Maximum subarray Problem

Maximum Subarray Problem

- Given an array A with integers,
- Find the contiguous subarray of A whose values have the maximum sum $(A[i] + A[i+1] + \cdots + A[j])$
- The maximum sum is zero if all the integers are negative. constraint
- An example
  - Number of possible ranges [i, j] for n numbers in the array A?

	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 sum subarray

Maximum Subarray Problem

- Possible ranges [i, j]

max

  - i = 1, j = 1, 2, …, n
  - i = 2, j = 2, 3, …, n
  - …
  - …
  - i = n-1, j = n-1, n
  - i = n, j = n

= Optimal Maximum Subarray

- There are n(n+1)/2 possible ranges

sum 이 될 수 있는 경우의 수는 n(n+1)/2

naive 하게 모든 pair의 sum을 구해 그중 최대인
값을 찾는다.

A Brute-force Solution

- Try every possible pair of range [i, j] and compute $A[i]+A[i+1]+\ldots+A[j]$.
- Since we have $\Theta(n^2)$ pairs, it takes $O(n^3)$ time.

A Brute-force Algorithm

- FIND-MAXIMUM-SUBARRAY1(A)

```
MaxSum = 0
for i= 1 to n
    for j= i to n
        ThisSum = 0
        for k= i to j
            ThisSum = ThisSum + A[ k ]
        if ThisSum > MaxSum
            MaxSum = ThisSum
return MaxSum
```

← 이 부분을 반복할 필요는 없다.

- Can you do better?

Actual Running Time

- For $n = 100$, actual time is 0.47 seconds on a particular computer.
- Can use this to estimate time for larger inputs:

$$T(n) = cn^3$$

$$T(10n) = c(10n)^3 = 1000cn^3 = 1000T(n)$$

- Inputs size increases by a factor of 10 means that running time increases by a factor of 1,000.
- For $n = 1{,}000$, estimate of running time is 470 seconds. (Actual running time was 449 seconds).
- For $n = 10{,}000$, estimate of running time is 449000 seconds (6 days).

How To Improve

- Remove a loop; not always possible.
- Here it is: innermost loop is unnecessary because it throws away information.
- ThisSum for next j is easily obtained from old value of ThisSum
  - Need $A[i] + A[i+1] + \cdots + A[j-1] + A[j]$
  - Just computed $A[i] + A[i+1] + \cdots + A[j-1]$
  - What we need is $(what\ we\ just\ computed) + A[j]$

A Better Brute-force Algorithm

- FIND-MAXIMUM-SUBARRAY2(A)

```
MaxSum = 0
for i= 1 to n
    ThisSum = 0
    for j= i to n
        ThisSum = ThisSum + A[ j ]
        if ThisSum > MaxSum
            MaxSum = ThisSum
return MaxSum
```

- Can you do better?

Analysis

- Same logic as before: now the running time is quadratic, or $O(n^2)$.
- As we will see, this algorithm is still usable for inputs in the tens of thousands.
- Recall that the cubic algorithm was not practical for this amount of input.

Actual running time

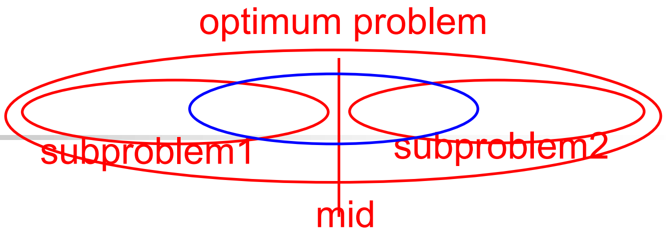
- For $n = 100$, actual time is 0.011 seconds on the same particular computer.
- Can use this to estimate time for larger inputs:

$$T(n) = cn^2$$

$$T(10n) = c(10n)^2 = 100cn^2 = 100T(n)$$

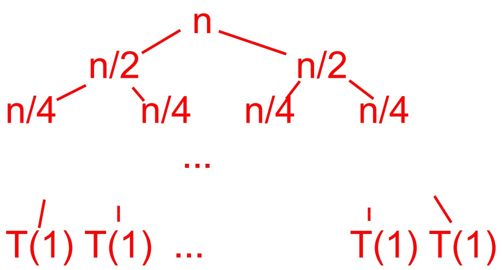
- Inputs size increases by a factor of 10 means that running time increases by a factor of 100.
- For $N = 1{,}000$, estimate of running time is 1.11 seconds. (Actual was 1.12 seconds).
- For $N = 10{,}000$, estimate of running time is 111 seconds.

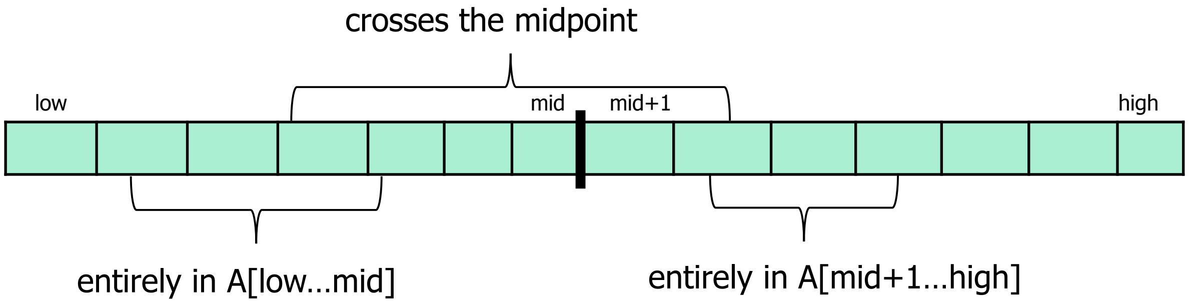
Divide-and-Conquer Algorithm



- The maximum subsequence either
  - lies entirely in the first half
  - lies entirely in the second half
  - starts somewhere in the first half, goes to the last element in the first half, continues at the first element in the second half, ends somewhere in the second half.
- Compute all three possibilities, and use the maximum.
- First two possibilities easily computed recursively.

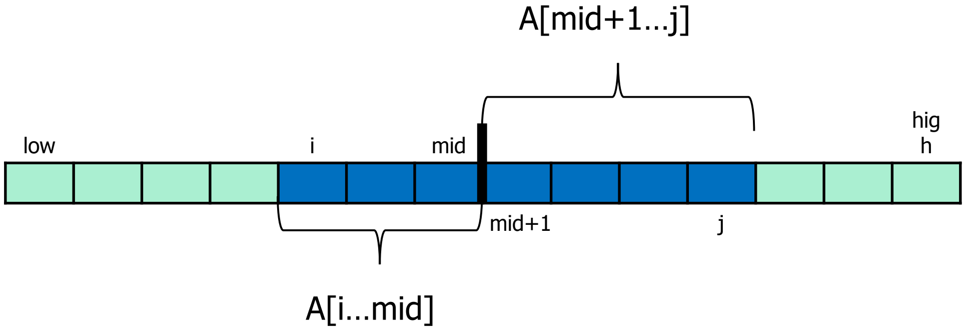
*optimal sol = max{ $\underset{T(n/2)}{\text{subproblem1}}$, $\underset{T(n/2)}{\text{subproblem2}}$, $\underset{n}{\text{crossing problem}}$}





Computing the Third Case

- Easily done with two loops.
- For maximum sum that starts in the first half and extends to the last element in the first half, use a right-to-left scan starting at the last element in the first half.
- For the other maximum sum, do a left-to-right scan, starting at the first element in the first half.



Analysis

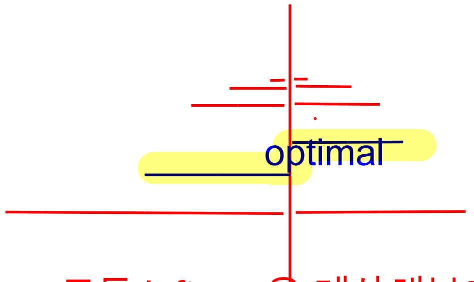
- Let $T(n)$ = the time for an algorithm to solve a problem of size N.
- Then $T(1) = 1$ (1 will be the quantum time unit; remember that constants don't matter).
- $T(n) = 2T\left(\frac{n}{2}\right) + n$
  - Two recursive calls, each of size $n/2$. The time to solve each recursive call is $T(n/2)$ by the above definition
  - Case three takes $O(n)$ time

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
       leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
       rightSum = sum
return leftSum+rightSum
```

max($\sum_{mid}^{low}$sum) = leftsum



모든 leftsum을 계산해보고 최대인것을 택함
모든 rightsum   """"

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
       leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
       rightSum = sum
return leftSum+rightSum
```

leftSum = -∞
sum = 0

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = -∞
sum = 0

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = -∞
sum = -2

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
       leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
       rightSum = sum
return leftSum+rightSum
```

leftSum = -2
sum = -2

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = -2
sum = -3

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = -2
sum = -3

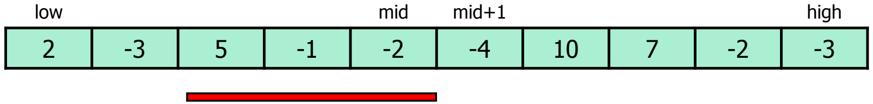
low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = -2
sum = 2



low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = 2
sum = 2

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = 2
sum = -1

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
       leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
       rightSum = sum
return leftSum+rightSum
```

leftSum = 2
sum = -1

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = 2
sum = 1

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = 2
sum = 1

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = 2
rightSum = -∞
sum = 0

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = 2
rightSum = -∞
sum = 0

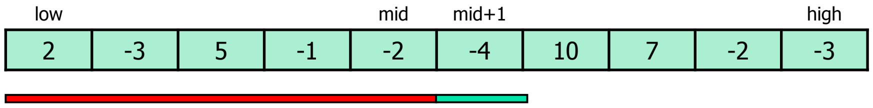
low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
       leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
       rightSum = sum
return leftSum+rightSum
```

leftSum = 2
rightSum = -∞
sum = -4



low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = 2
rightSum = -4
sum = -4

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = 2
rightSum = -4
sum = 6

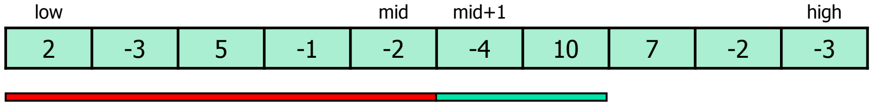
low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3
🟥	🟥	🟥	🟥	🟥	🟩	🟩			

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = 2
rightSum = 6
sum = 6



low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
       leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
       rightSum = sum
return leftSum+rightSum
```

leftSum = 2
rightSum = 6
sum = 13

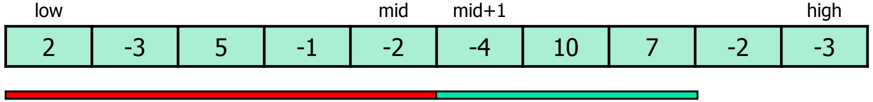
low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = 2
rightSum = 13
sum = 13



low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = 2
rightSum = 13
sum = 11

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = 2
rightSum = 13
sum = 11

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = 2
rightSum = 13
sum = 8

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3
🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩

A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
        leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
        rightSum = sum
return leftSum+rightSum
```

leftSum = 2
rightSum = 13
sum = 8

low				mid	mid+1				high
2	-3	5	-1	-2	-4	10	7	-2	-3

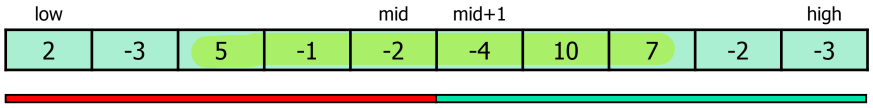
A Divide-and-Conquer Algorithm

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

```
leftSum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > leftSum
      leftSum = sum
rightSum = -∞
sum = 0
for i = mid+1 to high
    sum = sum + A[i]
    if sum > rightSum
      rightSum = sum
return leftSum+rightSum
```

leftSum = 2
rightSum = 13
sum = 8

return 15



A Divide-and-Conquer Algorithm

