IV. Divide-and-Conquer Algorithms

Divide-and-Conquer algorithms – Overview

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

The maximum-subarray problem

Problem statement:

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

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

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

The maximum-subarray problem

Example 1: stock prices and changes

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

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

The maximum-subarray problem

Example 1: stock prices and changes

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

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

Example 2: stock prices and changes

Day	0	1	2	3	4	5	6
Price	10	11	7	10	14	12	18
Change ($= A[...]$)		1	-4	3	4	-2	6

The maximum-subarray problem

Example 1: stock prices and changes

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

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

Example 2: stock prices and changes

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

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
	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
A		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

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

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

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
	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
A		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

- 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.

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:

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