Experiment 7

Student Name: Rhythm Tyagi
UID: 22BCS17203
Branch: CSE
Section/Group:IOT_NTPP_602-A
Semester: 6th
Date of Performance:20-01-25
Subject Name: AP2
Subject Code: 22CSP-351

Aim: Greedy Approach

1) Maximum Units on a Truck
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.

2) Maximum 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.Return *the maximum points you can gain after applying the above operations on* s.

3) Minimum Operations to Make the Array Increasing
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.

Objective: To determine the maximum depth of a binary tree by finding the longest path from the root to a leaf node using recursive DFS (O(n)) or iterative BFS (O(n)) approaches.

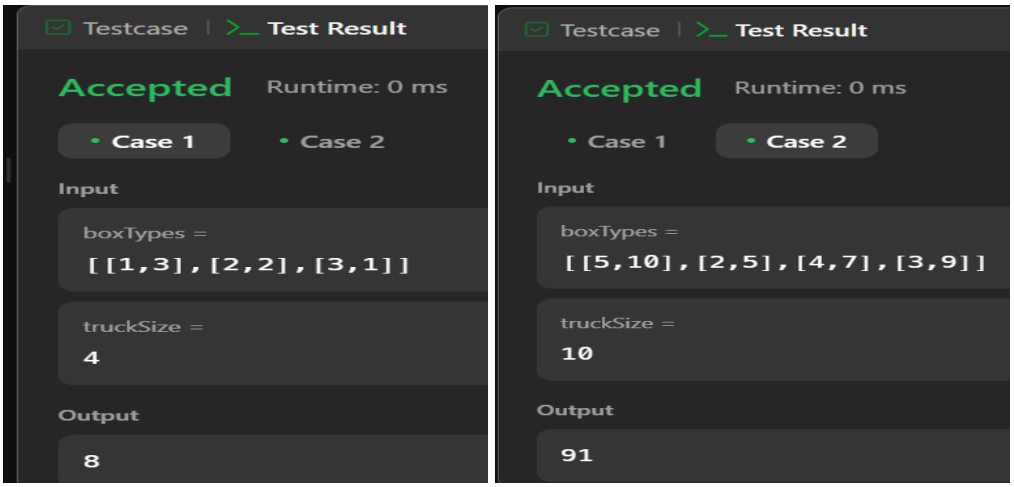
Algorithm 1:

1. **Sort** boxTypes in descending order
based on units per box.
2. **Initialize** maxUnits = 0.
3. **Iterate** through boxTypes:
   - Take the minimum of available boxes or remaining truck capacity.
   - Update maxUnits by multiplying selected boxes with units per box.
   - Reduce truckSize accordingly.
   - **Stop** if truck is full.
4. **Return** maxUnits

Code 1:

```
import java.util.Arrays;
class Solution {
    public int maximumUnits(int[][] boxTypes, int truckSize) {
        // Sort the boxes in descending order based on units per box
        Arrays.sort(boxTypes, (a, b) -> b[1] - a[1]);
        int maxUnits = 0;  // Stores the maximum units we can load
        for (int[] box : boxTypes) {
            int numBoxes = box[0];  // Number of boxes available
            int unitsPerBox = box[1];  // Units per box
            if (truckSize <= 0) break;  // If truck is full, stop
            int boxesToTake = Math.min(numBoxes, truckSize);  // Take as many boxes as possible
            maxUnits += boxesToTake * unitsPerBox;  // Update the total units
            truckSize -= boxesToTake;  // Reduce the truck capacity
        }
            return maxUnits;  // Return the maximum units we can load
    }
}
```

Output 1:



Code 2:

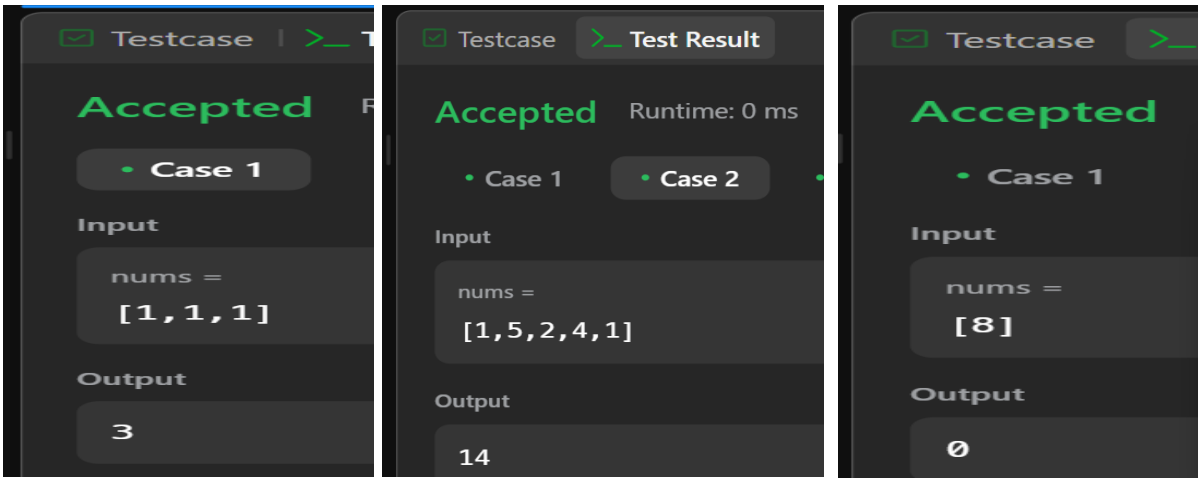
```
import java.util.Stack;
class Solution {
    public int maximumGain(String s, int x, int y) {
        if (y > x) return maximumGain(new StringBuilder(s).reverse().toString(), y, x); // Process
higher value first
        return removeSubstrings(s, 'a', 'b', x) + removeSubstrings(s, 'b', 'a', y);
    }
    private int removeSubstrings(String s, char first, char second, int points) {
        Stack<Character> stack = new Stack<>();
        int score = 0;
        for (char c : s.toCharArray()) {
            if (!stack.isEmpty() && stack.peek() == first && c == second) {
                stack.pop();
                score += points;
            } else {
                stack.push(c);
            }
        }
        return score;
    }
}
```

Output 2:

Code 3:

```
class Solution {
    public int minOperations(int[] nums) {
        int operations = 0;
            for (int i = 1; i < nums.length; i++) {
            if (nums[i] <= nums[i - 1]) {
                int increment = nums[i - 1] - nums[i] + 1;
                nums[i] += increment;
                operations += increment;
            }
        }
            return operations;
    }
}
```

Output:



Learning Outcomes:

1. Understanding greedy algorithms for optimal selection (Maximum Units on a Truck).
2. Applying stack-based or greedy approaches to maximize score from substring removals.
3. Implementing array manipulation techniques to achieve a strictly increasing sequence.