ITCS 6114: Finding a Maximum Subsequence Sum through Divide-and-Conquer

The **goal** of the programming task is to practice the divide-and-conquer technique and review the time complexity analysis of recurrences.

Please develop a divide-and-conquer solution of determining a maximum sum of certain subsets of one-dimensional arrays in the running time of O(n lg n). Please also perform the time complexity analysis.

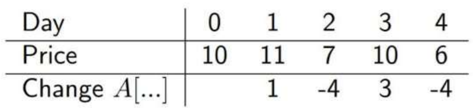
Problem Statement
Maximum Subsequence Sum problem (also called maximum subarrays problem): given an array *A* with signed integer elements, find a contiguous subarray with a maximum possible sum.

Finding the Maximum Subsequence Sum

Given an array *A* of signed integers, design an algorithm that finds a subarray *A*[*i, …, j*] such that
A[*i*] + *A*[*i* + 1] + … + *A*[*j*] is maximum in time *O*(*n* log *n*). More specifically, find a pair (*i, j*) such that$\forall (x,y)\ \sum_{k=i}^{j} A[k] > \sum_{k'=i}^{j} A[k']$.

For example,

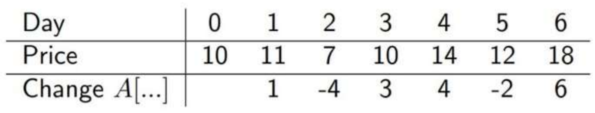
Example 1:



Day	0	1	2	3	4
Price	10	11	7	10	6
Change $A[...]$		1	-4	3	-4

maximum-subarray: $A[3]$ $(i = j = 3)$ and Sum $= 3$

Example 2:



Day	0	1	2	3	4	5	6
Price	10	11	7	10	14	12	18
Change $A[...]$		1	-4	3	4	-2	6

maximum-subarray: $A[3...6]$ $(i = 3, j = 6)$ and Sum $= 11$.

High-level Description:
The intuitive method is to scan all pairs, A[i] and A[j], sum up all elements from A[i] through A[j], and find a maximum sum. However, the solution takes the running time of O(n^2) .

To obtain a running time of $O(n \log n)$, we can apply the divide-and-conquer method.

Note the following properties of the Maximum Subsequence Sum (MSS). First, if the array elements are all nonnegative, then the MSS is given by the sum from $i = 1$ to $j = n$. Second, if $A[i] < 0\ \forall i$, then the MSS can be defined as the element closest to zero. Note that, by definition, we disallow the empty set as a valid input (or solution to) the Maximum Subsequence Sum problem. In addition, note that the MSS is not necessarily unique; there could be more than one subarray which achieves the maximum sum.

Let's split the sequence in half, recursively solve the problem on both halves and see what we can do. For example, imagine we split the sequence in the middle and we get the following answers:

$$\langle \text{——}\ L\ \text{——} \| \text{——}\ R\ \text{——} \rangle$$
$$\Downarrow$$
$$L = \underbrace{\langle\ \cdots\ \rangle}_{\text{mcss}=56} \qquad R = \underbrace{\langle\ \ldots\ \rangle}_{\text{mcss}=17}$$

The maximum may appear from the left half, right half, or a consecutive subsequence across the element in the middle. As for the last one, we can scan the left subsequence and right subsequence in order to find a left start which gives a maximum sub-sum in the left subsequence, ending in the middle element of the whole sequence, and find a right end which gives a maximum sub-sum in the right subsequence, starting from the middle element of the whole sequence. The procedure can be recursively carried out till the base case (only one element is left in each subsequence.

Divide-and-Conquer in details:
Generic problem: Find a maximum subarray of A[low...high]
Initial call: low = 1 and high = n
Divide-and-conquer strategy:
1) **Divide:** A[low...high] into two subarrays of as equal size as possible by finding the midpoint mid
2) **Conquer**:
    (a) finding maximum subarrays of A[low...mid] and A[mid + 1...high]
    (b) finding a max-subarray that crosses the midpoint
3) **Combine**: returning the max of the three

This strategy works because any subarray must either lie entirely in one side of midpoint or cross the midpoint.

Pseudo Code for your reference:
Find-MSS(*A, low, high*)

```
if $low = high$
    then return $(low, low, A[low])$
```

```
 3  $mid \leftarrow floor((low + high)/2)$
 4  $(i_l, j_l, T_l) \leftarrow$ Find-MSS($A, low, mid$)
 5  $(i_r, j_r, T_r) \leftarrow$ Find-MSS($A, mid + 1, high$)
 6  $(i_s, j_s, T_s) \leftarrow$ Max-Span($A, low, mid, high$)
 7  $T \leftarrow \max(T_l, T_s, T_r)$

 8  switch
 9     case $T = T_l$ :
10           return $(i_l, j_l, T_l)$
11     case $T = T_r$ :
12           return $(i_r, j_r, T_r)$
13     case $T = T_s$ :
14           return $(i_s, j_s, T_s)$
```

```
Max-Span (A, low, mid, high)
1  leftsum = -infty; sum = 0
2  for i = mid downto low   // Find max-subarray of A[i..mid]
3      sum = sum + A[i]
4      if sum > leftsum
5          leftsum = sum
6          maxleft = i
7  rightsum = -infty; sum = 0
8  for j = mid+1 to high      // Find max-subarray of A[mid+1..j]
9       sum = sum + A[j]
10      if sum > rightsum
11          rightsum = sum
12          maxright = j
13   return (maxleft,maxright,leftsum + rightsum)
 // Return the indices i and j and the sum of two subarrays
```

To analyze the time complexity, please use one of the following methods:

- Substitution method
- Recursive tree method
- Master method

Please submit your code and analysis through canvas.

Optional question: Can we design a divide-and-conquer solution with a linear running time?