Experiment 8

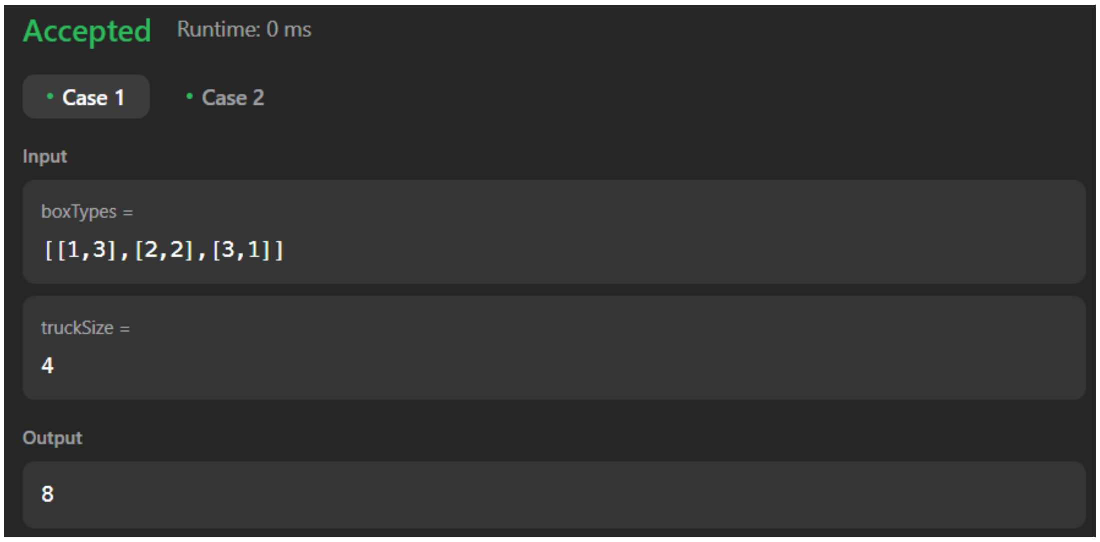
Name: Shuvam Dhara **UID: 22BET10009**
Branch: BE-IT **Section/Group: 22BET_701-A**
Semester: 6 **Date of Performance: 28-03-25**
Subject Name: Advanced Programming Lab-2 **Subject Code: 22ITP-351**

Problem 1. Max Units on a Truck- You are assigned to put some amount of boxes onto one truck. You are given a 2D array boxTypes, where boxTypes[i] = [numberOfBoxesi, numberOfUnitsPerBoxi]

Code:

```
class Solution {
public:
    static bool myfunction(vector<int>& a, vector<int>& b){
        return a[1] > b[1];
    }
    int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {
        sort(boxTypes.begin(),boxTypes.end(),myfunction);
                //greedily pick boxes till capacity is full
        int ans=0;
        for(auto box: boxTypes){
            int x=min(box[0],truckSize);
            ans+=(x*box[1]);  //adding units in ans
            truckSize-=x;  //reduce the capacity
            if(!truckSize) break;  //capacity full
        }
        return ans;
    }
};
```

Output:



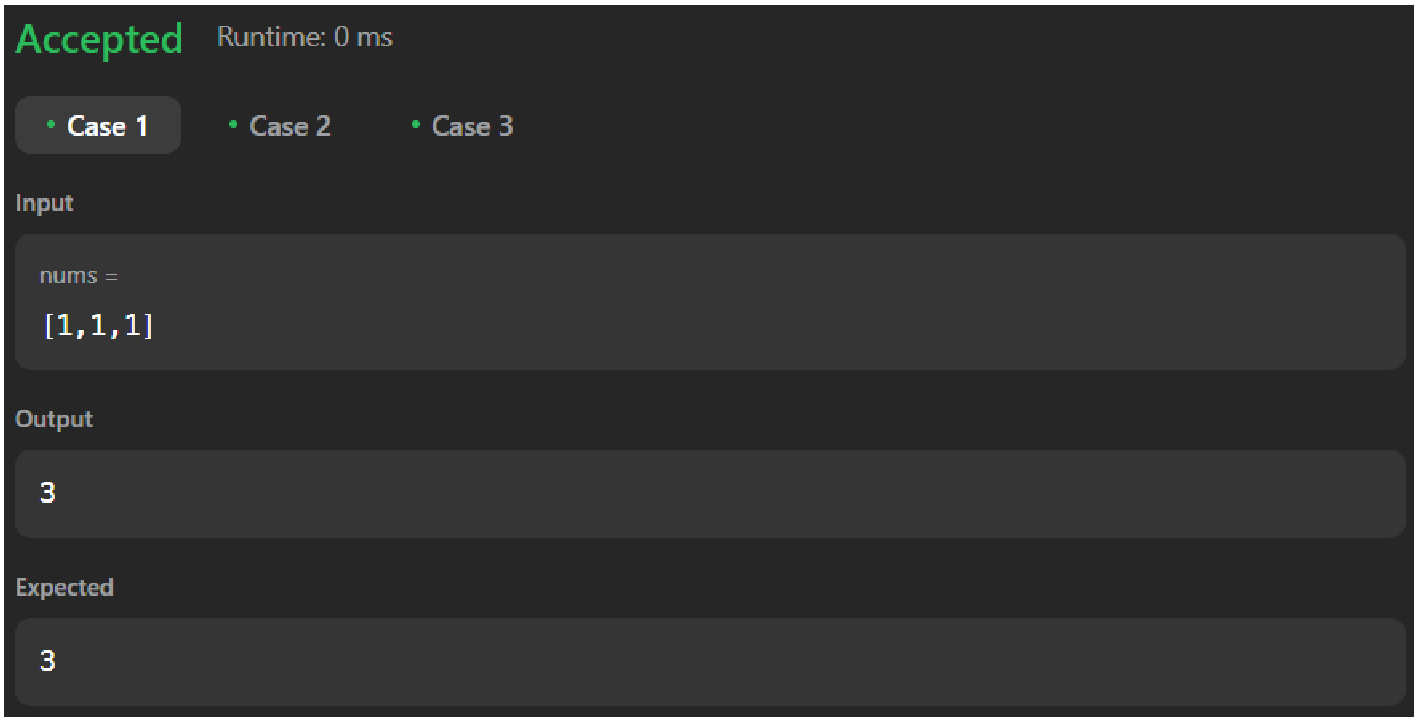
```
Accepted   Runtime: 0 ms
Case 1   Case 2
Input
boxTypes =
[[1,3],[2,2],[3,1]]
truckSize =
4
Output
8
```

Problem 2. Min Operations to make array incresing- You are given an integer array nums (0-indexed). In one operation, you can choose an element of the array and increment it by 1

Code:

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

Output:



```
nums =
[1,1,1]
```

```
3
```

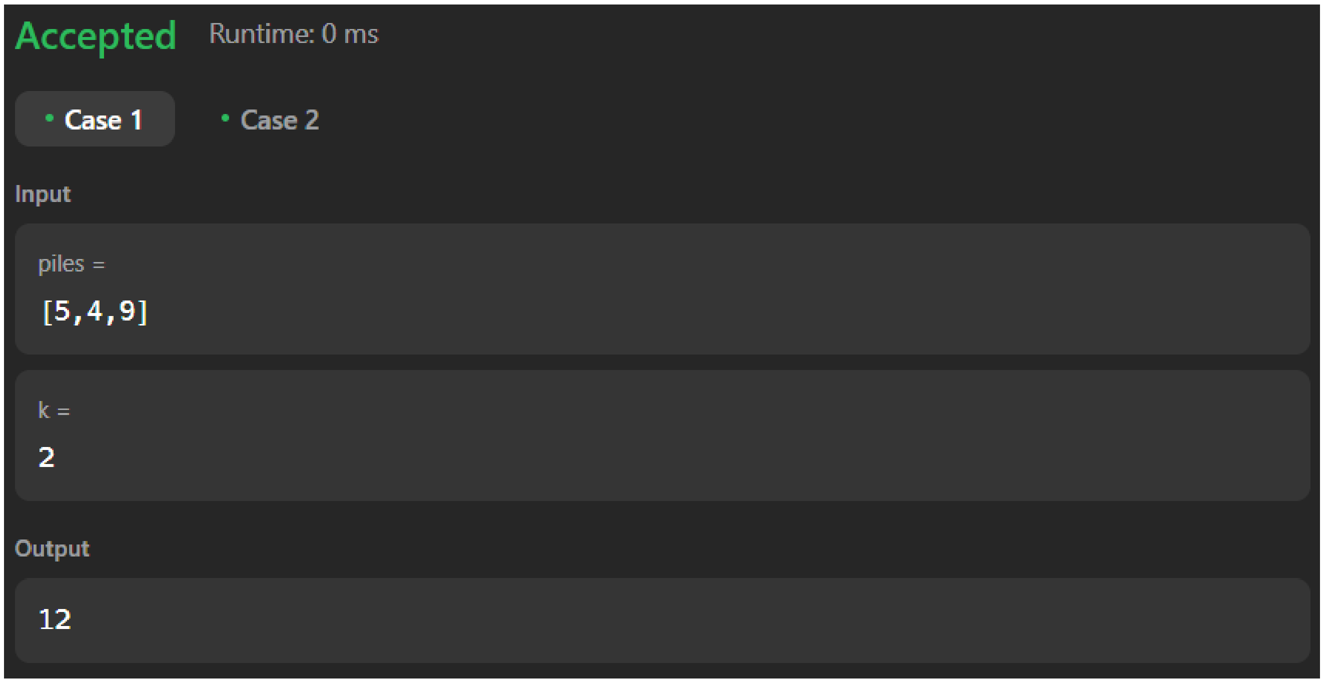
```
3
```

Problem 3. Remove stones to Maximize total - You are given a 0-indexed integer array piles, where piles[i] represents the number of stones in the ith pile, and an integer k. You should apply the following operation exactly k times

Code:

```
class Solution {
public:
    int minStoneSum(vector<int>& A, int k) {
        priority_queue<int> pq(A.begin(), A.end());
        int res = accumulate(A.begin(), A.end(), 0);
        while (k--) {
            int a = pq.top();
            pq.pop();
            pq.push(a - a / 2);
            res -= a / 2;
        }
        return res;
    }
};
```

Output:



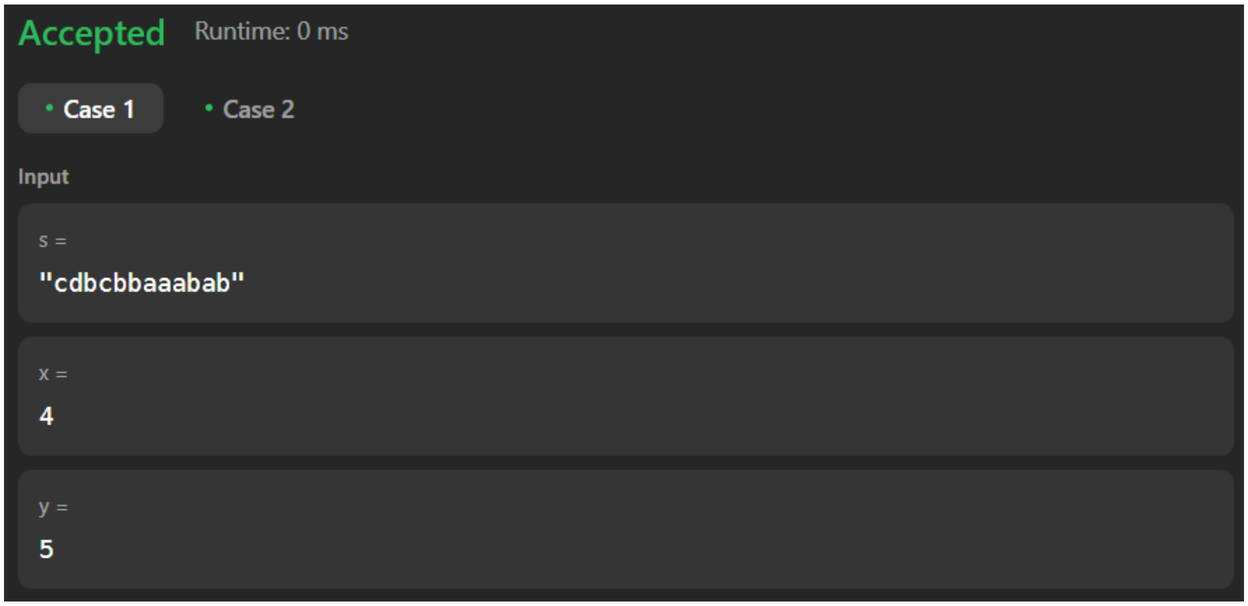
```
Accepted   Runtime: 0 ms
• Case 1   • Case 2
Input
piles =
[5,4,9]
k =
2
Output
12
```

Problem 4. Max Score from removing substrings- You are given a string s and two integers x and y. You can perform two types of operations any number of times.

Code:

```
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 {
                    bCount++;}}}
        result += min(aCount, bCount) * lesser;
        return result;
    }
};
```

Output:



Accepted Runtime: 0 ms

Case 1 Case 2

Input

s =
"cdbcbbaaabab"

x =
4

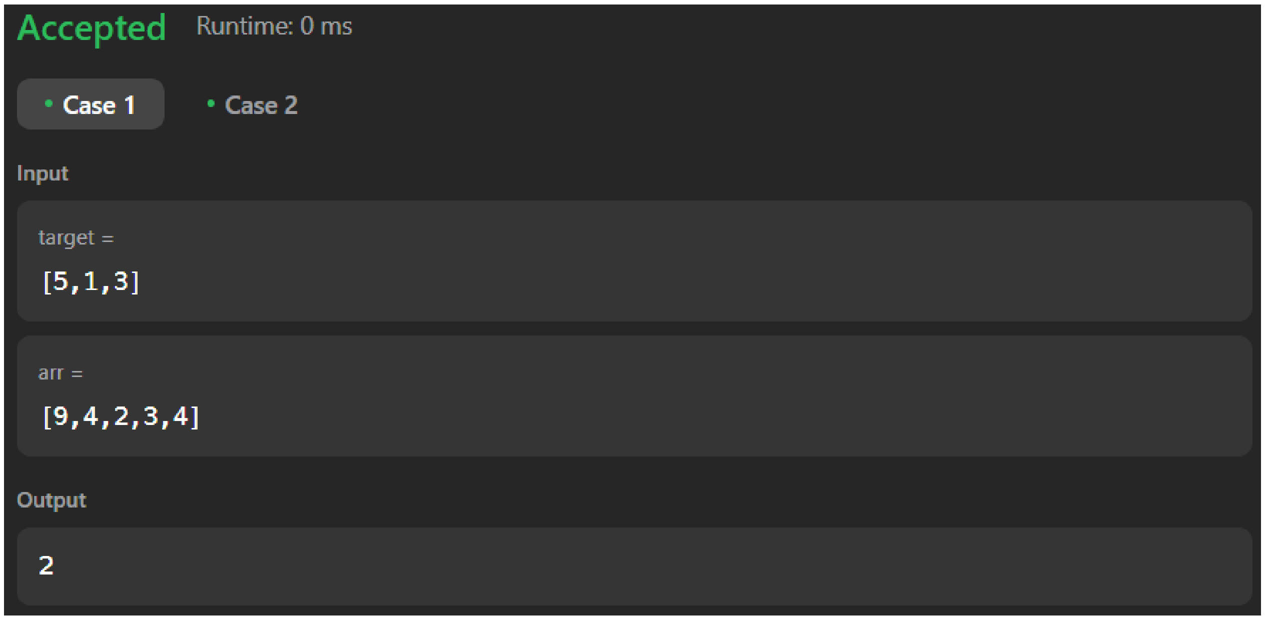
y =
5

Problem 5. Min operations to make a subsequence- You are given an array target that consists of distinct integers and another integer array arr that can have duplicates.In one operation, you can insert any integer at any position in arr. For example, if arr = [1,4,1,2], you can add 3 in the middle and make it [1,4,3,1,2]. Note that you can insert the integer at the very beginning or end of the array.

Code:

```
class Solution {
public:
    int minOperations(vector<int>& target, vector<int>& arr) {
    unordered_map<int, int> m;
    vector<int> stack;
    for (auto t : target)
        m[t] = m.size();
    for (auto n : arr) {
        auto it = m.find(n);
        if (it != end(m)) {
            if (stack.empty() || stack.back() < it->second)
                stack.push_back(it->second);
            else
                *lower_bound(begin(stack), end(stack), it->second) = it->second;
        }
    }
    return target.size() - stack.size();
}
};
```

Output:



```
Accepted   Runtime: 0 ms

• Case 1   • Case 2

Input

target =
[5,1,3]

arr =
[9,4,2,3,4]

Output

2
```

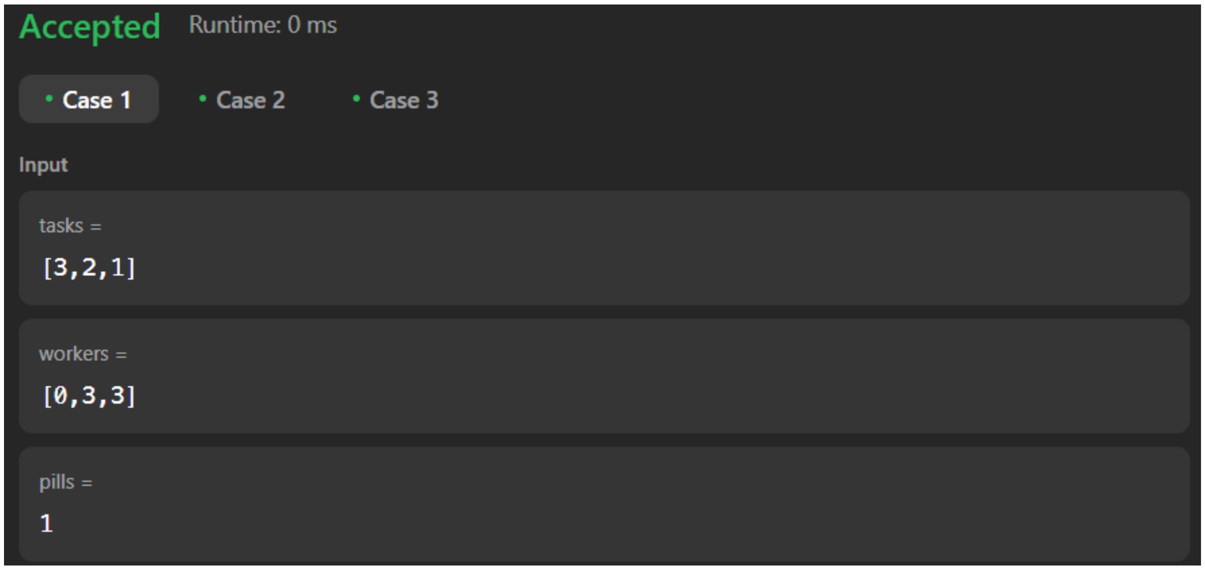
Problem 6. Max number of tasks you can assign-You have n tasks and m workers. Each task has a strength requirement stored in a 0-indexed integer array tasks, with the ith task requiring tasks[i] strength to complete. The strength of each worker is stored in a 0-indexed integer array workers, with the jth worker having workers[j] strength. Each worker can only be assigned to a single task and must have a strength greater than or equal to the task's strength requirement (i.e., workers[j] >= tasks[i])

Code:

```
class Solution {
public:
    bool can(vector<int>& t, vector<int>& w, int p, int s, int k) {
        multiset<int> ws(w.end() - k, w.end());
        for (int i = k - 1; i >= 0; --i) {
            auto it = ws.lower_bound(t[i]);
            if (it != ws.end()) ws.erase(it);
            else if (p-- && (it = ws.lower_bound(t[i] - s)) != ws.end()) ws.erase(it);
            else return false;
        }
        return true;
    }

    int maxTaskAssign(vector<int>& t, vector<int>& w, int p, int s) {
        sort(t.begin(), t.end()), sort(w.begin(), w.end());
        int l = 0, r = min(t.size(), w.size()), m;
        while (l < r) (m = (l + r + 1) / 2, can(t, w, p, s, m) ? l = m : r = m - 1);
        return l;
    }
};
```

Output:



```
tasks =
[3,2,1]

workers =
[0,3,3]

pills =
1
```