Divide and Conquer Algorithms

The Maximum-subarray Problem

Problem:

Input: *an array $A[1...n]$ of (positive/negative) numbers.*

Output: *Indices i and j such that $A[i...j]$ has the greatest sum of any nonempty, contiguous subarray of A, along with the sum of the values in $A[i...j]$.*

Note: Maximum subarray might not be unique, though its value is, so we speak of a maximum subarray, rather than the maximum subarray.

The Maximum-subarray Problem

Example 1:

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

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

The Maximum-subarray Problem

Solve by brute force:

- Total number of subarrays $A[i...j]$:

$$\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{1}{2}n(n-1) = \Theta(n^2)$$

  plus the arrays of length $= 1$.

- Check all subarrays: $\Theta(n^2)$
  (can organize the computation so that each subarray $A[i...j]$ takes $O(1)$ time, given that $A[i..., j-1]$ has been computed.)

The Maximum-subarray Problem

Solve by divide-and-conquer:

- Subproblem: Find a maximum subarray of $A[low...high]$
- Divide-and-Conquer algorithm
  1. Divide: the (sub)array 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 best of the three
- This strategy works because any subarray must either lie entirely in one side of midpoint or cross the midpoint.

The Maximum-subarray Problem

Pseudocode of DC algorithm:

```
MaxSubarray(A,low,high)
if high == low                    // base case: only one element
   return (low, high, A[low])
else                              // divide, conquer and combine
   mid = floor( (low + high)/2 )
   (leftlow,lefthigh,leftsum) = MaxSubarray(A,low,mid)
   (rightlow,righthigh,rightsum) = MaxSubarray(A,mid+1,high)
   (xlow,xhigh,xsum) = MaxXingSubarray(A,low,mid,high)
   if leftsum >= rightsum and leftsum >= xsum
      return (leftlow,lefthigh,leftsum)
   else if rightsum >= leftsum and rightsum >= xsum
      return (rightlow,righthigh,rightsum)
   else
      return (xlow,xhigh,xsum)
   end if
end if
```

The Maximum-subarray Problem

Pseudocode of DC algorithm, cont'd:

```
MaxXingSubarray(A,low,mid,high)
leftsum = -infty; sum = 0   // Find a max-subarray of A[i..mid]
for i = mid downto low
    sum = sum + A[i]
    if sum > leftsum
       leftsum = sum
       maxleft = i
    end if
end for
rightsum = -infty; sum = 0  // Find a max-subarray of A[mid+1..j]
for j = mid+1 to high
    sum = sum + A[j]
    if sum > rightsum
       rightsum = sum
       maxright = j
    end if
end for
return (maxleft,maxright,leftsum + rightsum)
```

The Maximum-subarray Problem

Remarks:

1. Initial call: MaxSubarray(A,1,n)
2. Base case is when the subarray has only 1 element.
3. Divide by computing mid.
   Conquer by the two recursive calls to MaxSubarray. and a call to MaxXingSubarray
   Combine by determining which of the three results gives the maximum sum.
4. Complexity:

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + \Theta(n) + \Theta(1) \\ &= \Theta(n \lg n) \end{aligned}$$

5. Question: What does MaxSubarray returns when all elements of A are negative?