DESIGN AND ANALYSIS OF ALGORITHMS
LAB III SEMESTER

Practical No. : 04

Name : Dipansh Tiwari

Section : A4_B3

Roll Num : 35

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

CODE :

```c
#include <stdio.h>

int SumSubarray(int *nums, int n, int con)
{
    for (int i = 0; i < n; i++)
    {
        int sum = 0;
        for (int j = i; j < n; j++)
        {
            sum += nums[j];
            if (sum == con)
            {
                printf("Subarray found: [");
                for (int k = i; k <= j; k++)
                {
                    printf("%d", nums[k]);
                    if (k < j)
                        printf(", ");
                }
                printf("]\n");
                return 1;
            }
        }
    }
    return 0;
    }

    int main()
    {
```

```
int n;
printf("Enter number of elements: ");
scanf("%d", &n);

if (n >= 100)
{
printf("I cannot Give input for these many elements"); return 0;
}

int arr[n];
printf("Enter %d elements: ", n);
for (int i = 0; i < n; i++)
{
scanf("%d", &arr[i]);
}

int con;
printf("Enter constraint: ");
scanf("%d", &con);


if (!SumSubarray(arr, n, con))
{
printf("No subarray with sum = %d found\n", con); }

return 0;
}
```

TEST CASES :

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

Output

```
Enter number of elements: 4
Enter 0 elements: 2
Enter 1 elements: 1
Enter 2 elements: 3
Enter 3 elements: 4
Enter constraint: 5
No subarray with sum = 5 found
```

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

Output

```
Enter number of elements: 4
Enter 0 elements: 2
Enter 1 elements: 2
Enter 2 elements: 2
Enter 3 elements: 2
Enter constraint: 4
Subarray found: [2, 2]
```

3. Single element equals constraint

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

Output

```
Enter number of elements: 4
Enter 0 elements: 1
Enter 1 elements: 5
Enter 2 elements: 2
Enter 3 elements: 3
Enter constraint: 5
Subarray found: [5]
```

4. All elements smaller but no combination fits

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

Output

```
Enter number of elements: 3
Enter 0 elements: 6
Enter 1 elements: 7
Enter 2 elements: 8
Enter constraint: 5
No subarray with sum = 5 found
```

---

5. 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).

Output

```
Enter number of elements: 5
Enter 0 elements: 1
Enter 1 elements: 2
Enter 2 elements: 3
Enter 3 elements: 2
Enter 4 elements: 1
Enter constraint: 5
Subarray found: [2, 3]
```

6. Large window valid

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

Output

```
Enter number of elements: 4
Enter 0 elements: 1
Enter 1 elements: 1
Enter 2 elements: 1
Enter 3 elements: 1
Enter constraint: 4
Subarray found: [1, 1, 1, 1]
```

7. 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.

Output

```
Enter number of elements: 4
Enter 0 elements: 4
Enter 1 elements: 2
Enter 2 elements: 3
Enter 3 elements: 1
Enter constraint: 5
Subarray found: [2, 3]
```

Output

```
Enter number of elements: 0
Enter constraint: 10
No subarray with sum = 10 found
```