Assignment 2 : Divide and Conquer

Deadline: 31/03/2021 11:55 PM

Instruction:

1. Make two copies of the c file named "max_subarray.c", one for task 1 and the other for task 2. Write your code in those files.
2. Avoid plagiarism. If you are found to adopt any unfair means you will get a straight 0.
3. You can not change or modify the structure of the existing code. Just complete the functions you are instructed to.
4. Zip the c files in a zip folder. Upload only the zip file in lms.
5. Deadline is 31/03/2021 11:55 PM.

Task 1: Maximum Sum Subarray Problem

Given an array, find the subarray that has the maximum sum. This problem is discussed in your theory classes. Nevertheless I am attaching the slide for your reference. You will the have to implement the divide and conquer based solution for this problem.

For example,
Input: A[0…3] = [1, -4, 3, 2]
Output: Maximum sum subarray A[2…3]. Sum = 5.
Another example,
Input: A[0…6] = [1, -4, 3, 4, -2, 6, -2]
Output: Maximum sum subarray A[2…5]. Sum = 11.

Pseudocode and explanation of the divide and conquer based solution for this problem is included in the slide. A solved example is shared in the last page for your convenience.

Task 2: Application in a real life problem

Suppose that you are investing in a company which you know the past stock prices and all you want is maximise your profit.

Naturally in this case what comes first to our mind is that either buying stocks at the lowest price or selling them at the highest price. For this we can first find the lowest and highest prices and then,

1. Start from the lowest price and go right until you find the highest price after the lowest price and calculate the profit.
2. Start from the highest price and go left until you find the lowest price before the highest price and calculate the profit.
3. Get the maximum of the above two.

But, this is not always the case. Let's see the following counter example.

Day	0	1	2	3	4
Price	10	11	7	10	6

In this example the highest and lowest prices are 11 and 6 respectively. So,

1. If we start from the lowest price(6) and go right until we find the highest price after the lowest price we won't find that. (6 is the stock price at the last day)
2. If we start from the highest price(11) and go left until we find the lowest price before the highest price we will find 10, and the profit would be 1
3. Comparing the above we will get 1 as the maximum profit.

But it is wrong! We can earn a profit of 3 if we look deeper.

Yes! buying stocks after day 2 at a price of 7 and selling them after day 3 at a price of 10 we would earn a profit of 3 per share.

From the above example you may understand that our approach fails.

So how to solve this problem and find the maximum profit? Well, for that we need to look into this problem in a slightly different way. What we need is to find a sequence of days over which the net change from the first day to the last day is maximum.

Let's transform the above array as we just mentioned. So we need to consider the daily change in price instead of considering the daily prices.

Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	−4	3	−4

We have to find the sequence of days over which the net change from the first day to the last day is maximum. This is where the maximum sum subarray problem occurs!

So, from Price array(input) calculate the Change array. You may assign -Infinity to Change[0]. Then calculate the maximum sum subarray of the Change array. Output should be Chage[3…3]. Sum = 3.

The index just before the starting index of the maximum sum subarray will be the buying date, ending index will be selling date, and sum would be maximum profit. So, you should buy the stock on day 2, sell on day 3 to gain maximum profit of 3.

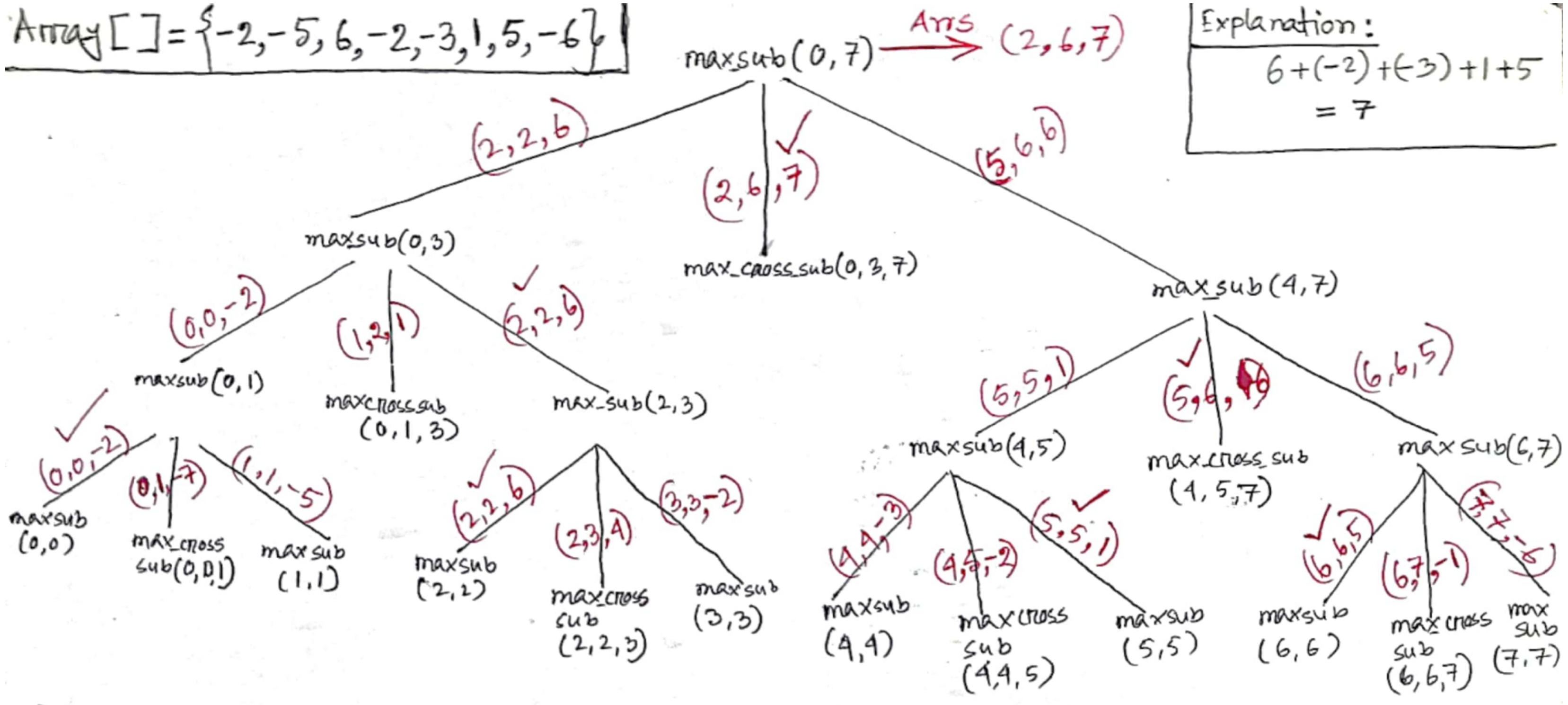
The following figure contains another sample input. Your code should output from 8 to 11 with sum 43. So, you should buy the stock on day 7, sell on day 11 to gain maximum profit of 43.

Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

Array[ ] = {-2, -5, 6, -2, -3, 1, 5, -6}

Explanation:

6 + (-2) + (-3) + 1 + 5 = 7



Function prototypes: maxsub(low, high)

and max_cross_sub(low, mid, high)

Return type: (from, to, sum)

* values that are propagated upwards are mark with a ✓.
* values with maximum sum are propagated upwards.