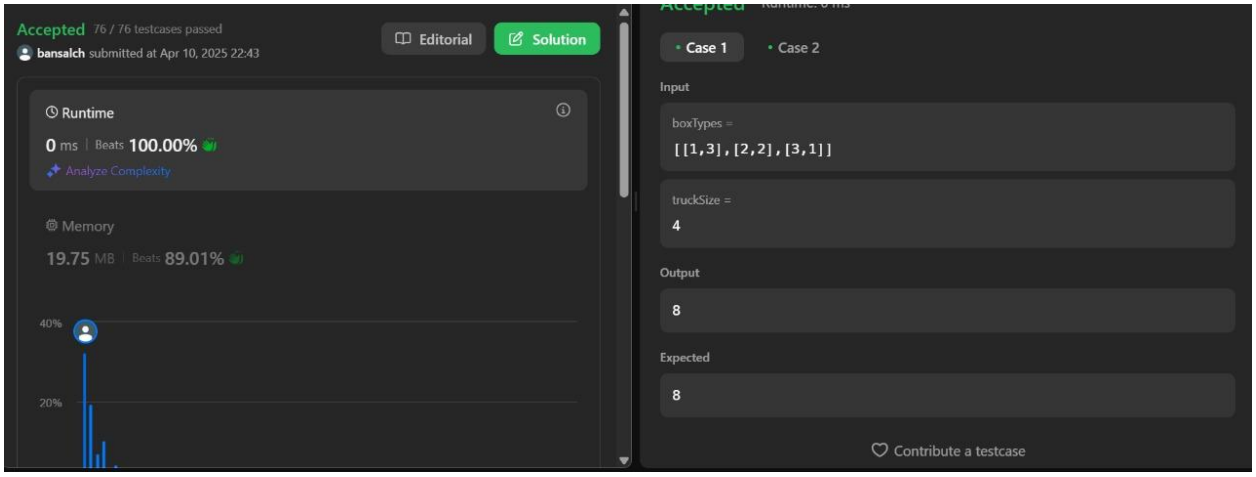
AP 8TH EXPERIMENT

Q1. Max Units on a Truck

Code:

```cpp
class Solution {
public:
    int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {
        sort(boxTypes.begin(), boxTypes.end(), [](auto &a, auto &b) {
            return a[1] > b[1];
        });
        int totalUnits = 0;
        for (auto& box : boxTypes) {
            int countOfBoxes = min(truckSize, box[0]);
            totalUnits += countOfBoxes * box[1];
            truckSize -= countOfBoxes;
            if (truckSize == 0) break;
        }
        return totalUnits;
    }
};
```

Output:

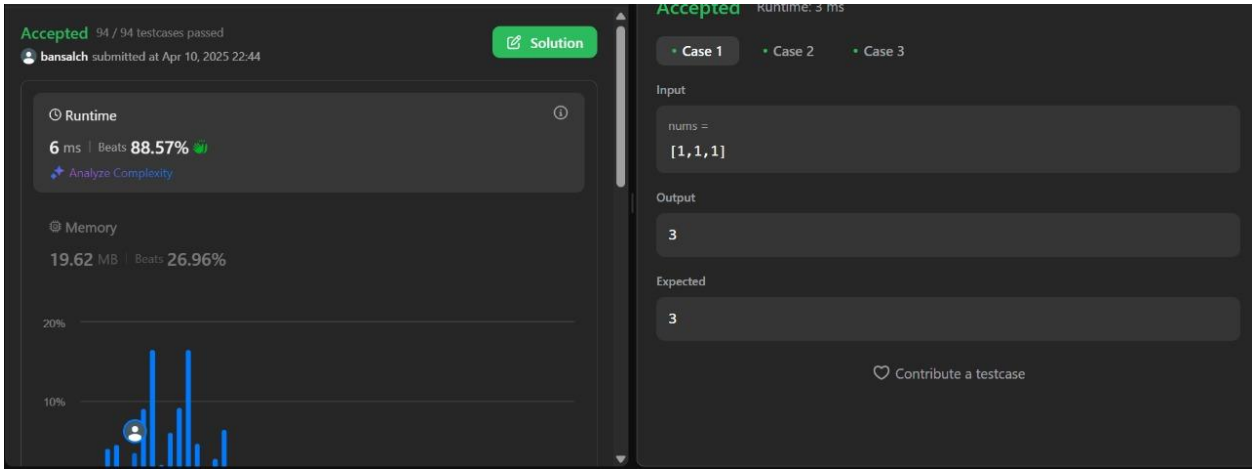


Q2. Min Operations to Make Array Increasing

Code:

```cpp
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 increment = (nums[i - 1] + 1) - nums[i];
                nums[i] = nums[i - 1] + 1;
                op += increment;
            }
        }
        return op;
    }
};
```

