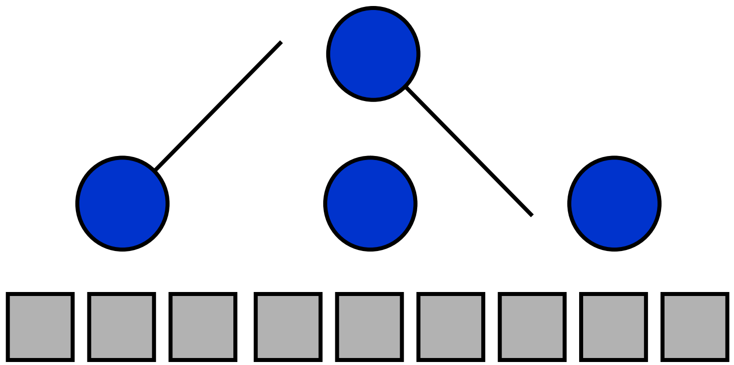
Divide-and-Conquer Technique:
Maximum Subarray problem

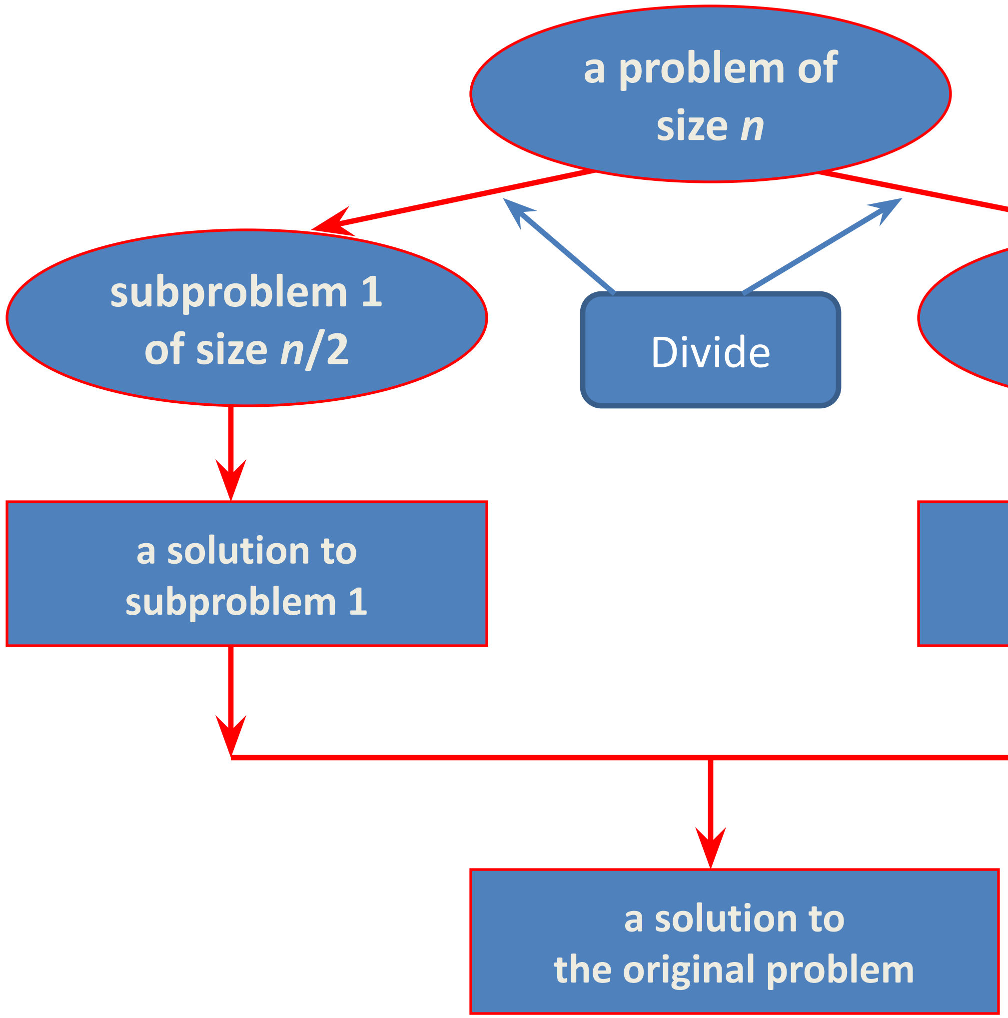
Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET

Divide-and-Conquer

- Divide-and-Conquer is a general algorithm design paradigm:
  - Divide the problem into a number of subproblems that are smaller instances of the same problem
  - Conquer the subproblems by solving them recursively
  - Combine the solutions to the subproblems into the solution for the original problem
- The base case for the recursion are subproblems of constant size
- Analysis can be done using **recurrence equations**



