# NAME- Akshit Boparai| UID- 22BCS14939 | SECTION- 601/A 1 Max Units on a Truck

class Solution {

public:

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

        priority\_queue<pair<int, int>> maxHeap;

        for (const auto& box : boxTypes) {

            maxHeap.push({box[1], box[0]});

        }

        int max\_in\_truck = 0;

        while (!maxHeap.empty() && truckSize > 0) {

            auto [unitsPerBox, numBoxes] = maxHeap.top();

            maxHeap.pop();

            int boxesToTake = min(truckSize, numBoxes);

            max\_in\_truck += boxesToTake \* unitsPerBox;

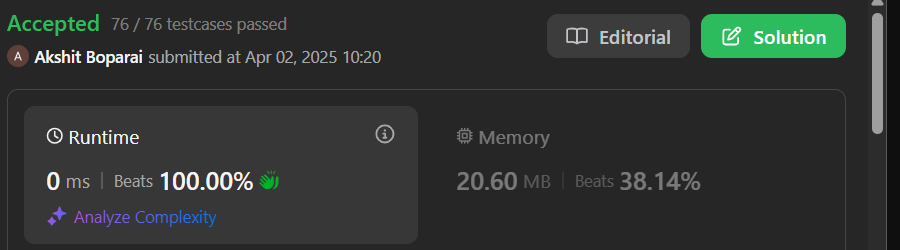
            truckSize -= boxesToTake;

        }

        return max\_in\_truck;

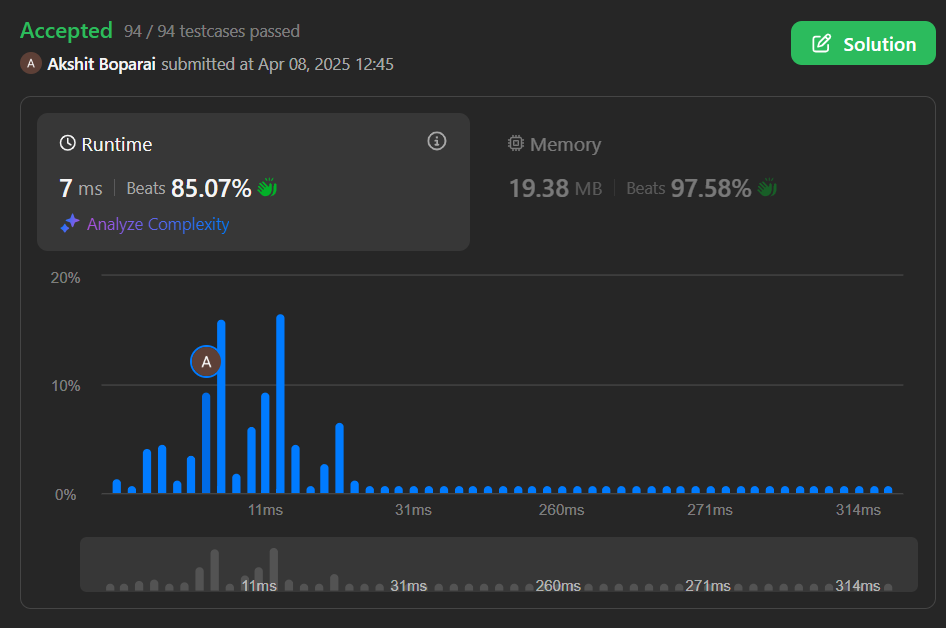
    }

};



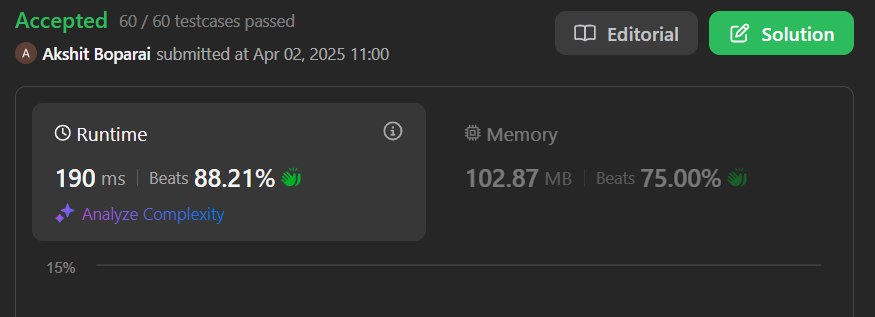
# MIN Operations to Make Array Increasing

class Solution {public: int minOperations(vector<int>& nums) { int count = 0; for (int i = 1; i < nums.size(); i++) { if (nums[i] <= nums[i - 1]) { int increment = nums[i - 1] - nums[i] + 1; count += increment; nums[i] += increment; } } return count; }};



