Maximum sub-Array sum

Saturday, September 9, 2023 2:33 PM

Problem Statement: Find consecutive memory locations in an array having maximum sum.
Let's take an example of stocks. Imagine we have a list of numbers that show when the stock prices went up and down anc data analysis. One very natural question that comes to our mind in this particular scenario is to determine the **maximum pr time span** when the maximum profit was earned.

Solution:
Brute force:

Compute all the possibilities and then determine the best out of all the possibilities. Time complexity will be $O(N^3)$

Divide and conquer:

Divide the array into two equally halved sub-problems i.e., from **left to mid** and **mid+1 to right**
There are three possibilities about the max sub-array:

1: max sub-array could be entirely within the left sub-problem
2: max sub-array could be entirely within the right sub-problem
3: max sub-array might be in the cross-sectional region of left and right sub problem

Just like merge sort, divide the array into equally halved sub problems until the base case will occur.

Algorithm of maxSubArray:

```
maxSubArray(Arr[], left, right)
{
    if(left < right)
    {
        Mid = (left + right)/2
        (L_st, L_end, L_sum) = maxSubArray(Arr, left, Mid)
        (R_st, R_end, R_sum) = maxSubArray(Arr, Mid+1, right)
        (Cr_st, Cr_end, Cr_sum) = maxCrossing(Arr, left, Mid, right)

        if( L_sum >= R_sum AND R_sum >= Cr_sum)
            Return (L_st, L_end, L_sum)
        else if (R_sum > Cr_sum)
            Return (R_st, R_end, R_sum)
        Else
            Return  (Cr_st, Cr_end, Cr_sum)
    }
    Return (left, right, A[left])
}
// following function will be used to determine the maximum sum in cross-sectional region of left and right sub array. First
maximum
// sum in the left sub array by starting from mid index and going down towards the starting index. We will also keep the trac
index of
// max left sub array. Similarly we will compute the maximum sum in the right sub array by starting from the index (mid+1)
besides
// computing maximum sum in the right sub array, we will also keep the track of ending index of max right sub array.
// Since we have starting and ending positions of max left and right sub arrays respectively and also the max sum in the lef
so it means
// that we have calculated the max crossing sum ranging from (max_Left_starting_Index to max_Right_ending_Index and t
sum will be the // sum of max_Left_sum and max_right_sum.
maxCrossing(Arr[], left, Mid, right)
{
    L_st, R_end, sum = 0
    L_sum = INT_MIN
    R_sum = INT_MIN

    For(i = Mid downto left)
    {
        sum + = Arr[i]
        If(sum > L_sum)
        {
            L_sum = sum
            L_st = i
        }
    }
    sum = 0
    For(j = Mid+1 to right)
```

```
    {
        sum + = Arr[j]
        If(sum > R_sum)
        {
            R_sum = sum
            R_end = j
        }
    }
    Return (L_st, R_end, L_sum + R_sum)
}
```