Divide-and-Conquer

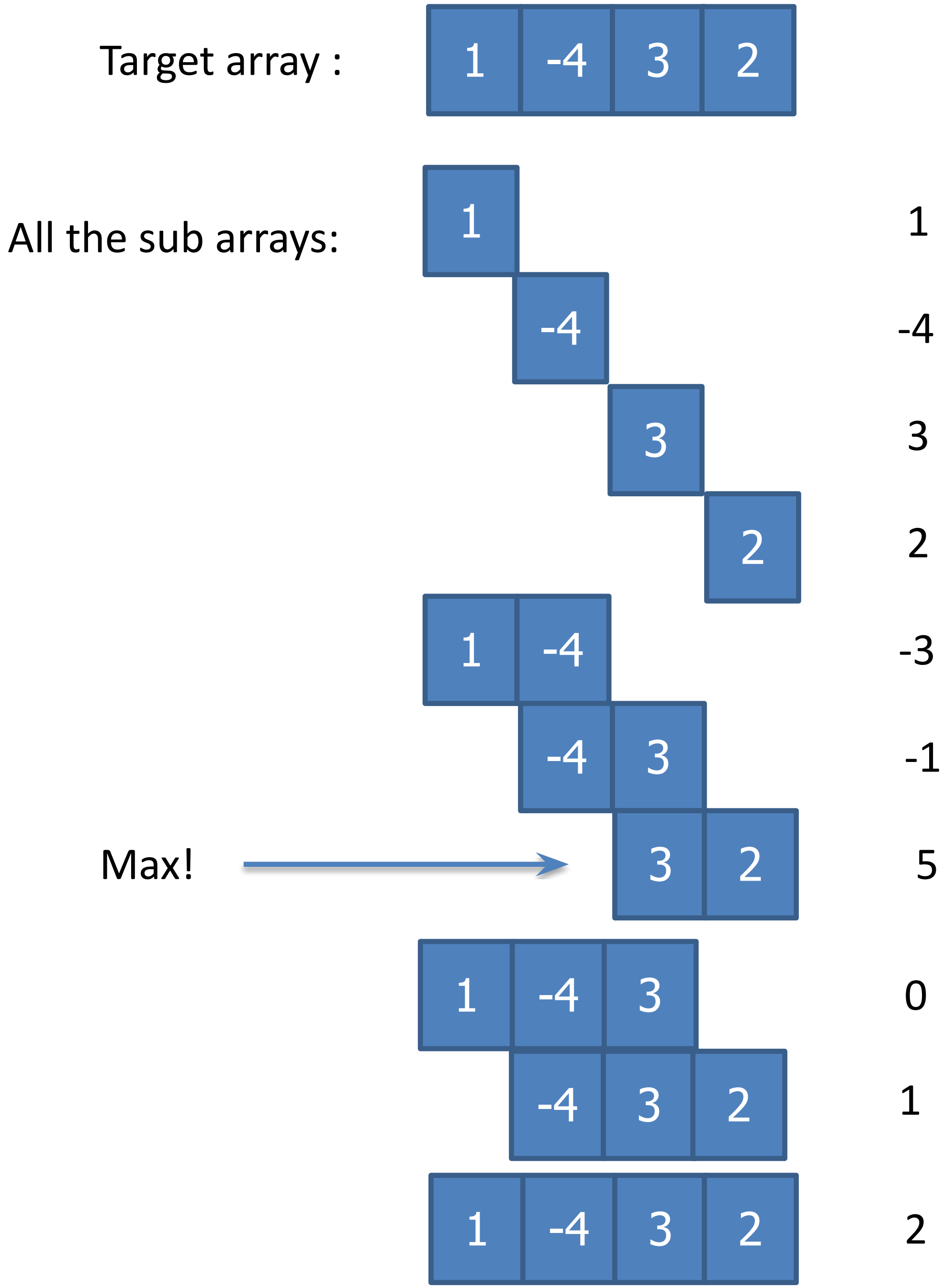


Maximum Subarray Problem

- *Input:* an array $A[1..n]$ of n numbers
  - Assume that some of the numbers are negative, because this problem is trivial when all numbers are nonnegative
- *Output:* a nonempty subarray $A[i..j]$ having the largest sum $S[i, j] = a_i + a_{i+1} + \ldots + a_j$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	**13**	**-3**	**-25**	**20**	**-3**	**-16**	**-23**	**18**	**20**	**-7**	**12**	**-5**	**-22**	**15**	**-4**	**7**

maximum subarray



What is a maximum subarray?

Ans: The subarray with the largest sum

What is the brute-force time?