Output:

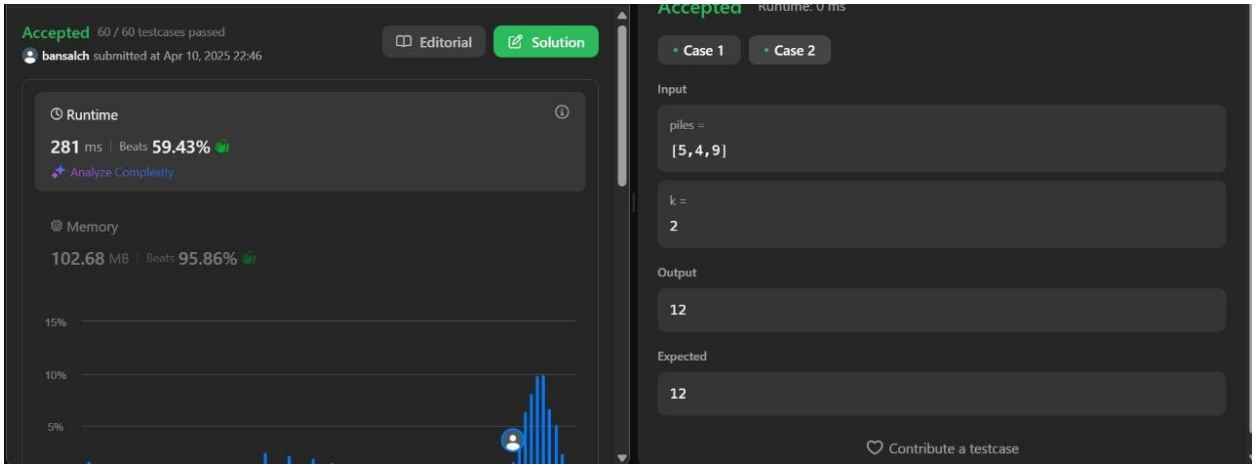


Q3. Remove Stones to Maximize Total

Code:

```cpp
class Solution {
public:
    int minStoneSum(vector<int>& piles, int k) {
        priority_queue<int> maxHeap(piles.begin(), piles.end());
        while (k-- > 0) {
            int largest = maxHeap.top();
            maxHeap.pop();
            maxHeap.push(largest - largest / 2);
        }
        int total = 0;
        while (!maxHeap.empty()) {
            total += maxHeap.top();
            maxHeap.pop();
        }
        return total;
    }
};
```

Output:

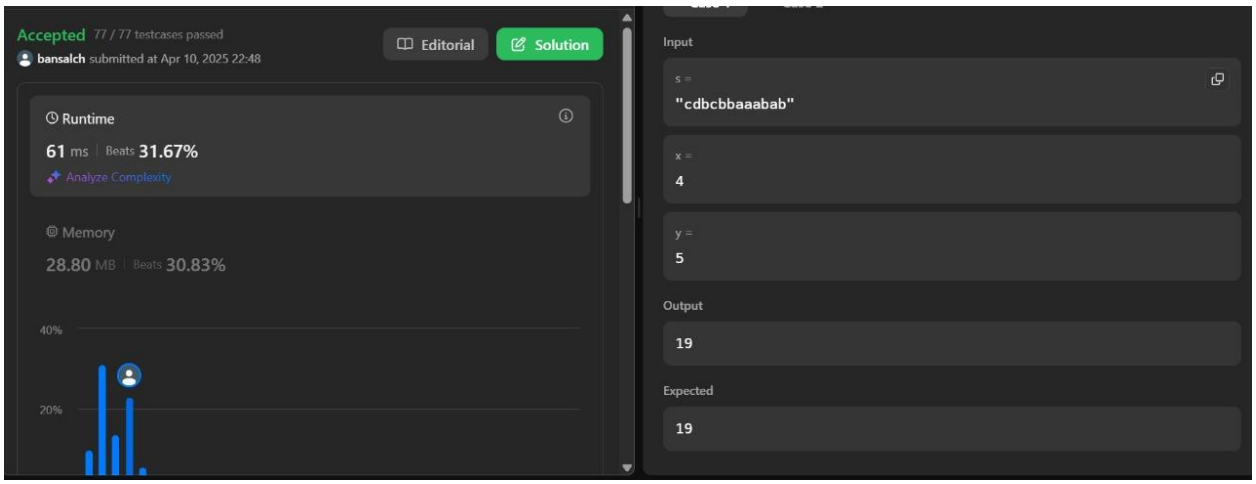


Q4. Max Score from Removing Substrings

Code:

```cpp
int maximumGain(string s, int x, int y) {
    if (x < y) {
        swap(x, y);
        for (char &c : s) {
            if (c == 'a') c = 'b';
            else if (c == 'b') c = 'a';
        } }
    int total = 0;
    stack<char> st;
    for (char c : s) {
        if (!st.empty() && st.top() == 'a' && c == 'b') {
            st.pop();
            total += x;
        } else {
            st.push(c);
        } }
    string remaining;
    while (!st.empty()) {
        remaining += st.top();
        st.pop();
    }
    reverse(remaining.begin(), remaining.end());
    for (char c : remaining) {
        if (!st.empty() && st.top() == 'b' && c == 'a') {
            st.pop();
            total += y;
        } else {
            st.push(c);
        } }
    return total;
```

Output:



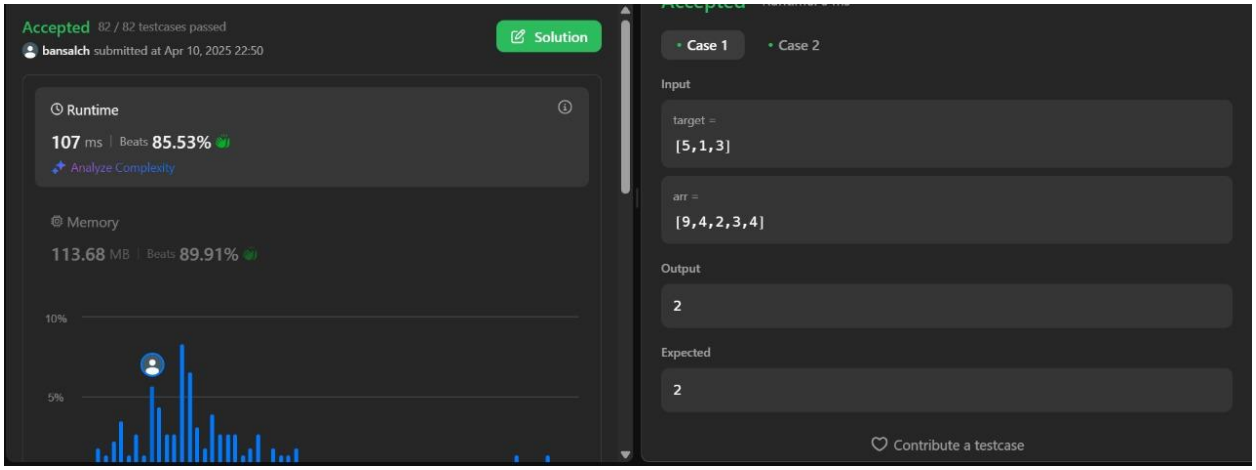
Q5. Min Operations to Make a Subsequence

Code:

```cpp
class Solution {
public:
    int minOperations(vector<int>& target, vector<int>& arr) {
        unordered_map<int, int> indexMap;
        for (int i = 0; i < target.size(); ++i) {
            indexMap[target[i]] = i;
        }
        vector<int> seq;
        for (int num : arr) {
            if (indexMap.find(num) != indexMap.end()) {
                int idx = indexMap[num];
                auto it = lower_bound(seq.begin(), seq.end(), idx);
                if (it == seq.end()) {
                    seq.push_back(idx);
                } else {
                    *it = idx;
                }
            }
        }

        return target.size() - seq.size();
    }
};
```

Output:



Q6. Max Number of Tasks You Can Assign

Code:

```
pills, int strength) {
        deque<int> dq(workers.end() - k, workers.end());
        int pillCount = pills;
        for (int i = k - 1; i >= 0; --i) {
            int task = tasks[i];
            if (!dq.empty() && dq.back() >= task) {
                dq.pop_back();
            } else if (!dq.empty() && pillCount > 0 && dq.front() + strength
>= task) {
                dq.pop_front();
                --pillCount;
            } else {
                return false;
            }    }
        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 result = 0;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (canAssign(mid, tasks, workers, pills, strength)) {
                result = mid;
                low = mid + 1;
            } else {
                high = mid - 1;
            }    }
        return result;
```

Output: