



Experiment 7

Student Name: Mayank Bhardwaj

UID :22BCS10988

Branch: BE-CSE

Section/Group: 22BCS_IOT-640(B)

Semester: 6th

Subject Code: 22CSP-351

Subject Name: AP Lab-II

A. Max Units on a Truck

1. Aim: You are assigned to put some amount of boxes onto one truck. You are given a 2D array boxTypes, where boxTypes[i] = [numberOfBoxes_i, numberOfUnitsPerBox_i]:

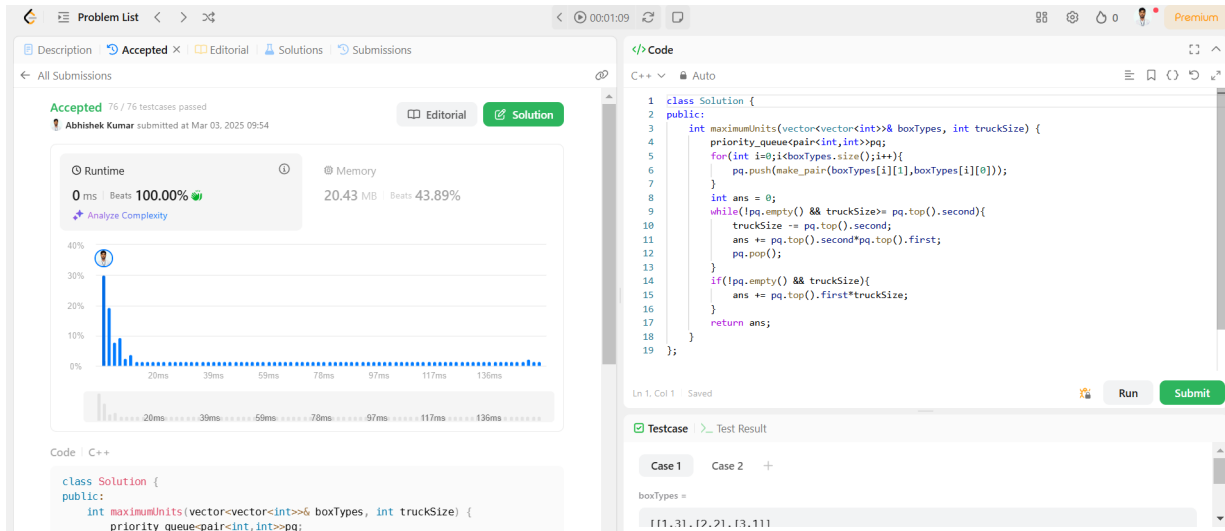
- numberOfBoxes_i is the number of boxes of type i.
- numberOfUnitsPerBox_i is the number of units in each box of the type i.

You are also given an integer truckSize, which is the maximum number of boxes that can be put on the truck. You can choose any boxes to put on the truck as long as the number of boxes does not exceed truckSize. Return the maximum total number of units that can be put on the truck.

2. Code

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

3. Output:



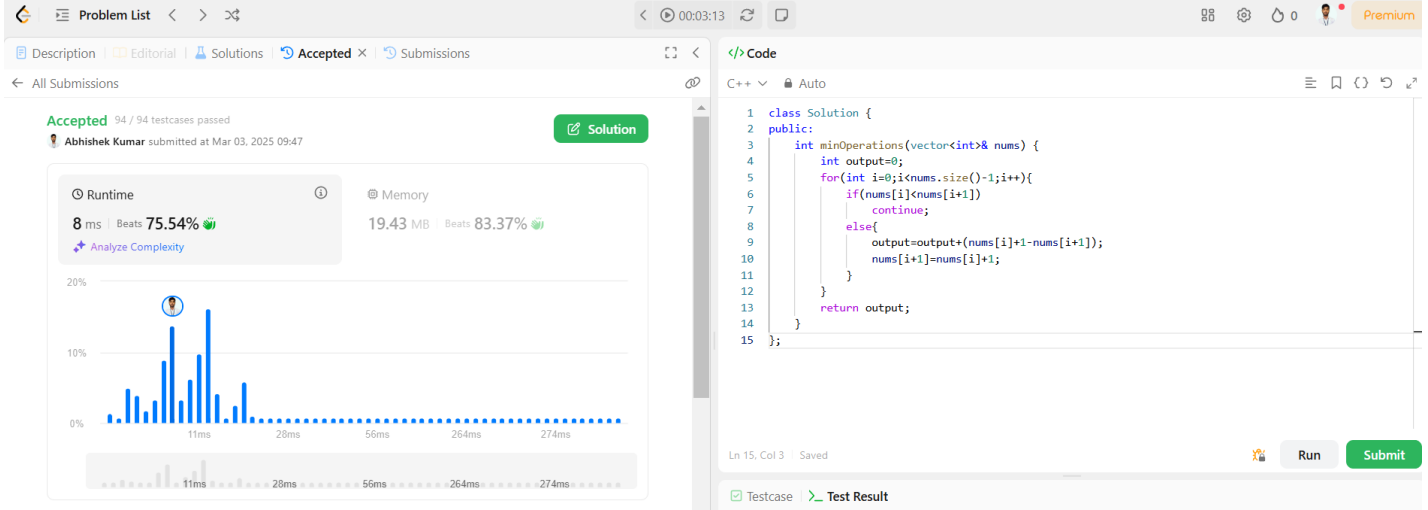
B. Minimum Operations to Make the Array Increasing

- 1. Aim:** 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. For example, if `nums = [1,2,3]`, you can choose to increment `nums[1]` to make `nums = [1,3,3]`. Return the minimum number of operations needed to make `nums` strictly increasing. An array `nums` is strictly increasing if `nums[i] < nums[i+1]` for all $0 \leq i < \text{nums.length} - 1$. An array of length 1 is trivially strictly increasing.

2. Code

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

3. Output:



C. Remove Stones to Minimize the Total

- Aim:** You are given a 0-indexed integer array `piles`, where `piles[i]` represents the number of stones in the *i*th pile, and an integer `k`. You should apply the following operation exactly `k` times: Choose any `piles[i]` and remove $\text{floor}(\text{piles}[i] / 2)$ stones from it. Notice that you can apply the operation on the same pile more than once. Return the minimum possible total number of stones remaining after applying the `k` operations. $\text{floor}(x)$ is the greatest integer that is smaller than or equal to x (i.e., rounds x down).

2. Code

```

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

```

3. Output:

