DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Experiment-7

Student Name: Shristi Rai **UID: 22BCS515330**

Branch: BE-CSE **Section/Group: _NTPP_IOT-602-A**

Semester: 6th **Date of Performance: 10/03/2025**

Subject Name: AP Lab **Subject Code: 22CSP-351**

1. Aim: Greedy Approach.

- Problem 1.2.1: Maximum Units on a Truck
- Problem 1.2.2: Min Operations to make array incresing
- Problem 1.2.2: Maximum Score From Removing Substrings

2. Objective:

Maximize the total number of units that can be loaded onto a truck within its box capacity. Determine the minimum number of increments needed to make an array strictly increasing. Minimize the total number of stones in piles by repeatedly removing half the stones from the largest pile.

3. Theory:

Maximum Units on a Truck: The problem involves loading boxes onto a truck to maximize the number of units. By sorting the box types in descending order based on units per box and greedily picking as many as possible within the truck's capacity, we achieve an optimal solution.

Minimum Operations to Make an Array Strictly Increasing: Given an integer array, the goal is to make it strictly increasing by incrementing elements as needed. By iterating through the array and ensuring each element is greater than the previous one, we count the number of operations required to achieve the condition efficiently.

Minimum Stone Sum After k Operations: The task is to minimize the total number of stones in piles by repeatedly removing half the stones (rounded down) from the pile with the most stones, exactly k times. Using a max heap (priority queue) ensures we always reduce the largest pile first, leading to the minimal possible sum after k operations.

4. Code:

Maximum Units on a Truck:

```
import java.util.Arrays;

class Solution {
  public int maximumUnits(int[][] boxTypes, int truckSize) {
    // Sort boxTypes in descending order based on units per box
    Arrays.sort(boxTypes, (a, b) -> b[1] - a[1]);

    int maxUnits = 0;

    for (int[] box : boxTypes) {
      int count = Math.min(truckSize, box[0]); // Take as many boxes
      as possible maxUnits += count * box[1]; // Add corresponding
      units
      truckSize -= count; // Reduce available truck space

      if (truckSize == 0) break; // Stop if the truck is full
    }

    return maxUnits;
  }
}
```

Minimum Operations to Make an Array Strictly Increasing:

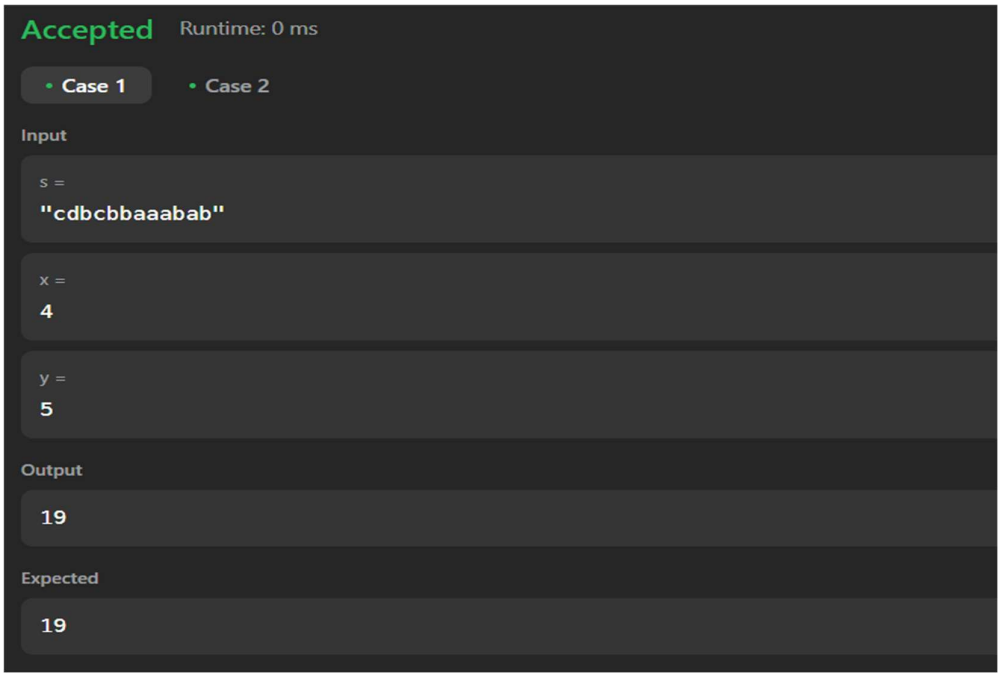
```
  class Solution {
    public int minOperations(int[]
      nums) { int operations = 0;

      for (int i = 1; i < nums.length;
        i++) { if (nums[i] <= nums[i
        - 1]) {
          int increment = nums[i - 1] - nums[i] + 1; // Calculate
          required increment nums[i] += increment; // Make nums[i]
          strictly greater
          operations += increment; // Count operations
        }
      }

      return operations;
    }
  }
```

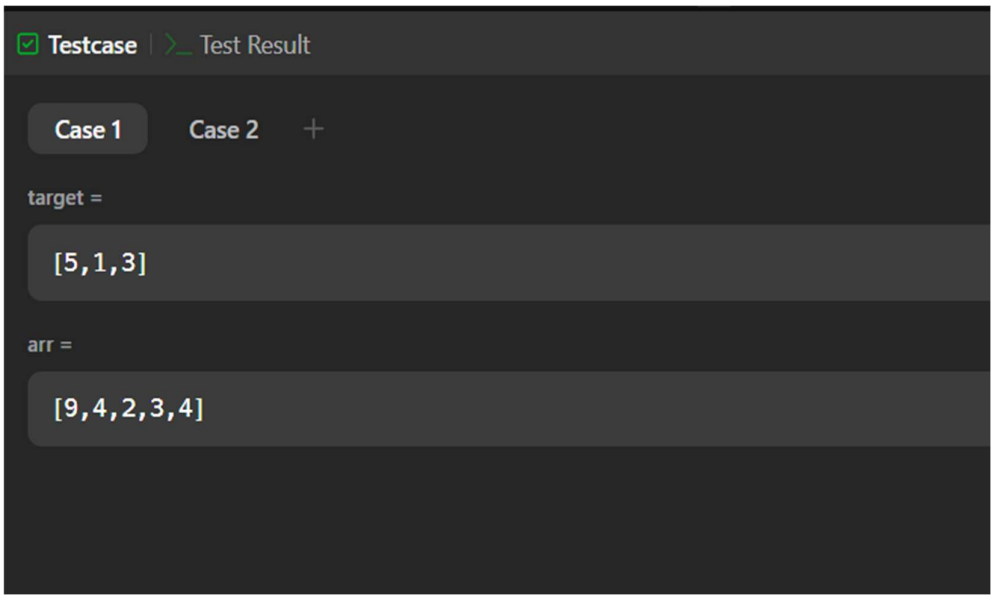
Minimum Stone Sum After k Operations:

```
import java.util.Stack;

class Solution {
  public int maximumGain(String s, int x, int y) {
    // Determine which pair to remove first for
    maximum points char first = x >= y ? 'a' : 'b';
    char second = x >= y ? 'b' :
    'a'; int firstPoints =
    Math.max(x, y);
    int secondPoints = Math.min(x, y);

    Stack<Character> stack = new
    Stack<>(); int maxPoints = 0;

    // Remove the more valuable
    substring first StringBuilder temp =
    new StringBuilder(); for (char c :
    s.toCharArray()) {

      if (!stack.isEmpty() && stack.peek() == first && c ==
        second) { stack.pop();
        maxPoints += firstPoints;
      } else {
        stack.push(c);
      }
    }

    // Extract the remaining string from the
    stack while (!stack.isEmpty()) {
      temp.append(stack.pop());
    }
    temp.reverse(); // Reverse to maintain order

    // Remove the second valuable substring
    for (char c : temp.toString().toCharArray()) {
      if (!stack.isEmpty() && stack.peek() == second && c ==
        first) { stack.pop();
```

```
                maxPoints += secondPoints;
            } else {
                stack.push(c);
            }
        }

        return maxPoints;
    }
}
```

6. Output:





7. Learning Outcomes:

- Understand how greedy choices impact overall optimization.
- Learn how ordering of operations affects final result.
- Understand stack-based substring elimination for optimal performance.
- Optimize character storage while performing iterative operations.