The Maximum Subsequence Sum Problem

Given (possibly negative) integers $a_1, a_2, \ldots, a_n$, find the maximum value of $\sum_{k=i}^{j} a_k$. The maximum subsequence sum is defined to be 0 if all the integers are negative.

For example, given the sequence -2, 11, -4, 13, -5, -2, the maximum subsequence sum is 20: $a_2$ through $a_4$.

A simple algorithm to solve this problem is the following.


Let sum and maxSum be integers initialized to 0.

For integer $i = 1$ to $N$ do
  Let sum = 0
  For integer $j = i$ to $N$ do
    Let sum = sum + $A[j]$
    If( sum > maxSum ) then
      Let maxSum = sum
  End For
End For

Return maxSum.

This simple algorithm solves the problem and is an $O(N^2)$ algorithm. There exists a better algorithm that uses a divide-and-conquer approach to solve the problem in $O(N \log N)$ time for a sequence of length $N$. The algorithm works by dividing the sequence into two halves. Then the maximum sum subsequence is either (1) entirely contained in the left half, or (2) entirely contained in the right half, or (3) it crosses the middle and is in both halves. The algorithm solves the first two cases recursively. Then it finds the maximum sum of the subsequences that have the last element of the left side as their right end, and it finds the maximum sum of the subsequences that have the first element of the right side as their left end. These two sums are added together for the third case. The largest of the three cases is the maximum subsequence sum.

For example, consider the sequence 4, -3, 5, -2 || -1, 2, 6, -4, where || marks the half-way point. The maximum subsequence sum of the left half is 6: 4 + -3 + 5. The maximum subsequence sum of the right half is 8: 2 + 6. The maximum subsequence sum of sequences having -2 as the right edge is 4: 4 + -3 + 5 + -2; and the maximum subsequence sum of sequences having -1 as the left edge is 7: -1 + 2 + 6. Therefore, comparing 6, 8 and 11 (4 + 7), the maximum subsequence sum is 11 and the subsequence spans both halves: 4 + -3 + 5 + -2 + -1 + 2 + 6.

The pseudocode for this algorithm follows. It is interesting to note that this is not the most efficient algorithm. A linear, $O(N)$, exists. It is known as an On-Line algorithm because at any given time, it can correctly give an answer to the subsequence problem for data it has already read. An on-line algorithm that runs in linear time and uses constant space is about as good as an algorithm can get.
Recursive Algorithm MaxSubsequenceSum
Inputs: A = <a₁, a₂, … aₙ> //The sequence being evaluated
     Integers left, right //These are the left and right
     //indices of the subsequence

If( left = right ) then // This is the base case for the recursion
    If( a_leaf > 0 ) then
        Return a_leaf
    Else
        Return 0

Let center be an integer.
Set center to (left + right) / 2.

//Calculate the maximum left half sum
Let maxLeftSum be an integer.
Call the algorithm recursively using A, left, and center as inputs
and set maxLeftSum to the result.

//Calculate the maximum right half sum.
Let maxRightSum be an integer.
Call the algorithm recursively using A, center, and right as inputs
and set maxRightSum to the result.

//Calculate the maximum sum that spans both halves.
Let leftBorderSum and maxLeftBorderSum be integers.
Set leftBorderSum and maxLeftBorderSum to 0.
For( integer i = center to left) do //Note this is counting down
    Set leftBorderSum to leftBorderSum + aᵢ
    If( leftBorderSum > maxLeftBorderSum ) then
        Set maxLeftBorderSum to leftBorderSum.

Let rightBorderSum and maxRightBorderSum be integers.
Set rightBorderSum and maxRightBorderSum to 0.
For( integer i = center to right ) do
    Set rightBorderSum to rightBorderSum + aᵢ
    If( rightBorderSum > maxRightBorderSum ) then
        Set maxRightBorderSum to rightBorderSum.

Let maxMiddleSum be an integer.
Set maxMiddleSum to maxLeftSum + maxRightSum.

//Calculate and return largest of the three sums
Let maxSum be an integer.
Set maxSum to the maximum of maxLeftSum, maxMiddleSum, or maxRightSum

Return maxSum.