Advanced Pragramming

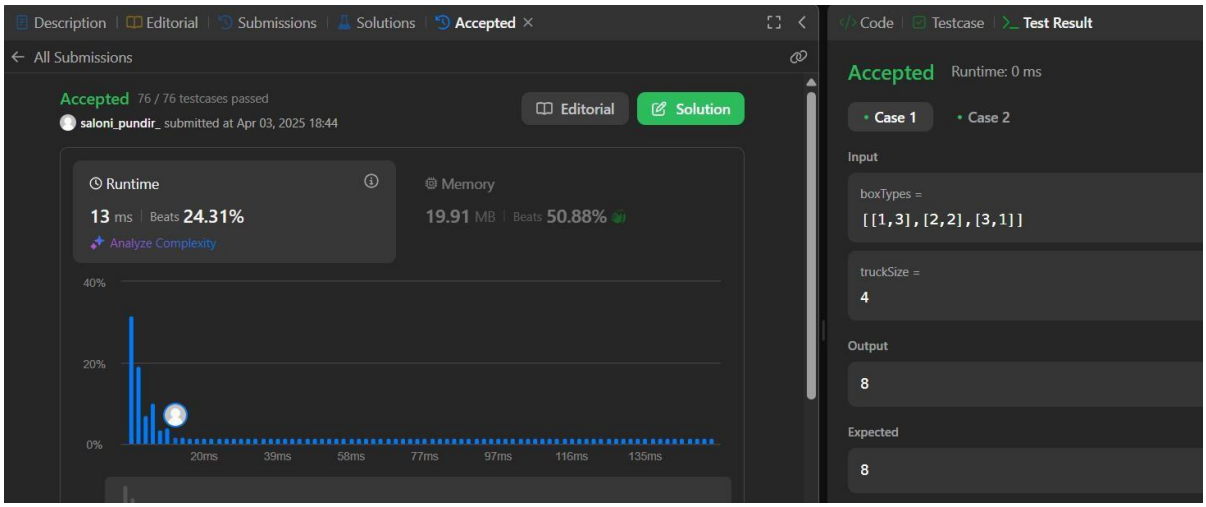
ASSIGNMENT 08

Q1. Max Units on a Truck.

Code:

```cpp
class Solution {
public:
    int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {
        for (int i=0;i<boxTypes.size();i++){
            reverse(boxTypes[i].begin(),boxTypes[i].end());
        }
        sort(boxTypes.begin(),boxTypes.end());
        reverse(boxTypes.begin(),boxTypes.end());
        int ans=0,i=0;
        while (truckSize>0 && i<boxTypes.size()){
            while (truckSize>0 && boxTypes[i][1]>0){
                ans+=boxTypes[i][0];
                truckSize--;
                boxTypes[i][1]--;
            }
            i++;
        }
        return ans;
    }
};
```

Output:

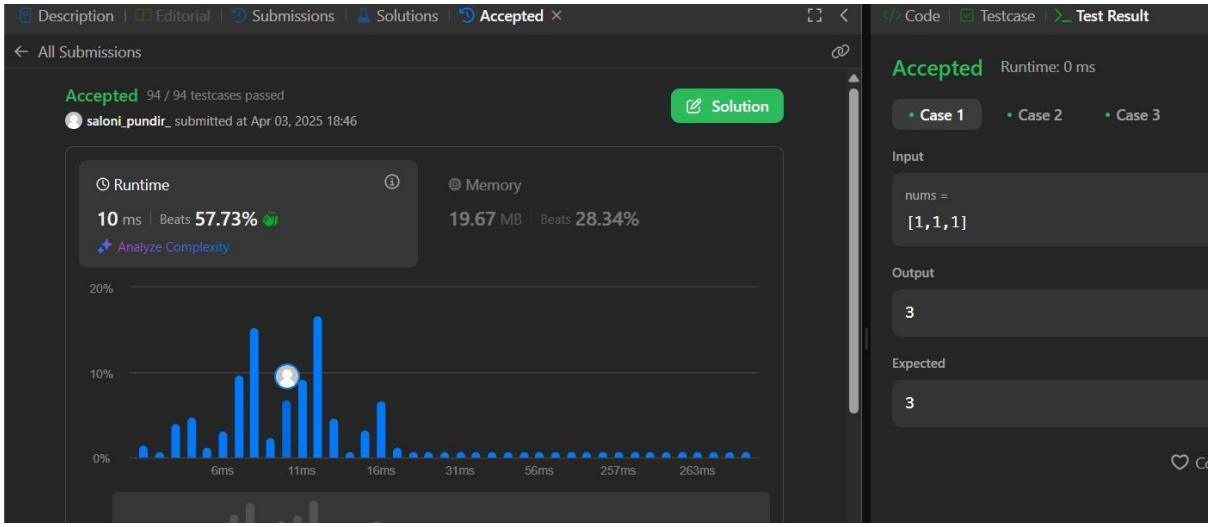


Q2. Min Operations to Make Array Increasing.

Code:

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

Output:



Q3. Remove Stones to Maximize Total.

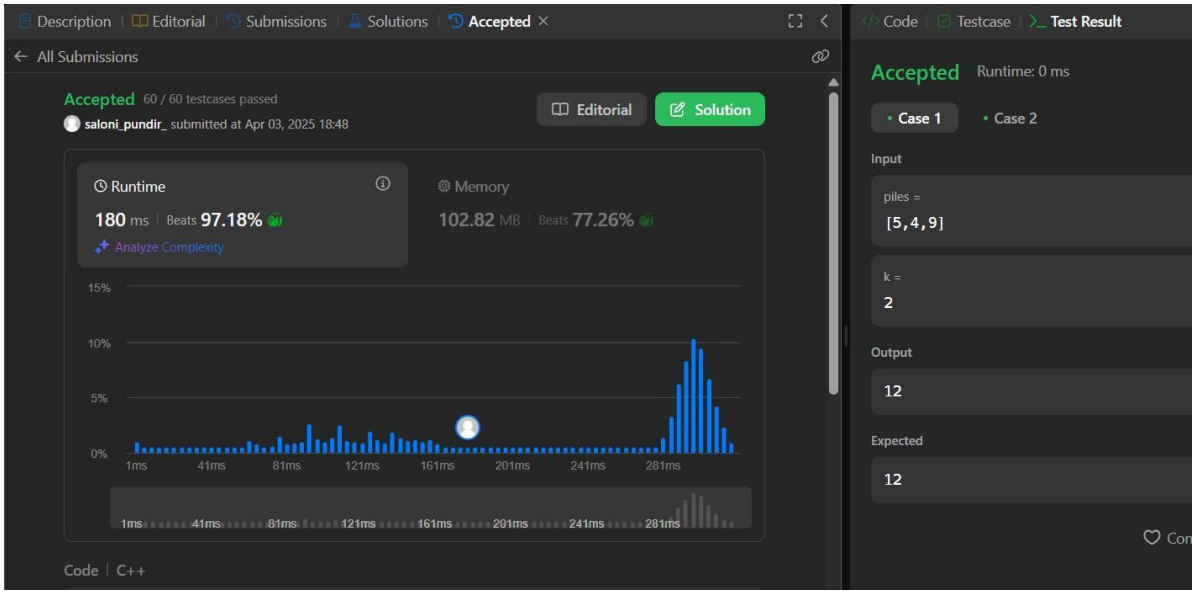
Code:

```cpp
class Solution {
public:
bool static help(int x,int y)
{
    return x>y;
}
    int minStoneSum(vector<int>& piles, int k) {

        int n=piles.size();
        priority_queue<int,vector<int>>pq(piles.begin(),piles.end());
        int ans=accumulate(piles.begin(),piles.end(),0);
        int i=0;
        while(k>0 && !pq.empty())
        {
            int temp=pq.top();
            pq.pop();
            ans-=(temp/2);
            pq.push(temp-temp/2);
            k--;
        }

        return ans;
    }
};
```

