**PRACTICAL NO. 4**

**Name –**  Ishwari Deshmukh

**Class –** A4 (B3)

**Roll no. –** 45

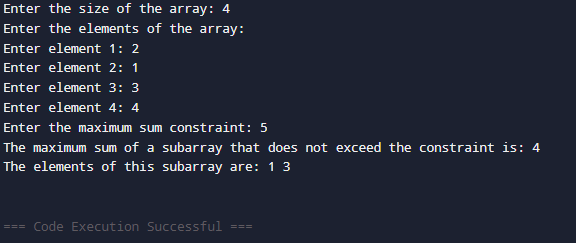
**Aim:** Implement maximum sum of subarray for the given scenario of resource allocation using the divide and conquer approach.

# Problem Statement:

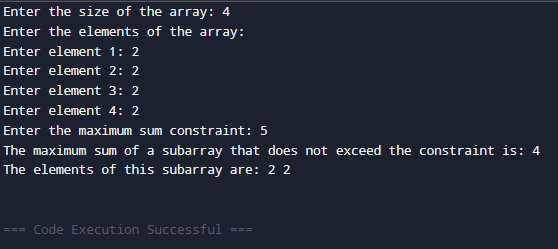
A project requires allocating resources to various tasks over a period of time. Each task requires a certain amount of resources, and you want to maximize the overall efficiency of resource usage. You're given an array resources where resources[i] represents the amount of resources required for the ith task. Your goal is to find the contiguous subarray of tasks that maximizes the total resources utilized without exceeding a given resource constraint.

Handle cases where the total resources exceed the constraint by adjusting the subarray window accordingly. Your implementation should handle various cases, including scenarios where there's no feasible subarray given the constraint and scenarios where multiple subarrays yield the same maximum resource utilization.

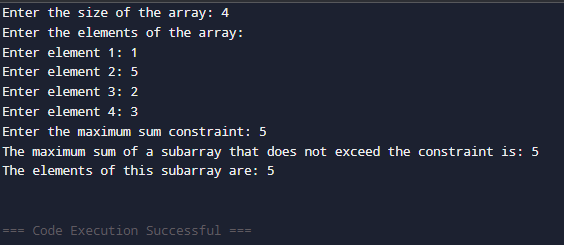
# Basic small array

* + resources = [2, 1, 3, 4], constraint = 5
    - Best subarray: [2, 1] or [1, 3] → sum = 4
    - Checks simple working.
    - 

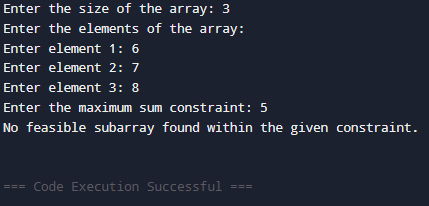
# Exact match to constraint

* + resources = [2, 2, 2, 2], constraint = 4
    - Best subarray: [2, 2] → sum = 4
    - Tests exact utilization.
    - 

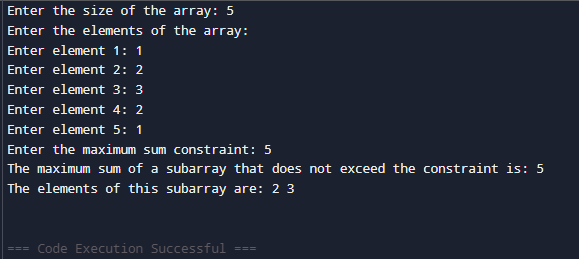
# Single element equals constraint

* + resources = [1, 5, 2, 3], constraint = 5
    - Best subarray: [5] → sum = 5
    - Tests one-element solution.
    - 

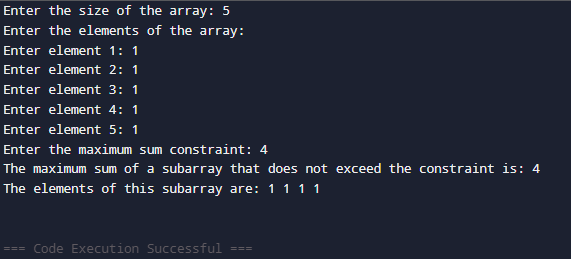
# All elements smaller but no combination fits

* + resources = [6, 7, 8], constraint = 5
    - No feasible subarray.
    - Tests "no solution" case.
    - 

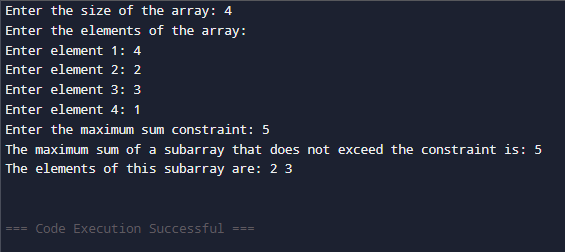
# Multiple optimal subarrays

* + resources = [1, 2, 3, 2, 1], constraint = 5
    - Best subarrays: [2, 3] and [3, 2] → sum = 5
    - Tests tie-breaking (should return either valid subarray).
    - 

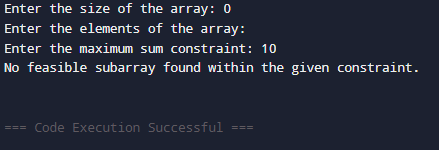
# Large window valid

* + resources = [1, 1, 1, 1, 1], constraint = 4
    - Best subarray: [1, 1, 1, 1] → sum = 4
    - Ensures long window works.
    - 

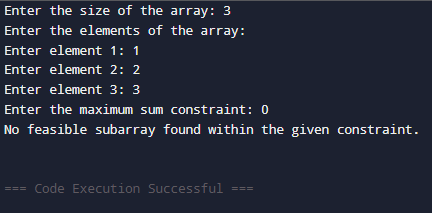
# Sliding window shrink needed

* + resources = [4, 2, 3, 1], constraint = 5
    - Start [4,2] = 6 (too big) → shrink to [2,3] = 5.
    - Tests dynamic window adjustment.
    - 

# Empty array

* + resources = [], constraint = 10
    - Output: no subarray.
    - Edge case: empty input.
    - 

# Constraint = 0

* + resources = [1, 2, 3], constraint = 0
    - No subarray possible.
    - Edge case: zero constraint.
    - 

# Very large input (stress test)

* + resources = [1, 2, 3, ..., 100000], constraint = 10^9
    - Valid subarray near full array.
    - Performance test.
    - 