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

The divide-and-conquer (DC) strategy solves a problem by
1. Breaking the problem into subproblems that are themselves smaller instances of the same type of problem ("divide"),
2. Recursively solving these subproblems ("conquer"),
3. Appropriately combining their answers ("combine")

Recall that MergeSort serves as our first example of the DC paradigm. In addition, in Homework 1, we have also explored the DC strategy for finding min and max, ...
Divide-and-Conquer algorithms – Overview

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

1. *Breaking* the problem into subproblems that are themselves smaller instances of the same type of problem (*“divide”*),
2. *Recursively* solving these subproblems (*“conquer”*),
3. * Appropriately* combining their answers (*“combine”*)
Divide-and-Conquer algorithms – Overview

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

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

Recall that MergeSort serves as our first example of the DC paradigm. In addition, in Homework 1, we have also explored the DC strategy for finding min and max, ...
The maximum-subarray problem

Problem statement:
- 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

Problem statement:

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

*(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

Problem statement:

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 $A[i...j]$. 

The maximum-subarray problem

Problem statement:

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: stock prices and changes

<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>
The maximum-subarray problem

Example 1: stock prices and changes

<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: stock prices and changes

<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>-2</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

**Example 1:** stock prices and changes

<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[\ldots]))</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:** stock prices and changes

<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[\ldots]))</td>
<td>1</td>
<td>-4</td>
<td>3</td>
<td>4</td>
<td>-2</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
The maximum-subarray problem

**Example 1:** stock prices and changes

<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>


**Example 2:** stock prices and changes

<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>

The maximum-subarray problem

Example 3: stock prices and changes

|     | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| A   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|     | 100| 113| 110| 85 | 105| 102| 86 | 63 | 81 | 101| 94 | 106| 101| 79 | 94 | 90 | 97 |
|     | 13 |   -3|  -25|  20|   -3|  -16|  -23|  18|  20|   -7|  12|  -5|  -22|  15|  -4|  7 |
The maximum-subarray problem

**Example 3: stock prices and changes**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100</td>
<td>113</td>
<td>110</td>
<td>85</td>
<td>105</td>
<td>102</td>
<td>86</td>
<td>63</td>
<td>81</td>
<td>101</td>
<td>94</td>
<td>106</td>
<td>101</td>
<td>79</td>
<td>94</td>
<td>90</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>-3</td>
<td>-25</td>
<td>20</td>
<td>-3</td>
<td>-16</td>
<td>-23</td>
<td>18</td>
<td>20</td>
<td>-7</td>
<td>12</td>
<td>-5</td>
<td>-22</td>
<td>15</td>
<td>-4</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

- maximum-subarray: $A[i...j]$?
The maximum-subarray problem

Example 3: stock prices and changes

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100</td>
<td>113</td>
<td>110</td>
<td>85</td>
<td>105</td>
<td>102</td>
<td>86</td>
<td>63</td>
<td>81</td>
<td>101</td>
<td>94</td>
<td>106</td>
<td>101</td>
<td>79</td>
<td>94</td>
<td>90</td>
<td>97</td>
</tr>
<tr>
<td>A</td>
<td>13</td>
<td>-3</td>
<td>-25</td>
<td>20</td>
<td>-3</td>
<td>-16</td>
<td>-23</td>
<td>18</td>
<td>20</td>
<td>-7</td>
<td>12</td>
<td>-5</td>
<td>-22</td>
<td>15</td>
<td>-4</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

- maximum-subarray: $A[i...j]$?
- Answer: $A[8...11]$ and sum = 43!
The maximum-subarray problem

Algorithm 1. Solve by brute-force
The maximum-subarray problem

Algorithm 1. Solve by **brute-force**

- Check all subarrays
The maximum-subarray problem

Algorithm 1. Solve by **brute-force**

- Check all subarrays
- 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.
The maximum-subarray problem

**Algorithm 1.** Solve by **brute-force**

- Check all subarrays
- 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 $= 1$.
- Cost $T(n) = \Theta(n^2)$. 
The maximum-subarray problem

Algorithm 2. Solve by Divide-and-Conquer
The maximum-subarray problem

Algorithm 2. Solve by Divide-and-Conquer

- Generic problem:
  
  Find a maximum subarray of $A[low...high]$

  with initial call: $low = 1$ and $high = n$
The maximum-subarray problem

Algorithm 2. Solve by Divide-and-Conquer

- **Generic problem:**
  Find a maximum subarray of $A[low...high]$
  with 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
The maximum-subarray problem

Algorithm 2. Solve by Divide-and-Conquer

- Generic problem:
  Find a maximum subarray of $A[low...high]$  
  with 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

- Correctness: 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.
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:
   \[ T(n) = 2T(\frac{n}{2}) + \Theta(n) + \Theta(1) = \Theta(n \log n) \]
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 \( \text{mid} \).
   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:

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