Experiment: - 8

Student Name: Jatin Garg
UID: 22BCS15676
Branch: CSE
Section/Group: 22BCS IOT-640/B
Semester: 6th
Date of Performance: 19/03/2025
Subject Name: Advanced Programming Lab-2
Subject Code: 22CSP-351

Problem -1

1. **Aim:** Max Units on a Truck

2. **Objective:**

- **Optimize loading of boxes onto a truck:** Learn how to maximize the total units of boxes that can be loaded given a truck's size limit, applying strategies to make the best use of available space.
- **Sort boxes by units per box:** Understand how sorting boxes based on the number of units per box can help prioritize which boxes to load first, ensuring the most valuable boxes are placed on the truck.
- **Apply greedy algorithm techniques:** Gain hands-on experience with greedy algorithms, which make locally optimal choices at each step, to achieve the global maximum of units loaded on the truck.
- **Work with 2D arrays and loops:** Improve your ability to handle and manipulate 2D arrays, as well as use loops and conditionals to process data efficiently in coding tasks.
- **Handle space constraints and optimization:** Learn how to manage situations where space is limited and how to optimize the use of resources, like loading boxes in the most efficient way possible.

3. **Implementation/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 totalUnits = 0, i = 0;
        while (truckSize > 0 && i < boxTypes.size()) {
            if (boxTypes[i][0] <= truckSize) {
                totalUnits += boxTypes[i][0] * boxTypes[i][1];
                truckSize -= boxTypes[i][0];
            } else {
                totalUnits += truckSize * boxTypes[i][1];
                truckSize = 0;
```

```
            }
            i++;
        }
        return totalUnits;
    }
};
```

4. Output

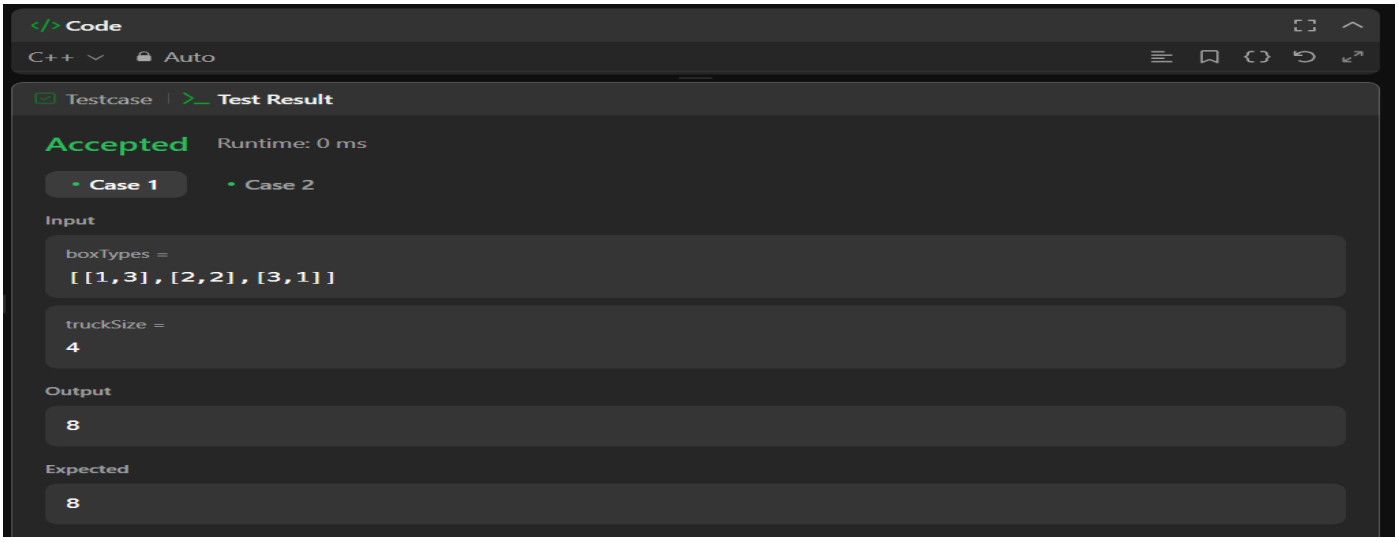

Figure 1

5. Learning Outcomes:

- **Efficient sorting and data processing:** Develop a clear understanding of sorting data based on specific criteria (like units per box) to solve real-world optimization problems effectively.
- **Calculating totals with loops and conditions:** Master the use of loops and conditional statements to calculate totals, ensuring correct results even with varying input sizes and constraints.
- **Handling edge cases:** Learn how to deal with different edge cases, such as when the truck runs out of space or there are more boxes than available space.
- **Strengthen problem-solving skills:** Enhance your ability to break down complex problems into simpler steps, applying algorithms and logic to find efficient solutions.
- **Optimize resource allocation:** Gain experience in maximizing resource use, such as truck space, by applying strategies that ensure the best possible use of available resources.

Problem-2

1. **Aim:** Min Operations to make array increasing.
2. **Objectives:**

- **Make an array strictly increasing:** Learn how to modify an array so that each number is greater than the previous one by making the fewest changes.
- **Use the smallest number of operations:** Understand how to increment elements efficiently to achieve the required increasing order with minimal changes.
- **Apply logic to find differences:** Learn how to compare consecutive elements and calculate how much an element needs to increase to maintain strict order.
- **Work with loops and conditionals:** Improve programming skills by using loops and conditions to check and update elements in an array.
- **Solve real-world optimization problems:** Understand how to optimize solutions by making the smallest possible changes to meet given constraints.

3. Implementation/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]) {
                int diff = nums[i - 1] - nums[i] + 1;
                nums[i] += diff;
                operations += diff;
            }
        }
        return operations;
    }
};
```

