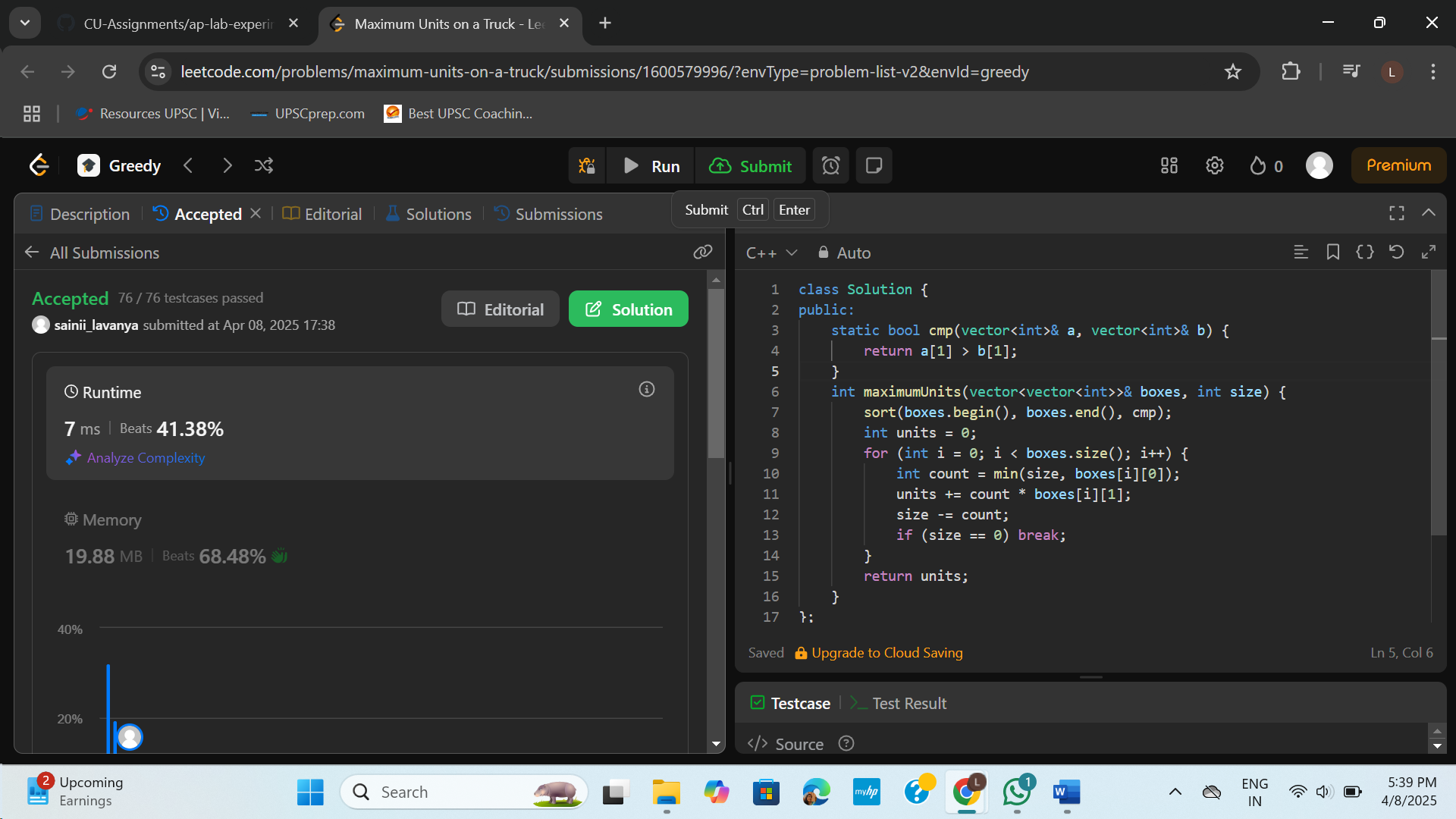
if (size == 0) break;

}

return units;

}

};



1. **Min Operations to make Array Increasing**

class Solution {

public:

int minOperations(vector<int>& nums) {

int op = 0;

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

if (nums[i] <= nums[i - 1]) {

int inc = (nums[i - 1] + 1) - nums[i];

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

op += inc;