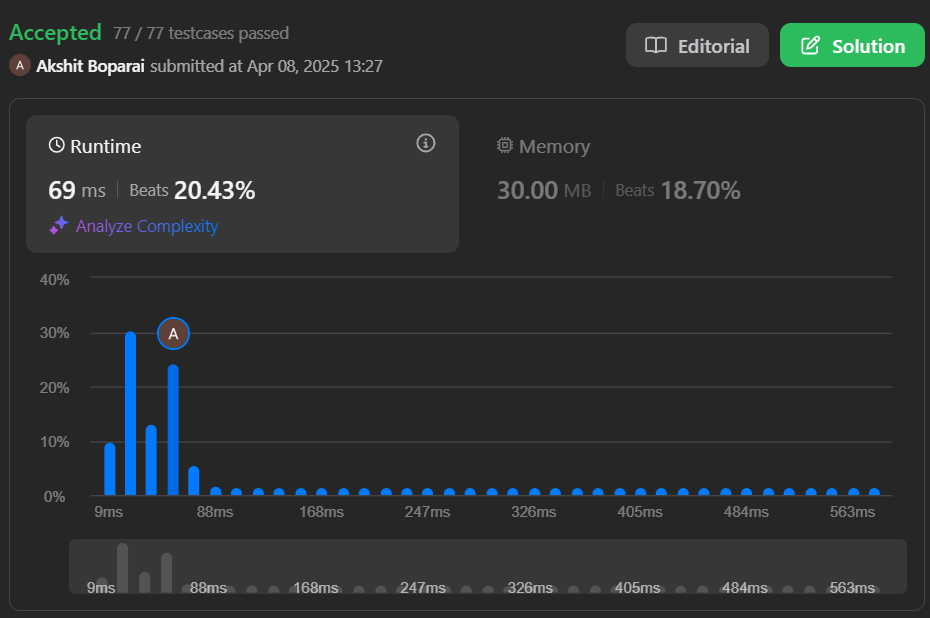
# Remove Stones to Minimize Total

# class Solution {public: int minStoneSum(vector<int>& piles, int k) { priority\_queue<int> maxHeap(piles.begin(), piles.end()); int sum = 0; for (int pile : piles) { sum += pile; } for (int i = 0; i < k; i++) { int maxPile = maxHeap.top(); maxHeap.pop(); int reducedPile = maxPile - floor(maxPile / 2); sum -= floor(maxPile / 2); maxHeap.push(reducedPile); } return sum; }};



# Max Score from Removing Substrings

# class Solution {public: int maximumGain(string s, int x, int y) { int total = 0; bool removeBAFirst = y > x; char firstChar = removeBAFirst ? 'b' : 'a'; char secondChar = removeBAFirst ? 'a' : 'b'; int firstGain = removeBAFirst ? y : x; int secondGain = removeBAFirst ? x : y; stack<char> st; string rem1; for (char c : s) { if (!st.empty() && st.top() == firstChar && c == secondChar) { st.pop(); total += firstGain; } else { st.push(c); } } while (!st.empty()) { rem1 += st.top(); st.pop(); } reverse(rem1.begin(), rem1.end()); string rem2; for (char c : rem1) { if (!rem2.empty() && rem2.back() == secondChar && c == firstChar) { rem2.pop\_back(); total += secondGain; } else { rem2.push\_back(c); } } return total; }};



# Min Operations to Make a Subsequence

# class Solution {

# public:

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

# unordered\_map<int, int> mapping;

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

# mapping[target[i]] = i;

# }

# vector<int> A;

# for (int num : arr) {

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

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

# }

# }

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

# }

# private:

# int lengthOfLIS(vector<int>& nums) {

# vector<int> piles;

# for (int num : nums) {

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

# if (it == piles.end()) {

# piles.push\_back(num);

# } else {

# \*it = num;

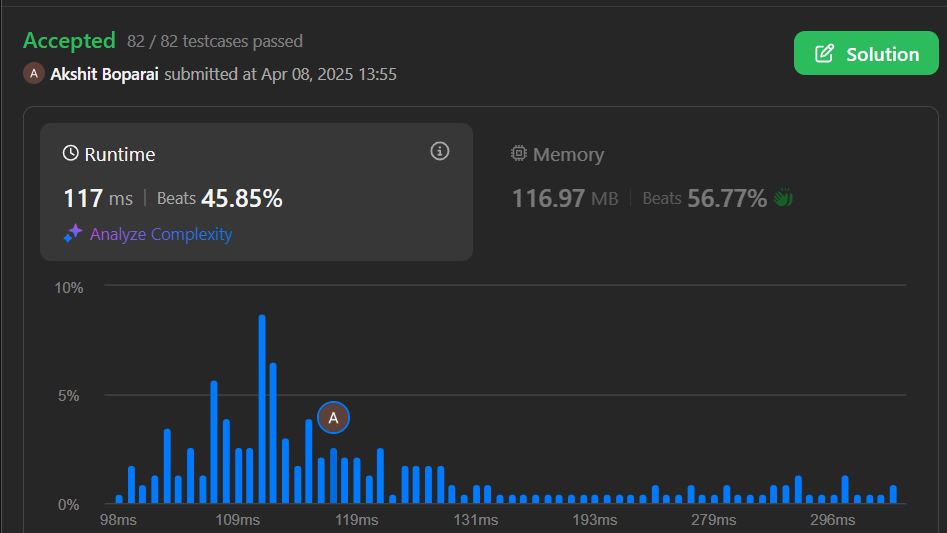
# }

# }

# return piles.size();

# }

# };



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

class Solution {public: int check(vector<int> &tasks, int take, map<int,int> count, int pills, int power) { while (take >= 1 && count.size()) { int task = tasks[take - 1]; auto it = count.end(); --it; if (task <= it->first) { --(it->second); if (it->second == 0) count.erase(it); --take; } else if (pills) { it = count.lower\_bound(task - power); if (it == count.end()) return 0; --(it->second); if (it->second == 0) count.erase(it); --pills; --take; } else { return 0; } } return take == 0; } int maxTaskAssign(vector<int>& t, vector<int>& w, int p, int s) { sort(t.begin(), t.end()); map<int,int> Count; for (auto &strength : w) Count[strength]++; int l = 0, r = t.size(), ans = 0; while (l <= r) { int mid = l + (r - l) / 2; if (check(t, mid, Count, p, s)) { ans = mid; l = mid + 1; } else { r = mid - 1; } } return ans; }};

