NAME-RUDRA KALE

A8-B4-64
PRACTICAL-4
DAALAB

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 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>


typedef struct {
    int sum;
    int start;
    int end;
} Subarray;


Subarray Max_Sum_Subarray(int arr[], int low, int high, int constraint) {
    Subarray no_solution = {0, -1, -1};


    if (low > high) return no_solution;
```

```
    if (low == high) {
        if (arr[low] <= constraint) {
            return (Subarray){arr[low], low, high};
        } else {
            return no_solution;
        }
    }

    int mid = (low + high) / 2;

    Subarray left = Max_Sum_Subarray(arr, low, mid, constraint);
    Subarray right = Max_Sum_Subarray(arr, mid + 1, high, constraint);

    int sum = 0;
    int left_sum = 0;
    int left_start = mid;
    for (int i = mid; i >= low; i--) {
        sum += arr[i];
```

```
        if (sum <= constraint && sum > left_sum) {
            left_sum = sum;
            left_start = i;
        }
    }

    sum = 0;
    int right_sum = 0;
    int right_end = mid + 1;
    for (int j = mid + 1; j <= high; j++) {
        sum += arr[j];
        if (sum <= constraint && sum > right_sum)
{
            right_sum = sum;
            right_end = j;
        }
    }

    int total_sum = left_sum + right_sum;
```

```
    Subarray cross = {total_sum, left_start, right_end};
    if (total_sum > constraint) {
        cross.sum = 0;
    }

    if (left.sum >= right.sum && left.sum >= cross.sum) {
        return left;
    } else if (right.sum >= left.sum && right.sum >= cross.sum) {
        return right;
    } else {
        return cross;
    }
}

int main() {
    int n;
```

```
    printf("Enter size of Resource Array: ");
    scanf("%d", &n);

    if (n <= 0) {
        printf("No resources in the array!\n");
        return 0;
    }

    int arr[n];
    printf("Enter the Resource Array values:\n");
    for (int i = 0; i < n; i++) {
        printf("Resource %d: ", i + 1);
        scanf("%d", &arr[i]);
    }

    int constraint;
    printf("Enter the resource constraint: ");
    scanf("%d", &constraint);
```

```
    Subarray result = Max_Sum_Subarray(arr, 0,
n - 1, constraint);

    if (result.sum == 0) {
        printf("No valid subarray found!\n");
    } else {
        printf("Max sum = %d, from index %d to
%d\n", result.sum, result.start, result.end);
    }

    return 0;
}
```

OUTPUTS ACC TO QUESTIONS

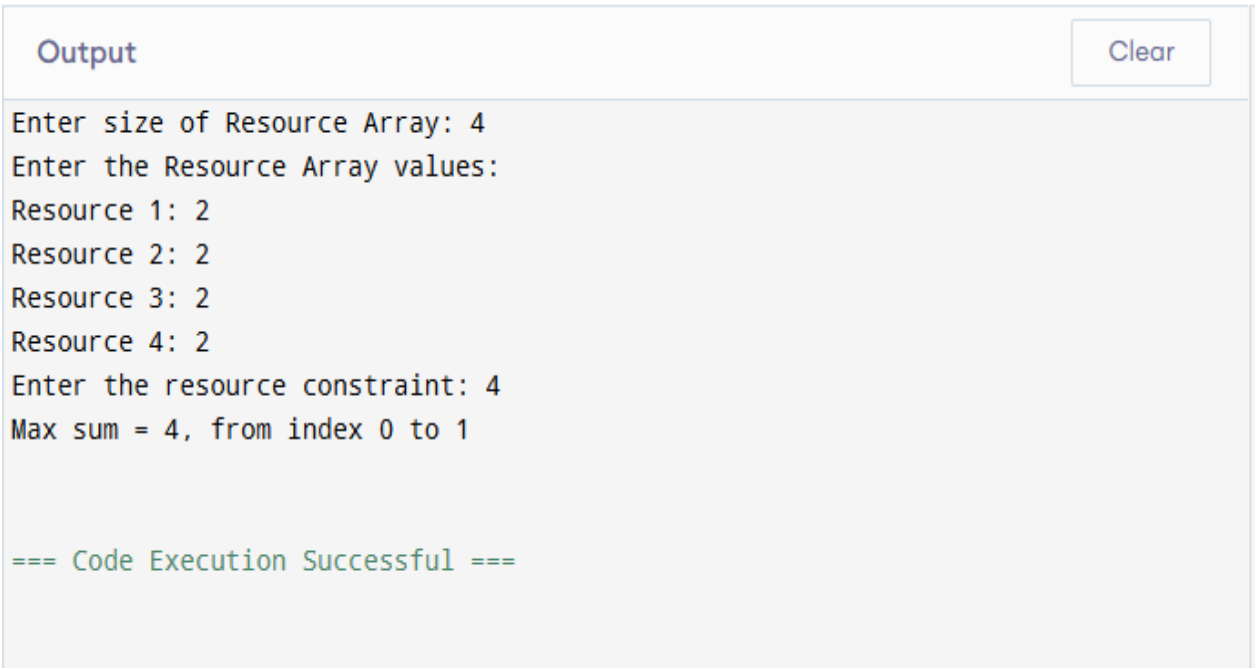
1. **Basic small array**

Output :

---

2. Exact match to constraint

Output :



3. Single element equals constraint

Output :

Output Clear

```
Enter size of Resource Array: 4
Enter the Resource Array values:
Resource 1: 1
Resource 2: 5
Resource 3: 2
Resource 4: 3
Enter the resource constraint: 5
Max sum = 5, from index 1 to 1


=== Code Execution Successful ===
```

4. All elements smaller but no combination fits

Output :

Output Clear

```
Enter size of Resource Array: 3
Enter the Resource Array values:
Resource 1: 6
Resource 2: 7
Resource 3: 8
Enter the resource constraint: 5
No valid subarray found!


=== Code Execution Successful ===
```

5. Multiple optimal subarrays

Output :

```
Output                                                    Clear

Enter size of Resource Array: 5
Enter the Resource Array values:
Resource 1: 1
Resource 2: 2
Resource 3: 3
Resource 4: 2
Resource 5: 1
Enter the resource constraint: 5
Max sum = 3, from index 0 to 1


=== Code Execution Successful ===
```

---

6. Large window valid

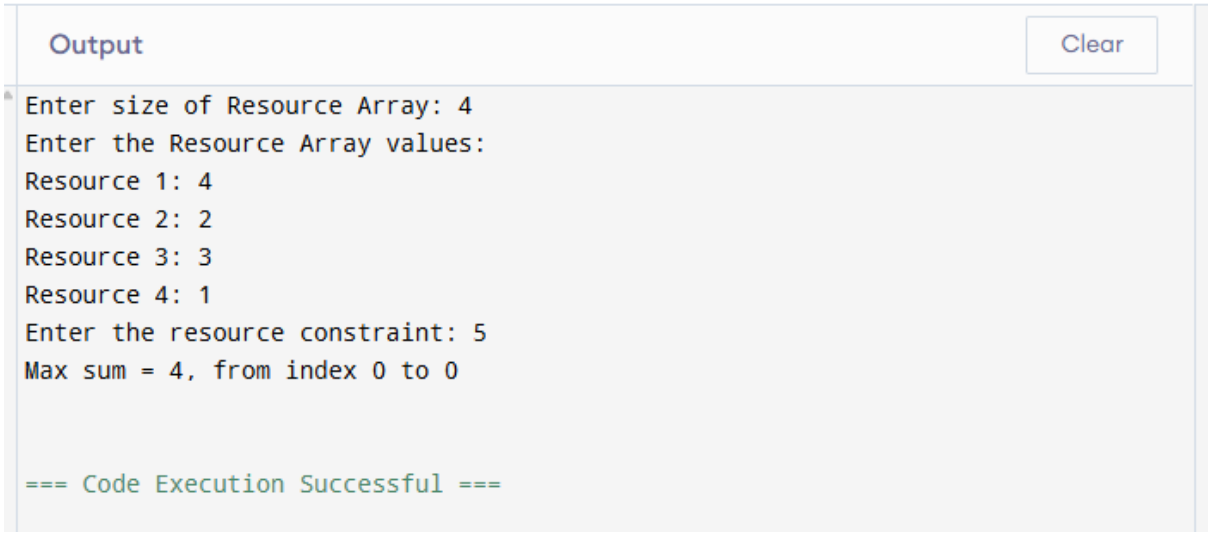
Output :

```
Output                                                    Clear

Enter size of Resource Array: 4
Enter the Resource Array values:
Resource 1: 1
Resource 2: 1
Resource 3: 1
Resource 4: 1
Enter the resource constraint: 4
Max sum = 4, from index 0 to 3


=== Code Execution Successful ===
```

---

7. Sliding window shrink needed

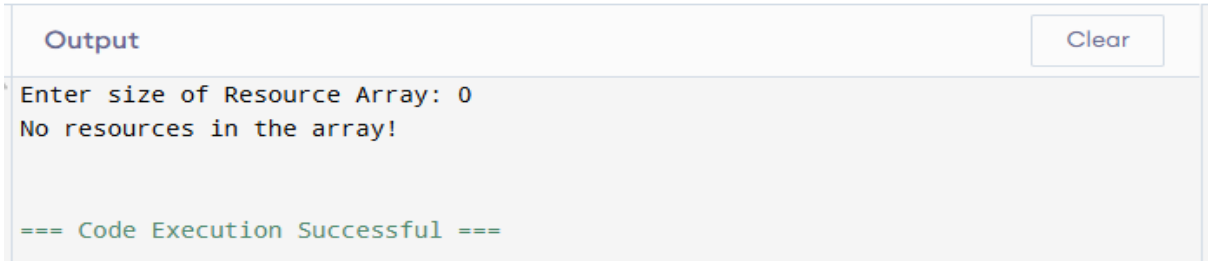
Output :



---

8. Empty array

Output :



---

9. Constraint = 0

Output :

Output Clear

```
Enter size of Resource Array: 3
Enter the Resource Array values:
Resource 1: 1
Resource 2: 2
Resource 3: 3
Enter the resource constraint: 0
No valid subarray found!


=== Code Execution Successful ===
```