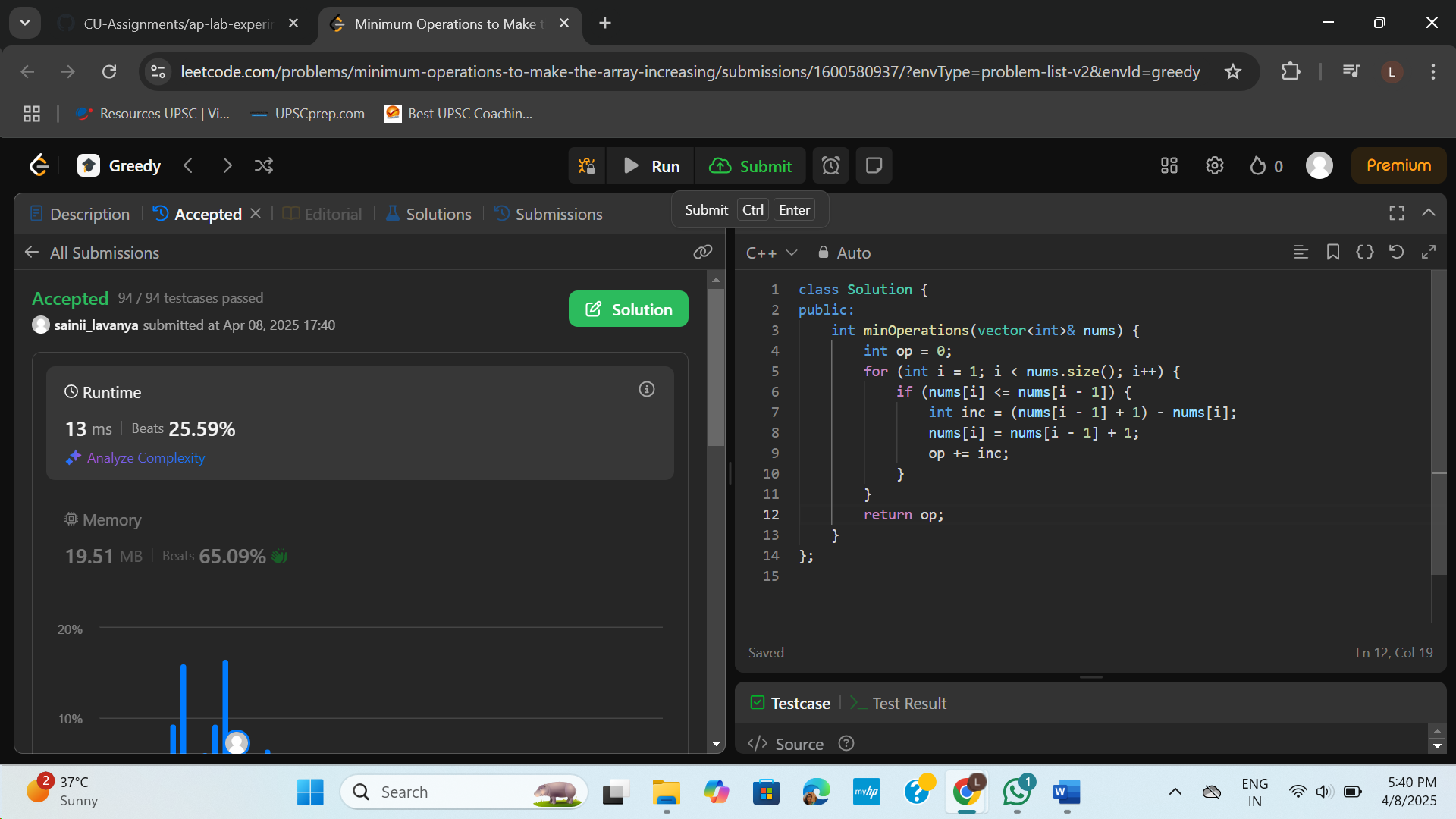
}

}

return op;

}

};



1. **Remove Stones to Maximise Total**

class Solution {

public:

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

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

while (k--) {

int top = pq.top();

pq.pop();

top -= top / 2;

pq.push(top);

}

int sum = 0;

while (!pq.empty()) {

sum += pq.top();

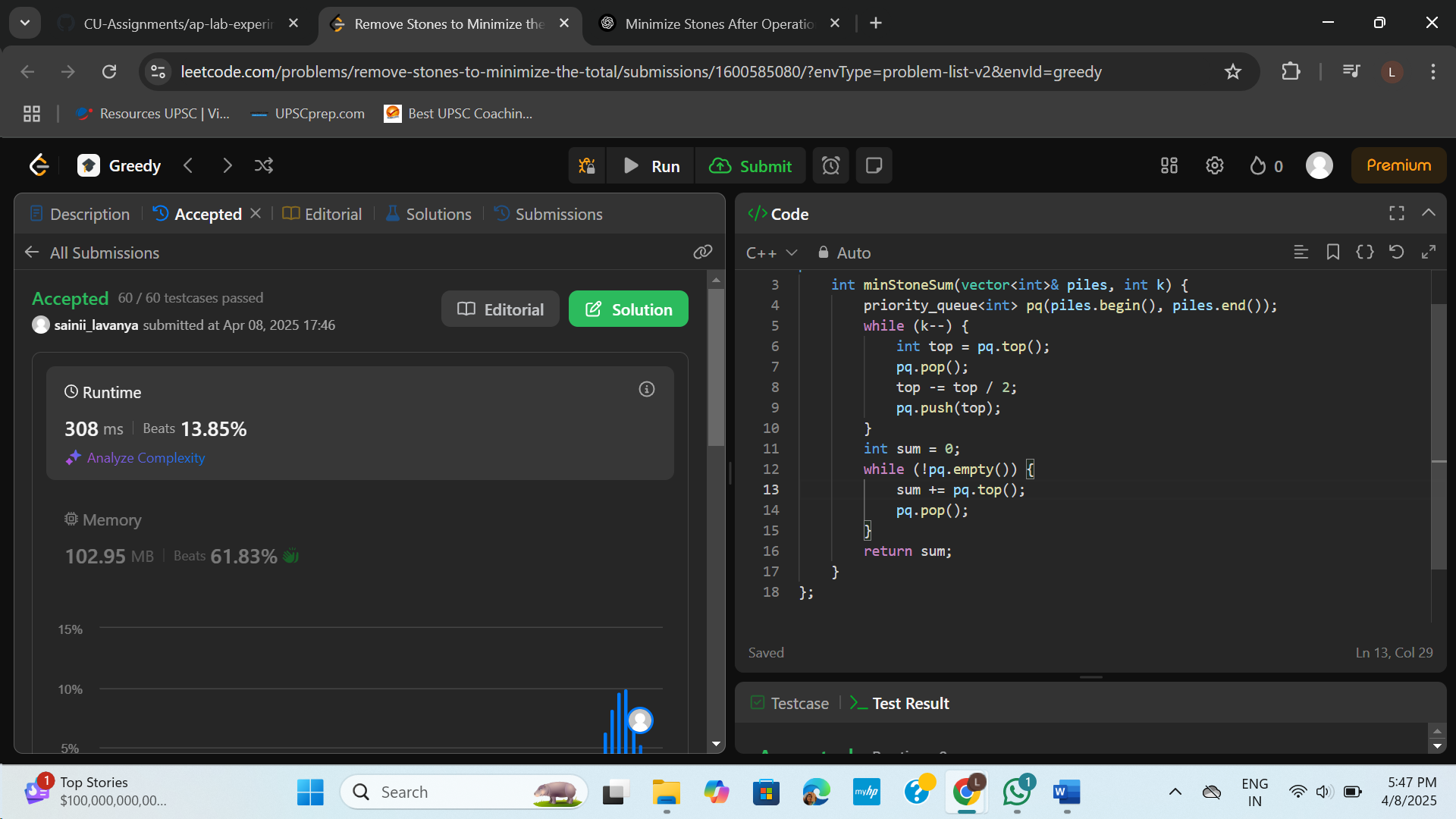
pq.pop();

}

return sum;

}

};



1. **Maximum Score from Removing Substring**

class Solution {

public:

int gain(string& s, char a, char b, int points) {

string temp = "";

int score = 0;

for (char c : s) {

if (!temp.empty() && temp.back() == a && c == b) {

temp.pop\_back();

score += points;

} else {

temp.push\_back(c);

}

}

s = temp;

return score;

}

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

int total = 0;

if (x > y) {

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

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

} else {

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

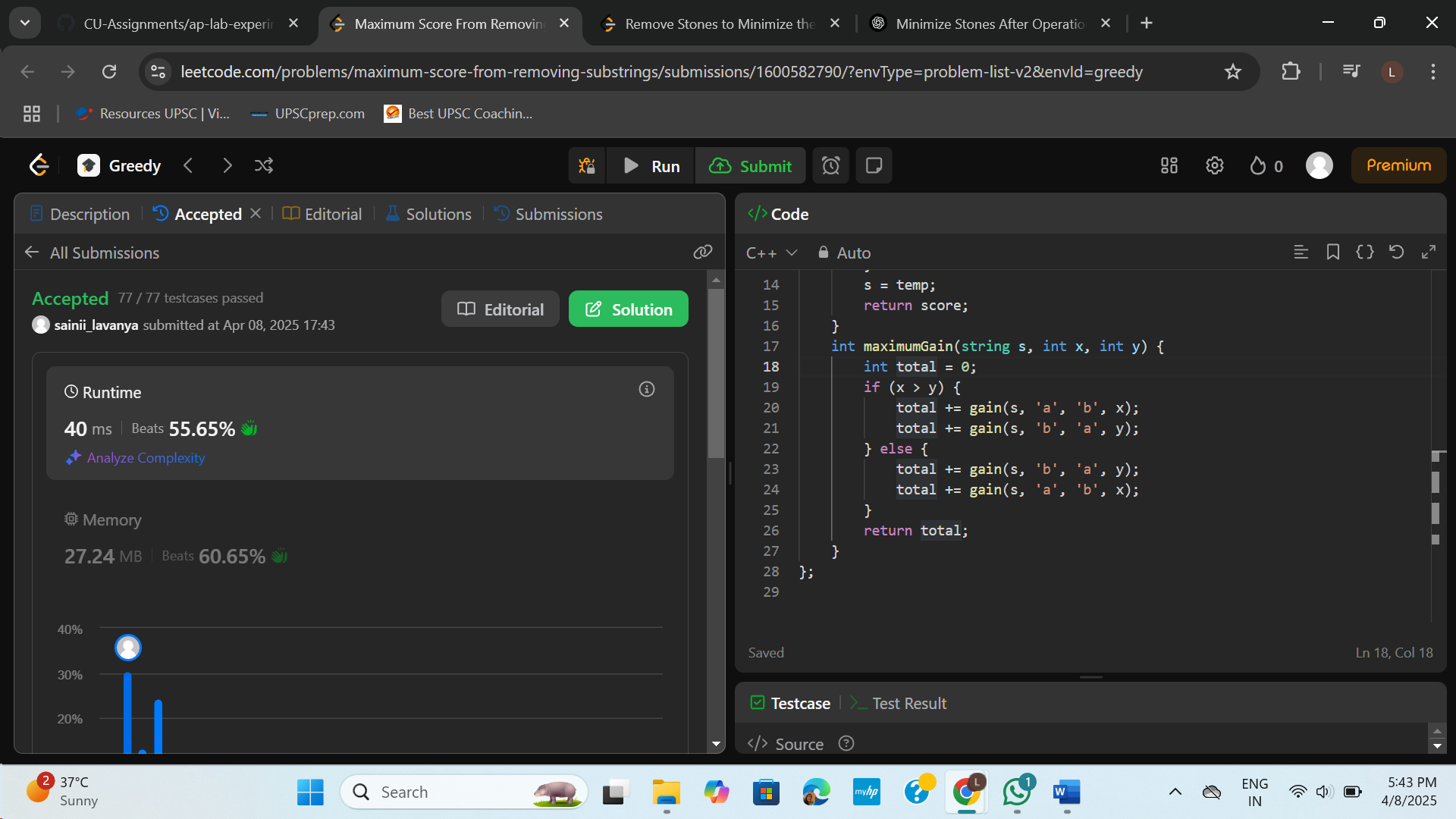
total += gain(s, 'a', 'b', x);

}

return total;

}

};



1. **Minimum Operations to make a Subsequence**

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> sequence;

for (int num : arr) {

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

sequence.push\_back(pos[num]);

