RBU, Nagpur
CSE III Sem
PRACTICAL NO. 4

Name: Kishan Saini
Sec-Batch-Rollno: A4-B4-55

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 th 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.

Code:

```
#include <stdio.h>

struct Result {
   int start;
   int end;
   int sum;
};

struct Result maxSubArrayWithConstraint(int n, int arr[n], int constraint) {
   int maxSum = -1;
   int currentSum = 0;
   int start = 0, end = 0;
   int tempStart = 0;
   int foundValidSubarray = 0;

   for (int i = 0; i < n; i++) {
      currentSum += arr[i];

      while (currentSum > constraint && tempStart <= i) {
```

```
            currentSum -= arr[tempStart];
            tempStart++;
        }

        if (currentSum > maxSum && currentSum <= constraint) {
            maxSum = currentSum;
            start = tempStart;
            end = i;
            foundValidSubarray = 1;
        }
    }

    struct Result res;
    if (foundValidSubarray) {
        res.sum = maxSum;
        res.start = start;
        res.end = end;
    } else {
        res.sum = -1;
        res.start = -1;
        res.end = -1;
    }

    return res;
}

void runTest(int testNumber, int n, int arr[], int constraint, const char* description) {
    printf("--- Test Case %d: %s ---\n", testNumber, description);

    struct Result result = maxSubArrayWithConstraint(n, arr, constraint);

    printf("Resources: [");
    for (int i = 0; i < n; i++) {
        printf("%d", arr[i]);
        if (i < n - 1) {
            printf(", ");
        }
    }
    printf("], Constraint: %d\n", constraint);

    if (result.sum == -1) {
        printf("Result: No feasible subarray found.\n");
    } else {
        printf("Result: Best subarray found with sum %d, from index %d to %d.\n",
```

```
            result.sum, result.start, result.end);
    }
    printf("\n");
}

int main() {
    int arr1[] = {2, 1, 3, 4};
    runTest(1, 4, arr1, 5, "Basic small array");

    int arr2[] = {2, 2, 2, 2};
    runTest(2, 4, arr2, 4, "Exact match to constraint");

    int arr3[] = {1, 5, 2, 3};
    runTest(3, 4, arr3, 5, "Single element equals constraint");

    int arr4[] = {6, 7, 8};
    runTest(4, 3, arr4, 5, "No feasible subarray");

    int arr5[] = {1, 2, 3, 2, 1};
    runTest(5, 5, arr5, 5, "Multiple optimal subarrays");

    return 0;
}
```

Output:

```
--- Test Case 1: Basic small array ---
Resources: [2, 1, 3, 4], Constraint: 5
Result: Best subarray found with sum 4, from index 1 to 2.

--- Test Case 2: Exact match to constraint ---
Resources: [2, 2, 2, 2], Constraint: 4
Result: Best subarray found with sum 4, from index 0 to 1.

--- Test Case 3: Single element equals constraint ---
```

```
--- Test Case 3: Single element equals constraint ---
Resources: [1, 5, 2, 3], Constraint: 5
Result: Best subarray found with sum 5, from index 1 to 1.

--- Test Case 4: No feasible subarray ---
Resources: [6, 7, 8], Constraint: 5
Result: Best subarray found with sum 0, from index 1 to 0.
```

```
--- Test Case 5: Multiple optimal subarrays ---
Resources: [1, 2, 3, 2, 1], Constraint: 5
Result: Best subarray found with sum 5, from index 1 to 2.
```