IV. Divide-and-Conquer Algorithms
Divide-and-Conquer algorithms – Overview

The divide-and-conquer (DC) strategy solves a problem by

- **Breaking** the problem into subproblems that are themselves smaller instances of the same type of problem ("divide"),
- **Recursively** solving these subproblems ("conquer"),
- ** Appropriately** combining their answers ("combine")

Recall that MergeSort serves as our first example of the DC paradigm.
The maximum-subarray problem

Problem:

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

**Output:** (1) Indices \( i \) and \( j \) such that the subarray \( A[i...j] \) has the greatest sum of any nonempty contiguous subarray of \( A \), and (2) 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:

<table>
<thead>
<tr>
<th>Day</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>10</td>
<td>11</td>
<td>7</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>Change $A[...]$</td>
<td>1</td>
<td>-4</td>
<td>3</td>
<td>-4</td>
<td></td>
</tr>
</tbody>
</table>

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

#### Example 2:

<table>
<thead>
<tr>
<th>Day</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>10</td>
<td>11</td>
<td>7</td>
<td>10</td>
<td>14</td>
<td>12</td>
<td>18</td>
</tr>
<tr>
<td>Change $A[...]$</td>
<td>1</td>
<td>-4</td>
<td>3</td>
<td>4</td>
<td>-2</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

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

Algorithm 1: 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: $\Theta(n^2)$
The maximum-subarray problem

Algorithm 2: Solve by Divide-and-Conquer:

▶ Generic problem: Find a maximum subarray of $A[low...high]$
▶ Initial call: $low = 1$ and $high = n$
▶ DC 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.
The maximum-subarray problem

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

MaxXingSubarray(A, low, mid, high)
leftsum = -infty; sum = 0  // Find 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 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 the indices i and j and the sum of two subarrays
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:

   \[ T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n) + \Theta(1) \]
   \[ = \Theta(n \log n) \]

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