Name: Areeb Haashir Roll No:A3-B1-01

Practical 4

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>

#include<limits.h>

int maxCrossingSum(int arr[], int l, int m, int h, int constraint, int n) {
```

```
int leftSum[n],
rightSum[n];
int sum=0;
int leftCount=0,
rightCount=0;

for (int i=m; i >=l; i--) {
   sum+=arr[i];

   if (sum <=constraint) {
      leftSum[leftCount++]=sum;
   }
}

sum=0;

for (int i=m + 1; i <=h; i++) {
   sum+=arr[i];

   if (sum <=constraint) {
      rightSum[rightCount++]=sum;
   }
}

int best=INT_MIN;

for (int i=0; i < leftCount; i++) {
   if (leftSum[i] > best) best=leftSum[i];
```

```
    for (int j=0; j < rightCount; j++) {
      int total=leftSum[i]+rightSum[j];

      if (total <=constraint && total > best) {
        best=total;
      }
    }
  }

  for (int j=0; j < rightCount; j++) {
    if (rightSum[j] > best) best=rightSum[j];
  }

  return best;
}

int maxSubArray(int arr[], int l, int h, int constraint, int n) {
  if (l > h) {
    return INT_MIN;
  }

  if (l==h) {
    if (arr[l] <=constraint) return arr[l];
    else return INT_MIN;
  }

  int mid=(l + h) / 2;
```

```
    int left=maxSubArray(arr, l, mid, constraint, n);
    int right=maxSubArray(arr, mid + 1, h, constraint, n);
    int cross=maxCrossingSum(arr, l, mid, h, constraint, n);
    int max=left;
    if (right > max) max=right;
    if (cross > max) max=cross;
    return max;
}

int main() {
    int n,
    constraint;
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements of the array:\n");

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

    printf("Enter the resource constraint: ");
    scanf("%d", &constraint);
    int ans=maxSubArray(arr, 0, n - 1, constraint, n);

    if (ans==INT_MIN) {
        printf("No feasible subarray found.\n");
    }
```

```
    else {
        printf("Maximum Sum is %d\n", ans);
    }

    return 0;
}
```

1. Basic small array

• resources = [2, 1, 3, 4], constraint = 5

o Best subarray: [2, 1] or [1, 3] → sum = 4

o Checks simple working.

2. Exact match to constraint

• resources = [2, 2, 2, 2], constraint = 4

o Best subarray: [2, 2] → sum = 4

o Tests exact utilization.

3. Single element equals constraint

• resources = [1, 5, 2, 3], constraint = 5

o Best subarray: [5] → sum = 5

o Tests one-element solution.

4. All elements smaller but no combination fits

• resources = [6, 7, 8], constraint = 5

o No feasible subarray.

o Tests "no solution" case.

5. Multiple optimal subarrays

• resources = [1, 2, 3, 2, 1], constraint = 5

o Best subarrays: [2, 3] and [3, 2] → sum = 5

o Tests tie-breaking (should return either valid subarray).

6. Large window valid

• resources = [1, 1, 1, 1, 1], constraint = 4

o Best subarray: [1, 1, 1, 1] → sum = 4

o Ensures long window works.

7. Sliding window shrink needed

• resources = [4, 2, 3, 1], constraint = 5

o Start [4,2] = 6 (too big) → shrink to [2,3] = 5.

o Tests dynamic window adjustment.

8. Empty array

• resources = [], constraint = 10

o Output: no subarray.

o Edge case: empty input.

9. Constraint = 0

• resources = [1, 2, 3], constraint = 0

o No subarray possible.

o Edge case: zero constraint.