Brute-Force Algorithm

All possible contiguous subarrays

- A[1..1], A[1..2], A[1..3], ..., A[1..(n-1)], A[1..n]
- A[2..2], A[2..3], ..., A[2..(n-1)], A[2..n]
- ...
- A[(n-1)..(n-1)], A[(n-1)..n]
- A[n..n]

How many of them in total?

$O(n^2)$

Algorithm: For each subarray, compute the sum.

Find the subarray that has the maximum sum.

Brute-Force Algorithm

Example: 2 -6 -1 3 -1 2 -2

sum from A[1]:	2	-4	-5	-2	-3	-1	-3
sum from A[2]:		-6	-7	-4	-5	-3	-5
sum from A[3]:			-1	2	1	3	1
sum from A[4]:				3	2	4	2
sum from A[5]:					-1	1	-1
sum from A[6]:						2	0
sum from A[7]:							-2

Brute-Force Algorithm

Outer loop: index variable i to indicate start of subarray, for $1 \le i \le$ n, i.e., A[1], A[2], ..., A[n]

- for $i = 1$ to n do ...

Inner loop: for each start index i, we need to go through A[$i..i$], A[$i..(i+1)$], ..., A[$i..n$]

- use an index j for $i \le j \le n$, i.e., consider A[$i..j$]
- for $j = i$ to n do ...

Brute-Force Algorithm

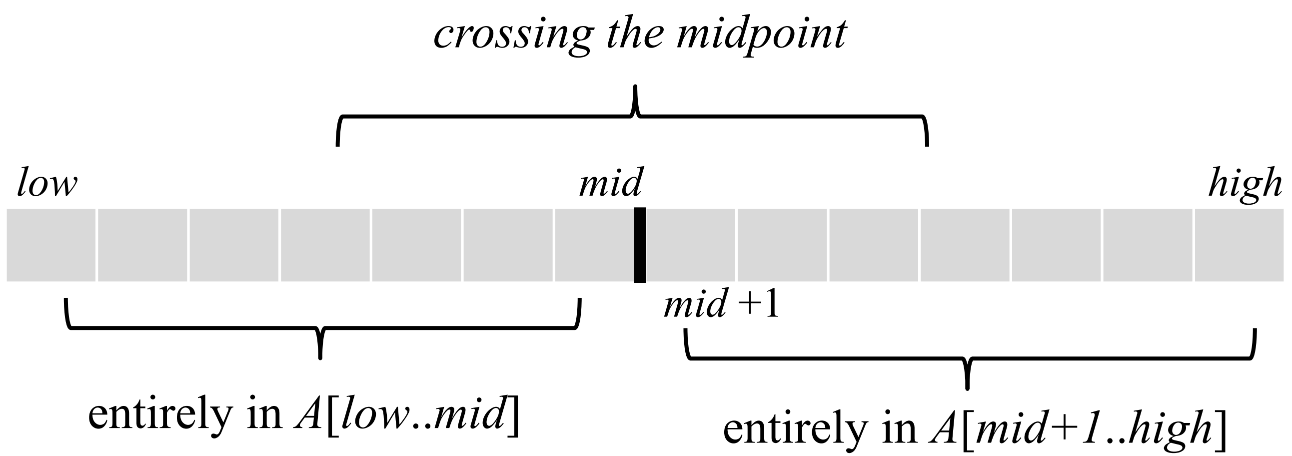
```
max = -∞
for i = 1 to n do
begin
    sum = 0
    for j = i to n do
    begin
        sum = sum + A[j]
        if sum > max
        then max = sum
    end
end
```

Time complexity?
$O(n^2)$

Divide-and-Conquer Algorithm

Possible locations of a maximum subarray $A[i..j]$ of $A[low..high]$, where $mid = \lfloor (low + high)/2 \rfloor$

- entirely in $A[low..mid]$ $(low \le i \le j \le mid)$
- entirely in $A[mid+1..high]$ $(mid < i \le j \le high)$
- crossing the midpoint $(low \le i \le mid < j \le high)$



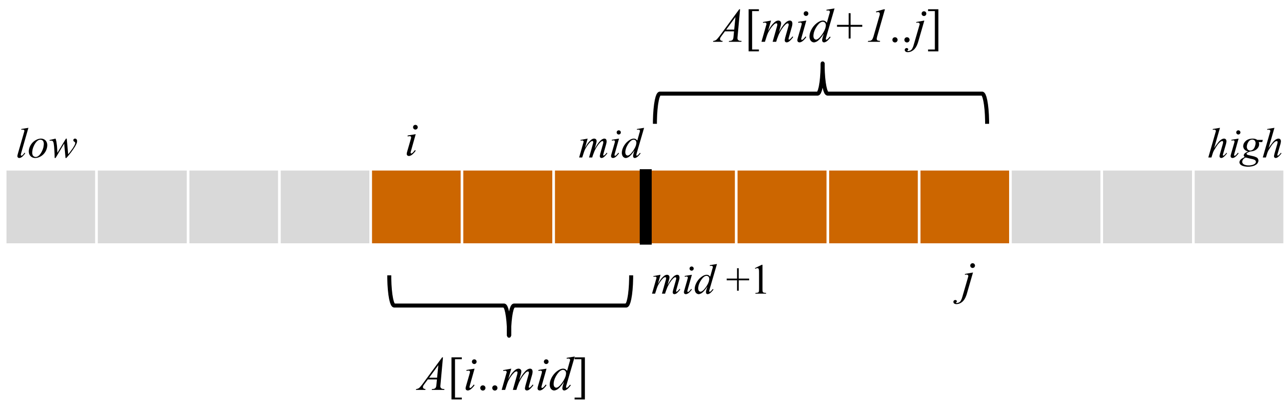
Possible locations of subarrays of A[low..high]

Divide-and-Conquer Algorithm

FIND-MAX-CROSSING-SUBARRAY (A, *low*, *mid, high*)

```
left-sum = -∞        // Find a maximum subarray of the form A[i..mid]
sum = 0
for i = mid downto low
    sum = sum + A[i ]
    if sum > left-sum
       left-sum = sum
       max-left = i
right-sum = -∞       // Find a maximum subarray of the form A[mid + 1 .. j ]
sum =0
for j = mid +1 to high
    sum = sum + A[j]
    if sum > right-sum
       right-sum = sum
       max-right = j
// Return the indices and the sum of the two subarrays
return (max-left, max-right, left-sum + right-sum)
```

Divide-and-Conquer Algorithm



$A[i..j]$ comprises two subarrays $A[i..mid]$ and $A[mid+1..j]$

Divide-and-Conquer Algorithm

mid =5

	1	2	3	4	5	6	7	8	9	10
A	**13**	**-3**	**-25**	**20**	**-3**	**-16**	**-23**	**18**	**20**	**-7**

$S[5 .. 5] =$ -3

$S[4 .. 5] =$ 17 $\Leftarrow$ (max-left = 4)

$S[3 .. 5] =$ -8

$S[2 .. 5] =$ -11

$S[1 .. 5] =$ 2

mid =5

	1	2	3	4	5	6	7	8	9	10
A	**13**	**-3**	**-25**	**20**	**-3**	**-16**	**-23**	**18**	**20**	**-7**

$S[6 .. 6] =$ -16

$S[6 .. 7] =$ -39

$S[6 .. 8] =$ -21

$S[6 .. 9] =$ (max-right = 9) $\Rightarrow$ -1

$S[6..10] =$ -8

$\Rightarrow$ maximum subarray crossing *mid* is $S[4..9] = 16$

Divide-and-Conquer Algorithm

FIND-MAXIMUM-SUBARRAY (*A*, *low*, *high*)

if *high* == *low*

return (*low*, *high*, *A*[*low*]) // *base case: only one element*

else *mid* = $\lfloor low + high / 2 \rfloor$

(*left-low*, *left-high*, *left-sum*) =

FIND-MAXIMUM-SUBARRAY(*A*, *low*, *mid*)

(*right-low*, *right-high*, *right-sum*) **=**

FIND-MAXIMUM-SUBARRAY(*A*, *mid* + *1*, *high*)

(*cross-low*, *cross-high*, *cross-sum*) =

FIND-MAX-CROSSING-SUBARRAY(*A*, *low*, *mid*, *high*)

if *left-sum* ≧ *right-sum* ***and*** *left-sum* ≧ *cross-sum*

return (*left-low*, *left-high*, *left-sum*)

elseif *right-sum* ≧ *left-sum* ***and*** *right-sum* ≧ *cross-sum*

return (*right-low*, *right-high*, *right-sum*)

else return (*cross-low*, *cross-high*, *cross-sum*)

Initial call: FIND-MAXIMUM-SUBARRAY (A, 1, n)

Divide-and-Conquer Algorithm

Analyzing time complexity

FIND-MAX-CROSSING-SUBARRAY : $\Theta(n)$,

where $n = high - low + 1$

FIND-MAXIMUM-SUBARRAY

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$$

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \end{aligned}$$

(similar to merge-sort)

Conclusion: Divide-and-Conquer

- This Divide and conquer algorithm is clearly substantially faster than any of the brute-force methods. It required some cleverness, and the programming is a little more complicated – but the payoff is large.

- Divide and conquer is just one of several powerful techniques for algorithm design
- Divide-and-conquer algorithms can be analyzed using recurrences
- Can lead to more efficient algorithms