}

}

vector<int> lis;

for (int num : sequence) {

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

if (it == lis.end())

lis.push\_back(num);

else

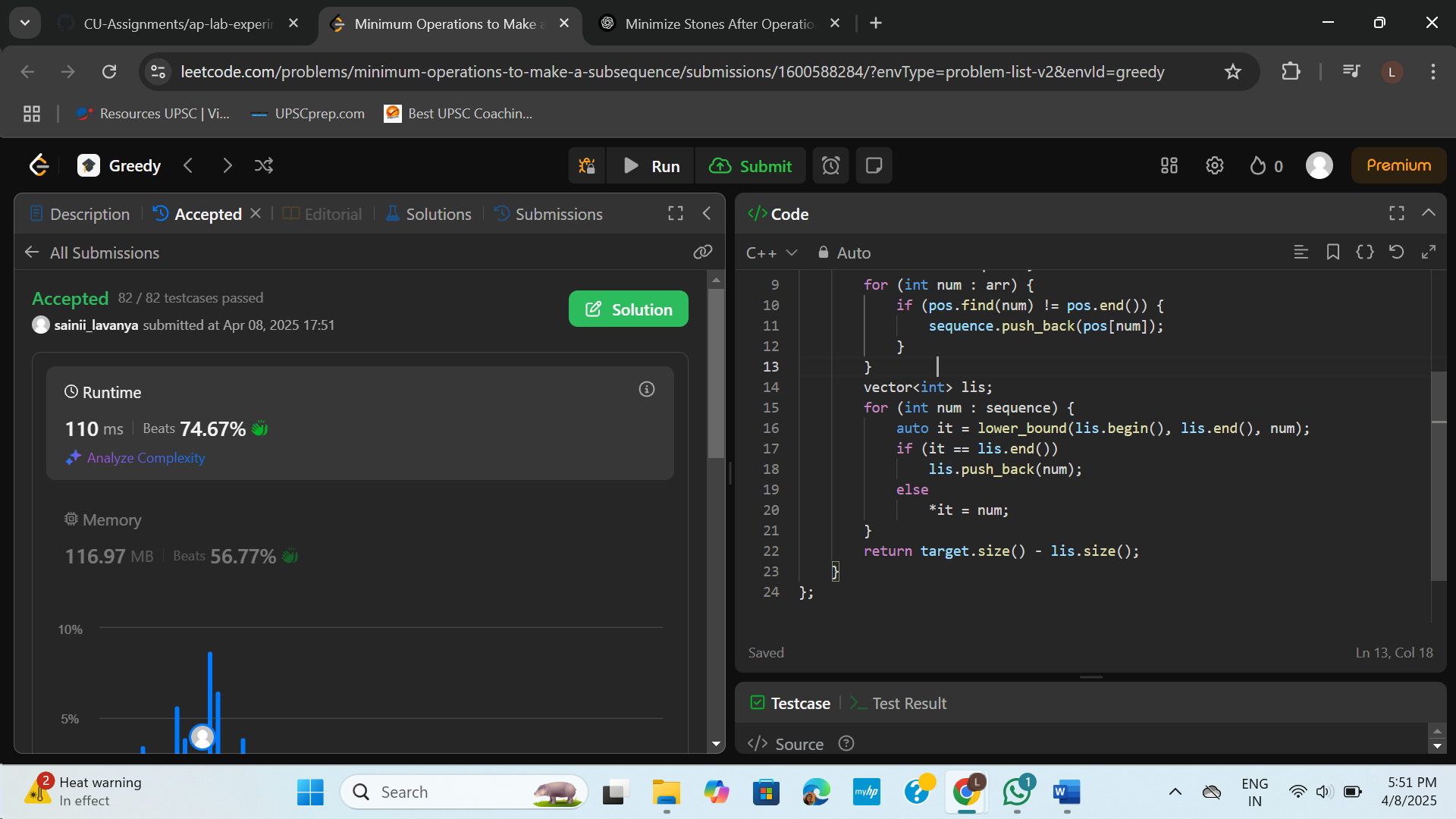
\*it = num;

}

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

}

};



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

class Solution {

public:

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

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

int remainingPills = pills;

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

int t = tasks[i];

auto it = wk.lower\_bound(t);

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

wk.erase(it);

} else {

if (remainingPills == 0) return false;

auto it2 = wk.lower\_bound(t - strength);

if (it2 == wk.end()) return false;

wk.erase(it2);

remainingPills--;

}

}

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 low = 0, high = min((int)tasks.size(), (int)workers.size());

int ans = 0;

while (low <= high) {

int mid = (low + high) / 2;

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

ans = mid;

low = mid + 1;

} else {

high = mid - 1;

}

}

return ans;

}

};