4. Output:

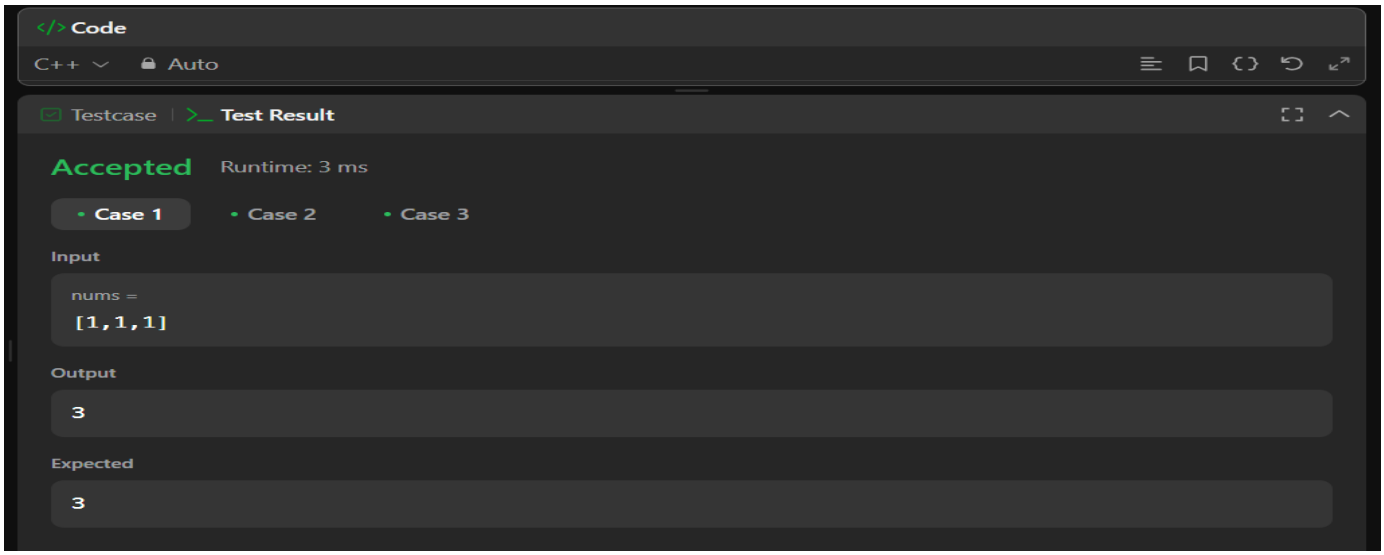

Figure 2

5. Learning Outcomes:

- **Understand array modifications:** Gain the ability to analyse and update an array to meet specific conditions using the least number of operations.
- **Use loops to check and adjust values:** Develop skills in using loops and conditionals to compare and modify elements efficiently.
- **Optimize problem-solving strategies:** Learn how to find the smallest number of changes needed to achieve a required goal in an algorithm.
- **Handle edge cases in constraints:** Be able to manage situations where numbers are already increasing or require multiple adjustments.
- **Improve algorithmic thinking:** Strengthen problem-solving skills by applying logical reasoning and efficient strategies to achieve the best result.

Problem: - 3

1. Aim: Max Score from removing substrings

2. Objectives:

- Remove specific substrings for maximum points: Learn how to remove "ab" and "ba" from a string to earn the highest possible score by applying the best order of operations.
- Use stack-based string processing: Understand how to efficiently remove substrings using a stack approach, making the process faster and more structured.
- Compare different operation orders: Learn how choosing the right sequence of removals (based on points assigned) can maximize the final score.
- Optimize string manipulation: Improve problem-solving skills by handling large strings efficiently without unnecessary operations or extra memory usage.
- Apply greedy algorithm concepts: Understand how a greedy approach helps in making the best choice at each step to achieve the maximum total score.

3. Implementation/Code:

```
class Solution {
public:
    int maximumGain(string s, int x, int y) {
        int score = 0;
        if (x > y) {
            score += removePair(s, 'a', 'b', x);
            score += removePair(s, 'b', 'a', y);
        } else {
            score += removePair(s, 'b', 'a', y);
            score += removePair(s, 'a', 'b', x);
        }
        return score;
    }
    int removePair(string &s, char first, char second, int points) {
        string temp = "";
```

```
            int score = 0;
            for (char c : s) {
                if (!temp.empty() && temp.back() == first && c == second) {
                    temp.pop_back();
                    score += points;
                } else {
                    temp.push_back(c);
                }
            }
            s = temp;
            return score;
        }
};
```

4. Output:

Figure 3

5. Learning Outcomes:

- **Understand substring removal strategies:** Gain the ability to remove specific pairs from a string while maintaining efficiency and correctness.
- **Improve problem-solving with stacks:** Learn how to use a stack-like method to keep track of character sequences and remove pairs dynamically.
- **Develop logical thinking for optimization:** Understand how to determine the best order of operations to achieve the highest possible score.
- **Handle large input sizes efficiently:** Learn how to manage operations on long strings while keeping execution time within acceptable limits.
- **Strengthen algorithmic skills:** Improve the ability to design and implement efficient algorithms that maximize output while minimizing computational cost.