DAA PRACTICAL 4

NAME : MAHI BHOYAR

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.ROLL NO : A5-B1-14

TASK-1:

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.

10. Very large input (stress test)

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

o Valid subarray near full array.

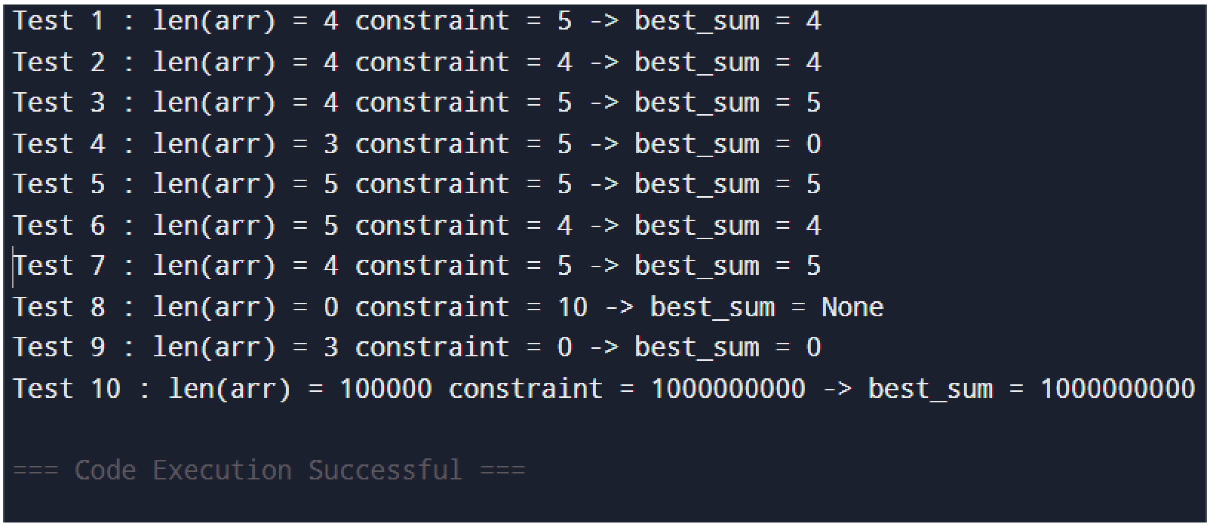
o Performance test.

CODE

```
def max_subarray_with_constraint(arr, constraint):
    n = len(arr)
    if n == 0:
        return None
    best_sum = None
    current_sum = 0
    left = 0

    for right in range(n):
        current_sum += arr[right]

        # shrink window while it violates constraint
        while current_sum > constraint and left <= right:
            current_sum -= arr[left]
            left += 1

        # check valid window
        if current_sum <= constraint:
            if best_sum is None or current_sum > best_sum:
                best_sum = current_sum

    return best_sum
```

```
#  TEST CASES
test_cases = [
    ([2, 1, 3, 4], 5),          # 1. Basic small array
    ([2, 2, 2, 2], 4),          # 2. Exact match
    ([1, 5, 2, 3], 5),          # 3. Single element
    ([6, 7, 8], 5),             # 4. All larger
    ([1, 2, 3, 2, 1], 5),       # 5. Multiple optimal
    ([1, 1, 1, 1, 1], 4),       # 6. Large window valid
    ([4, 2, 3, 1], 5),          # 7. Shrink window needed
    ([], 10),                   # 8. Empty array
    ([1, 2, 3], 0),             # 9. Constraint = 0
    (list(range(1, 100001)), 10**9)  # 10. Stress test
]


i = 1
for arr, constraint in test_cases:
    result = max_subarray_with_constraint(arr, constraint)
    print("Test", i, ": len(arr) =", len(arr), "constraint =", constraint, "-> best_sum =", result)
    i += 1
```

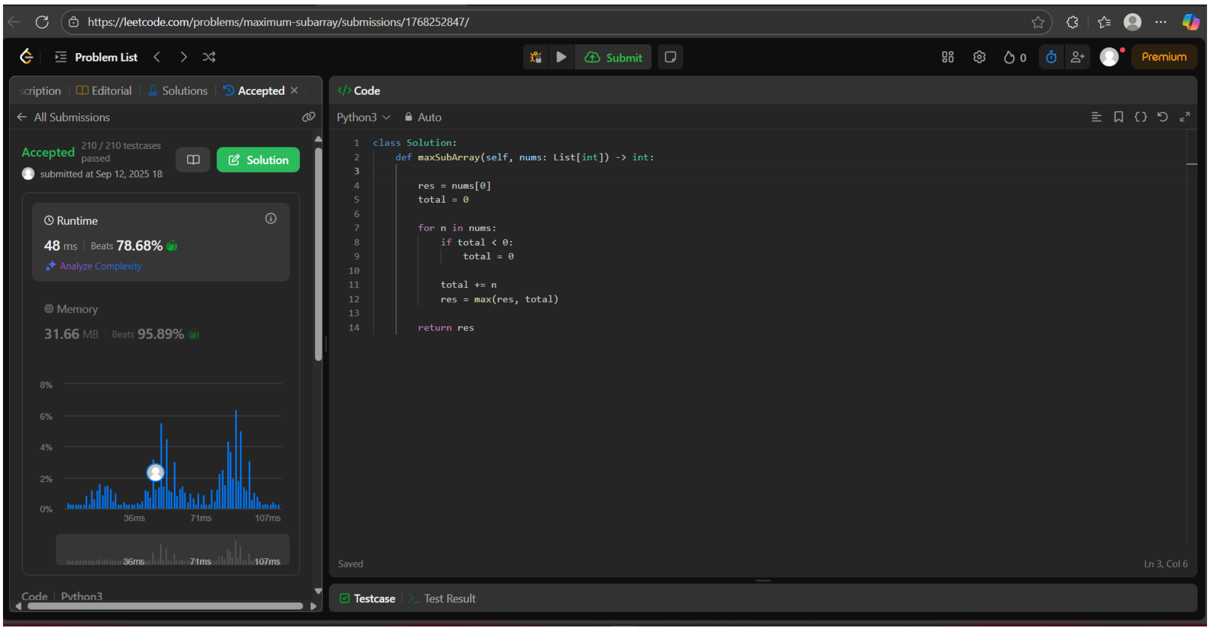
OUTPUT



```
Test 1 : len(arr) = 4 constraint = 5 -> best_sum = 4
Test 2 : len(arr) = 4 constraint = 4 -> best_sum = 4
Test 3 : len(arr) = 4 constraint = 5 -> best_sum = 5
Test 4 : len(arr) = 3 constraint = 5 -> best_sum = 0
Test 5 : len(arr) = 5 constraint = 5 -> best_sum = 5
Test 6 : len(arr) = 5 constraint = 4 -> best_sum = 4
Test 7 : len(arr) = 4 constraint = 5 -> best_sum = 5
Test 8 : len(arr) = 0 constraint = 10 -> best_sum = None
Test 9 : len(arr) = 3 constraint = 0 -> best_sum = 0
Test 10 : len(arr) = 100000 constraint = 1000000000 -> best_sum = 1000000000

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

TASK 2 : LEETCODE /HACKEREARTH SUBMISSION

LEETCODE



Accepted 210 / 210 testcases passed

submitted at Sep 12, 2025 18:

Runtime

48 ms | Beats 78.68%

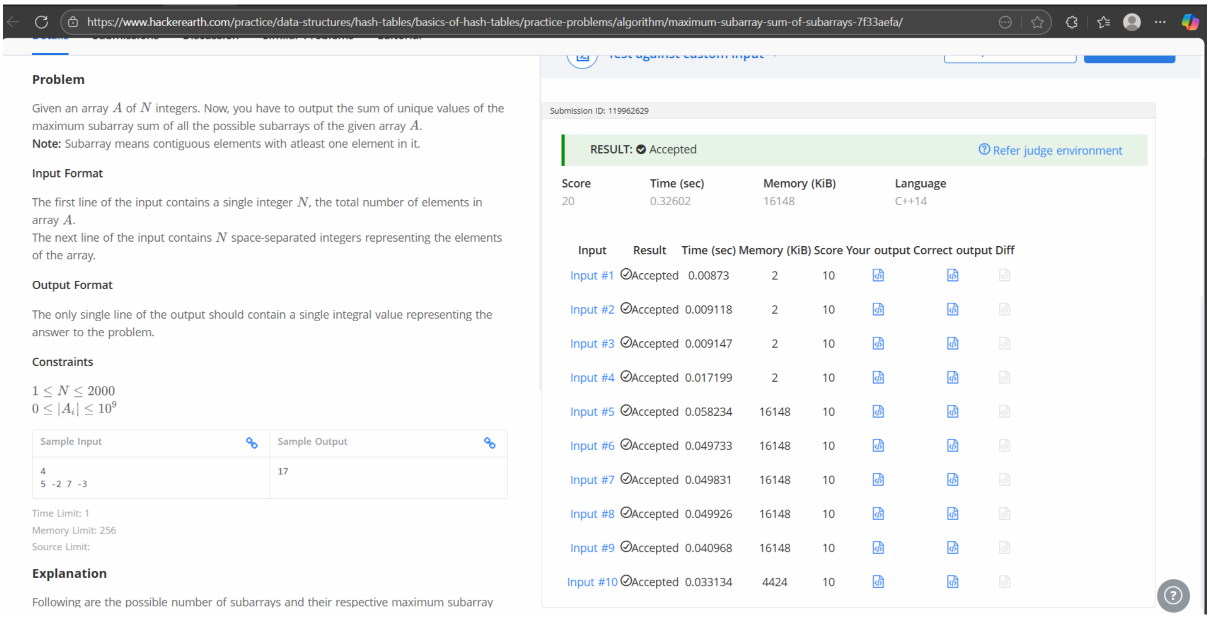
Analyze Complexity

Memory

31.66 MB | Beats 95.89%

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:

        res = nums[0]
        total = 0

        for n in nums:
            if total < 0:
                total = 0

            total += n
            res = max(res, total)

        return res
```

HACKERERATH



https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/practice-problems/algorithm/maximum-subarray-sum-of-subarrays-7f33aefa/

Problem

Given an array A of N integers. Now, you have to output the sum of unique values of the maximum subarray sum of all the possible subarrays of the given array A.
Note: Subarray means contiguous elements with atleast one element in it.

Input Format

The first line of the input contains a single integer N, the total number of elements in array A.
The next line of the input contains N space-separated integers representing the elements of the array.

Output Format

The only single line of the output should contain a single integral value representing the answer to the problem.

Constraints

$1 \leq N \leq 2000$
$0 \leq |A_i| \leq 10^9$

Sample Input	Sample Output
4 5 -2 7 -3	17

Time Limit: 1
Memory Limit: 256
Source Limit:

Explanation

Following are the possible number of subarrays and their respective maximum subarray

Submission ID: 119962629

RESULT: ✓ Accepted

ⓘ Refer judge environment

Score	Time (sec)	Memory (KiB)	Language
20	0.32602	16148	C++14

Input	Result	Time (sec)	Memory (KiB)	Score	Your output	Correct output	Diff
Input #1	✓Accepted	0.00873	2	10	📄	📄	📄
Input #2	✓Accepted	0.009118	2	10	📄	📄	📄
Input #3	✓Accepted	0.009147	2	10	📄	📄	📄
Input #4	✓Accepted	0.017199	2	10	📄	📄	📄
Input #5	✓Accepted	0.058234	16148	10	📄	📄	📄
Input #6	✓Accepted	0.049733	16148	10	📄	📄	📄
Input #7	✓Accepted	0.049831	16148	10	📄	📄	📄
Input #8	✓Accepted	0.049926	16148	10	📄	📄	📄
Input #9	✓Accepted	0.040968	16148	10	📄	📄	📄
Input #10	✓Accepted	0.033134	4424	10	📄	📄	📄