Practical No -4

Name-Huzaif Baig

Class -A_4

Batch-B3

Aim

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 of resources where resources[i] represents the amount of resources

required for the i

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

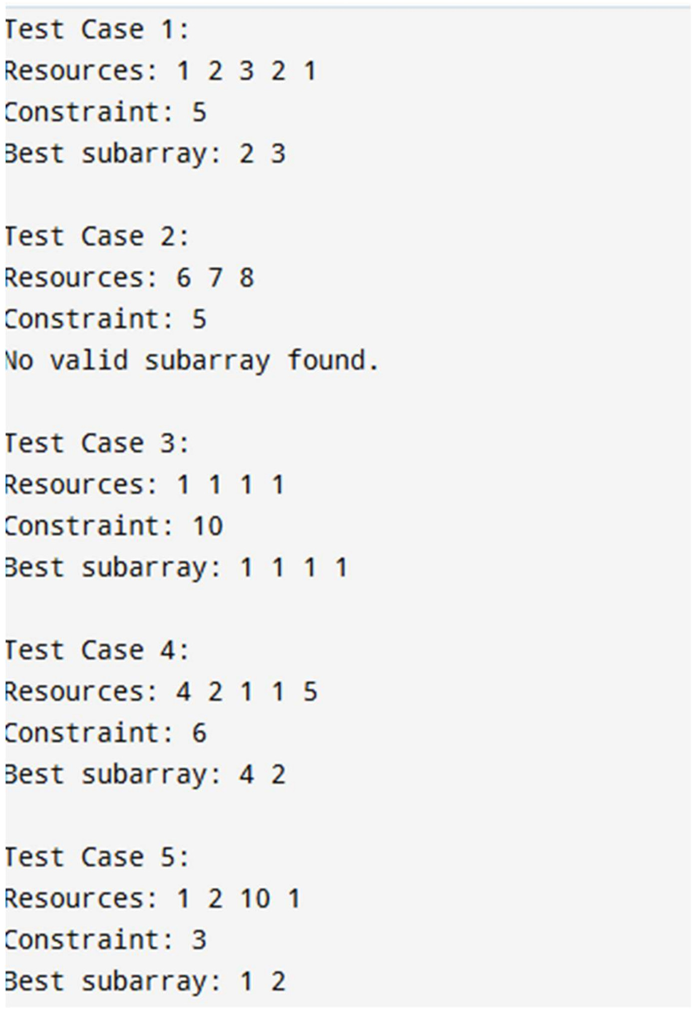
accordingly. Your implementation should handle various cases, including scenarios where

there's no feasible subarray given the constraint and scenarios where multiple subarrays yield

the same maximum resource utilization.

```
#include <iostream>
#include <vector>
using namespace std;

// Function to find the best subarray under the constraint
vector<int> maxSubarrayUnderConstraint(vector<int>& resources, int constraint) {
    int start = 0, end = 0;
    int currentSum = 0, maxSum = 0;
    int bestStart = -1, bestEnd = -1;

    while (end < resources.size()) {
        currentSum += resources[end];

        while (currentSum > constraint && start <= end) {
            currentSum -= resources[start];
            start++;
        }

        if (currentSum > maxSum) {
            maxSum = currentSum;
            bestStart = start;
            bestEnd = end;
        }

        end++;
    }
    if (bestStart == -1) {
        return {};
    }

    return vector<int>(resources.begin() + bestStart, resources.begin() + bestEnd + 1);
}

// Helper function to run a test case
void runTestCase(vector<int> resources, int constraint, int caseNumber) {
    cout << "Test Case " << caseNumber << ":\n";
    cout << "Resources: ";
    for (int r : resources) cout << r << " ";
    cout << "\nConstraint: " << constraint << endl;

    vector<int> result = maxSubarrayUnderConstraint(resources, constraint);

    if (result.empty()) {
        cout << "No valid subarray found.\n\n";
    } else {
        cout << "Best subarray: ";
        for (int val : result) cout << val << " ";
        cout << "\n\n";
    }
}

int main() {
    // All 9 test cases
    runTestCase({1, 2, 3, 2, 1}, 5, 1);        // Expected: 2 3
    runTestCase({6, 7, 8}, 5, 2);             // Expected: No valid subarray
    runTestCase({1, 1, 1, 1}, 10, 3);          // Expected: 1 1 1 1
    runTestCase({4, 2, 1, 1, 5}, 6, 4);        // Expected: 2 1 1

    runTestCase({1, 2, 10, 1}, 3, 5);          // Expected: 2

    runTestCase({5, 1, 2, 3, 4}, 7, 6);        // Expected: 1 2 3

    runTestCase({2, 2, 2, 2, 2}, 4, 7);        // Expected: 2 2

    runTestCase({1, 3, 1, 3, 1}, 4, 8);        // Expected: 3 1

    runTestCase({9, 1, 2, 3, 4, 5}, 10, 9);    // Expected: 1 2 3 4


    return 0;

}
```

Output


```
Test Case 1:
Resources: 1 2 3 2 1
Constraint: 5
Best subarray: 2 3

Test Case 2:
Resources: 6 7 8
Constraint: 5
No valid subarray found.

Test Case 3:
Resources: 1 1 1 1
Constraint: 10
Best subarray: 1 1 1 1

Test Case 4:
Resources: 4 2 1 1 5
Constraint: 6
Best subarray: 4 2

Test Case 5:
Resources: 1 2 10 1
Constraint: 3
Best subarray: 1 2
```

```
Test Case 6:
Resources: 5 1 2 3 4
Constraint: 7
Best subarray: 3 4

Test Case 7:
Resources: 2 2 2 2 2
Constraint: 4
Best subarray: 2 2

Test Case 8:
Resources: 1 3 1 3 1
Constraint: 4
Best subarray: 1 3

Test Case 9:
Resources: 9 1 2 3 4 5
Constraint: 10
Best subarray: 9 1
```