Name vansh sharma

Class A6-B3-45

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

def max\_sum\_subarray\_under\_constraint(arr, constraint):

def helper(left, right):

if left > right:

return float('-inf'), []

if left == right:

if arr[left] <= constraint:

return arr[left], [arr[left]]

else:

return float('-inf'), []

mid = (left + right) // 2

left\_sum, left\_sub = helper(left, mid)

right\_sum, right\_sub = helper(mid + 1, right)

max\_left\_sum = float('-inf')

temp\_sum = 0

cross\_left\_idx = mid

for i in range(mid, left - 1, -1):

temp\_sum += arr[i]

if temp\_sum > max\_left\_sum:

max\_left\_sum = temp\_sum

cross\_left\_idx = i

max\_right\_sum = float('-inf')

temp\_sum = 0

cross\_right\_idx = mid + 1

for j in range(mid + 1, right + 1):

temp\_sum += arr[j]

if temp\_sum > max\_right\_sum:

max\_right\_sum = temp\_sum

cross\_right\_idx = j

cross\_sum = max\_left\_sum + max\_right\_sum

if cross\_sum <= constraint:

cross\_sub = arr[cross\_left\_idx:cross\_right\_idx + 1]

else:

cross\_sum = float('-inf')

cross\_sub = []

candidates = [

(left\_sum, left\_sub),

(right\_sum, right\_sub),

(cross\_sum, cross\_sub)

]

valid = [(s, sub) for s, sub in candidates if s <= constraint]

if not valid:

return float('-inf'), []

return max(valid, key=lambda x: x[0])

max\_sum, subarray = helper(0, len(arr) - 1)

if max\_sum == float('-inf'):

return 0, []

return max\_sum, subarray

arr = [4, -2, 3, -1, 5]

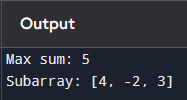
constraint = 7

max\_sum, subarray = max\_sum\_subarray\_under\_constraint(arr, constraint)

print("Max sum:", max\_sum)

print("Subarray:", subarray)

output



TASK-2: Submission on LEETCODE/HACKEREARTH