Output:



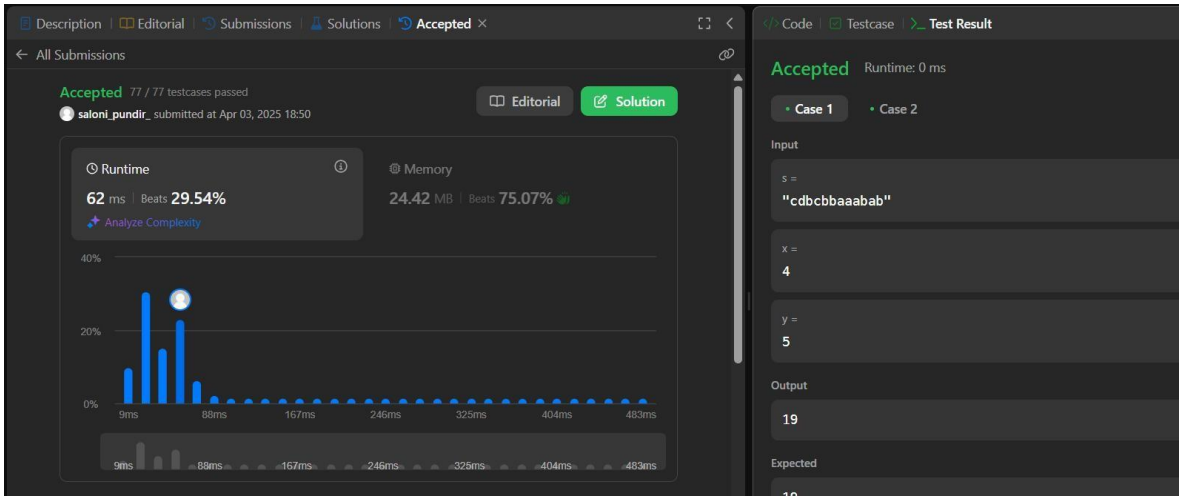
Q4. Max Score from Removing Substrings.

Code:

```cpp
class Solution {
public:
    int maximumGain(string s, int x, int y) {
        vector<char> v, v1;
        int ans = 0;
        for (auto a : s) {
            if (!v.empty()) {
                if (x > y) {
                    if (v.back() == 'a' && a == 'b') {
                        ans += x;
                        v.pop_back();
                    } else
                        v.push_back(a);
                } else {
                    if (v.back() == 'b' && a == 'a') {
                        ans += y;
                        v.pop_back();
                    } else
                        v.push_back(a);
                }
            } else {
                v.push_back(a);
            }
        }
        for (auto a : v)
            if (!v1.empty()) {
                if (x < y) {
                    if (v1.back() == 'a' && a == 'b') {
                        ans += x;
                        v1.pop_back();
                    } else
                        v1.push_back(a);
                } else {
                    if (v1.back() == 'b' && a == 'a') {
                        ans += y;
                        v1.pop_back();
                    } else
                        v1.push_back(a);
                }
            } else {
                v1.push_back(a);
            }

        return ans;
    }
};
```

Output:



Q5. Min Operations to Make a

Subsequence.

Code:

```cpp
lass Solution {
ublic:
    int minOperations(vector<int>& target, vector<int>& arr) {
        unordered_map<int, int> mp;
        for(int i = 0; i < target.size(); i++) mp[target[i]] = i;
        vector<int> v;
        for(int a: arr) if (mp.count(a)) v.push_back(mp[a]);
        int n = v.size(), ans = 0;
        vector<int> tail(n + 1, INT_MAX);
        tail[0] = INT_MIN;
        for(int a: v) {
            int b = upper_bound(tail.begin(), tail.begin() + min(ans + 1, n), a) - tail.begir
            if (b == 0 || (tail[b - 1] < a && tail[b] > a)) {
                tail[b] = a;
                ans = max(ans, b);
            }
        }
        return target.size() - ans;
    }
;
```

Output: