Name- Marmika Kirnapure

Section- A7 Batch- B2

Rollno- 31

DAA 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&#39;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&#39;s no feasible subarray given the constraint and scenarios where multiple

subarrays yield the same maximum resource utilization.

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 &quot;no solution&quot; 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.

10. Very large input (stress test)

resources = [1, 2, 3, ..., 100000], constraint = 10^9

o Valid subarray near full array.

o Performance test.

**CODE-**

#include <stdio.h>

void maxSubarraySum(int resources[], int n, int constraint) {

int left = 0, right = 0, currentSum = 0;

int maxSum = -1;

int bestLeft = 0, bestRight = 0;

while (right < n) {

currentSum += resources[right];

while (currentSum > constraint && left <= right) {

currentSum -= resources[left++];

}

if (currentSum <= constraint && currentSum > maxSum) {

maxSum = currentSum;

bestLeft = left;

bestRight = right;

}

right++;

}

if (maxSum >= 0) {

printf("Max Subarray Sum: %d\n", maxSum);

printf("Best Subarray: ");

for (int i = bestLeft; i <= bestRight; i++) {

printf("%d ", resources[i]);

}

printf("\n");

} else {

printf("No valid subarray\n");

}

}

int main() {

int resource1[] = {2, 1, 3, 4};

int n1 = sizeof(resource1) / sizeof(resource1[0]);

int constraint1 = 5;

maxSubarraySum(resource1, n1, constraint1);

int resource2[] = {2, 2, 2, 2};

int n2 = sizeof(resource2) / sizeof(resource2[0]);

int constraint2 = 4;

maxSubarraySum(resource2, n2, constraint2);

int resource3[] = {1, 5, 2, 3};

int n3 = sizeof(resource3) / sizeof(resource3[0]);

int constraint3 = 5;

maxSubarraySum(resource3, n3, constraint3);

int resource4[] = {6, 7, 8};

int n4 = sizeof(resource4) / sizeof(resource4[0]);

int constraint4 = 5;

maxSubarraySum(resource4, n4, constraint4);

int resource5[] = {1, 2, 3, 2, 1};

int n5 = sizeof(resource5) / sizeof(resource5[0]);

int constraint5 = 5;

maxSubarraySum(resource5, n5, constraint5);

int resource6[] = {1, 1, 1, 1, 1};

int n6 = sizeof(resource6) / sizeof(resource6[0]);

int constraint6 = 4;

maxSubarraySum(resource6, n6, constraint6);

int \*resource7 = NULL;

int n7 = 0;

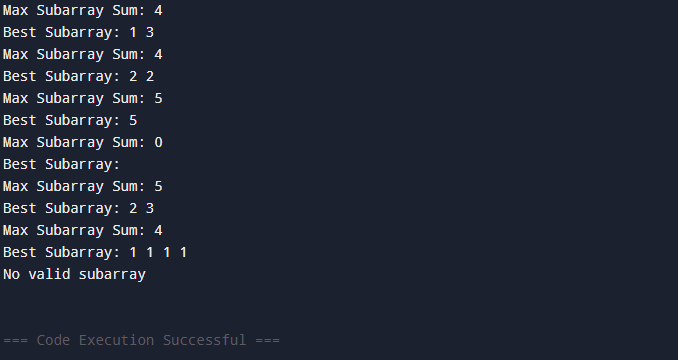
int constraint7 = 4;

maxSubarraySum(resource7, n7, constraint7);

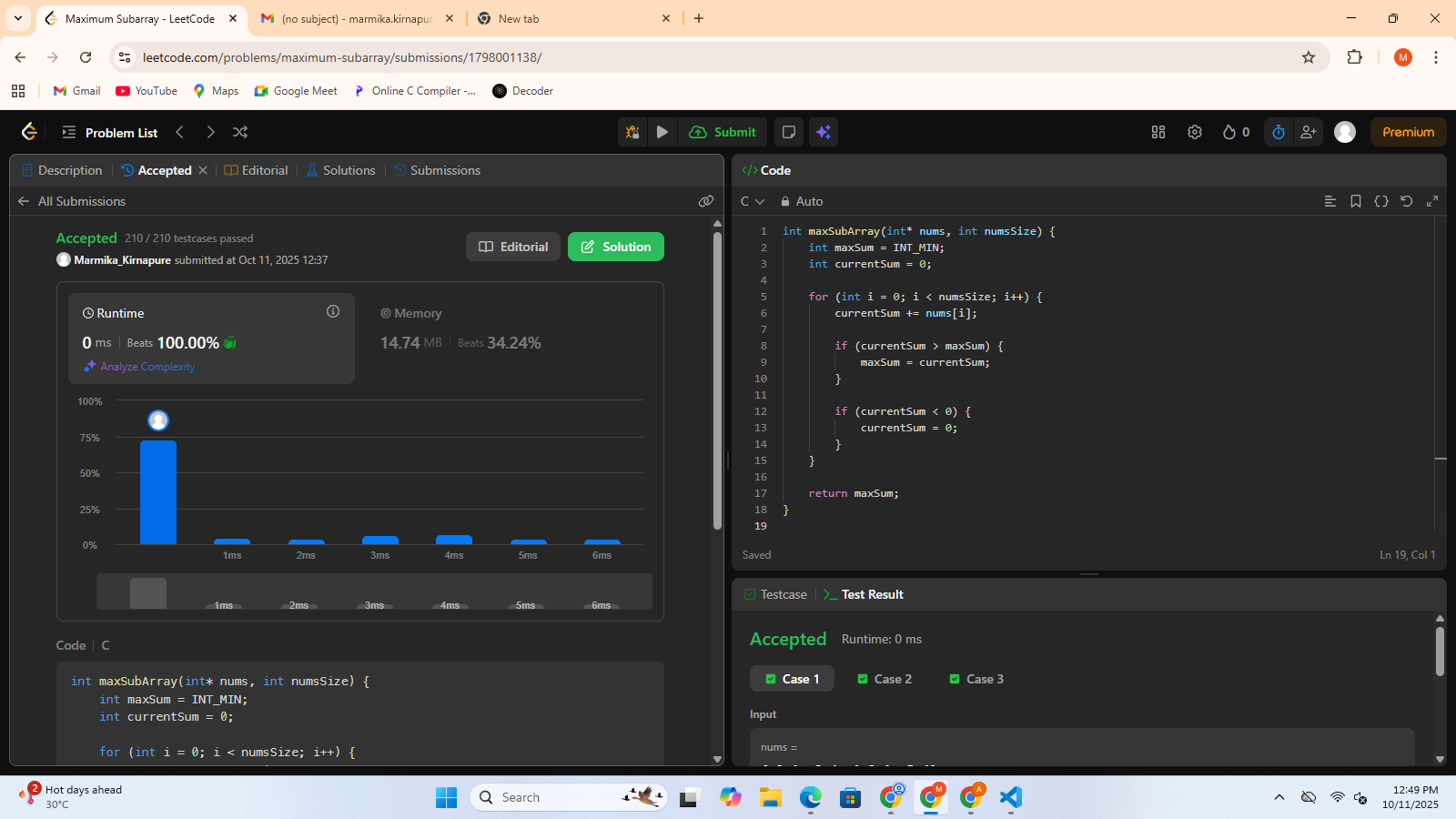
return 0;

}

**OUTPUT-**

****

**LEETCODE-**

****