```
FIND-MAXIMUM-SUBARRAY3(A, low, high)
    if high == low
        return (low, high, A[low] ) // base case
    else
        mid = (low +high)/2
        leftSum =  FIND-MAXIMUM-SUBARRAY3(A, low, mid)
        rightSum =  FIND-MAXIMUM-SUBARRAY3(A, mid+1, high)
        crossSum = FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
    return max(leftSum, rightSum, crossSum)
```

Bottom up
max{
,
,
}

Incremental Algorithm

- Kadane's Algorithm
- If we know
  - the maximum subarray sum ending at position A[i] (Call it ThisSum)
  - the maximum subarray sum for the range [1, i] (Call it MaxSum)
- What is the maximum subarray sum for the range [i, i+1]?
  - max(MaxSum, A[i+1], ThisSum + A[i+1])

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 0
ThisSum = 0

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 0
ThisSum = 0

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 0
ThisSum = 2
j = 1

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 2
ThisSum = 2
j = 1

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

range of MaxSum

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 2
ThisSum = -1
j = 2

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

range of MaxSum

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 2
ThisSum = -1
j = 2

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

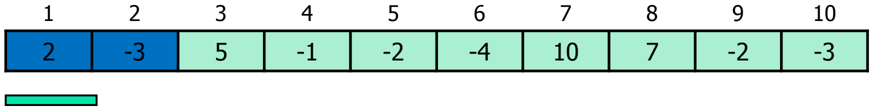
range of MaxSum

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 2
ThisSum = 0
j = 2



1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

range of MaxSum

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 2
ThisSum = 5
j = 3

	1	2	3	4	5	6	7	8	9	10
	2	-3	5	-1	-2	-4	10	7	-2	-3
range of MaxSum	🟩									

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 5
ThisSum = 5
j = 3

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

range of MaxSum

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 5
ThisSum = 4
j = 4

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3
		▬							

range of MaxSum

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 5
ThisSum = 4
j = 4

	1	2	3	4	5	6	7	8	9	10
	2	-3	5	-1	-2	-4	10	7	-2	-3
range of MaxSum			▬							

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 5
ThisSum = 4
j = 4

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

range of MaxSum

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 5
ThisSum = 2
j = 5

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

range of MaxSum

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 5
ThisSum = 2
j = 5

	1	2	3	4	5	6	7	8	9	10
	2	-3	5	-1	-2	-4	10	7	-2	-3
range of MaxSum			🟩							

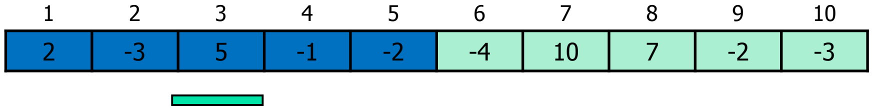
A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 5
ThisSum = 2
j = 5

range of MaxSum



1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 5
ThisSum = -2
j = 6

	1	2	3	4	5	6	7	8	9	10
	2	-3	5	-1	-2	-4	10	7	-2	-3
range of MaxSum			▬							

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 5
ThisSum = -2
j = 6

	1	2	3	4	5	6	7	8	9	10
	2	-3	5	-1	-2	-4	10	7	-2	-3
range of MaxSum			▬							

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 5
ThisSum = 0
j = 6

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

range of MaxSum

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 5
ThisSum = 10
j = 7

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

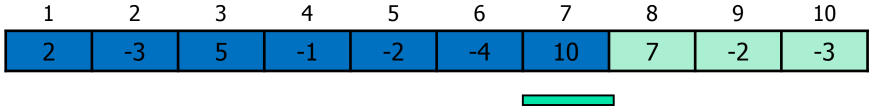
range of MaxSum

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 10
ThisSum = 10
j = 7



1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

range of MaxSum

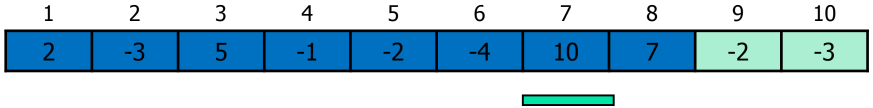
A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 10
ThisSum = 17
j = 8

range of MaxSum



1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 17
ThisSum = 17
j = 8

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

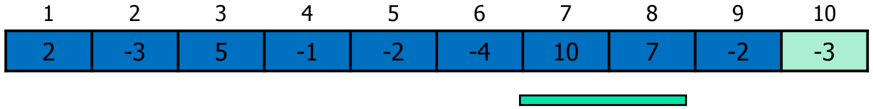
range of MaxSum

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 17
ThisSum = 15
j = 9



1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

range of MaxSum

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 17
ThisSum = 15
j = 9

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

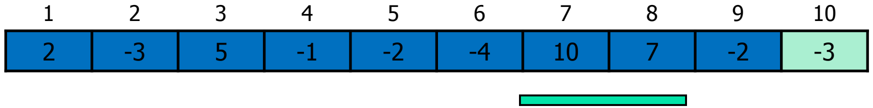
range of MaxSum

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 17
ThisSum = 15
j = 9



1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

range of MaxSum

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 17
ThisSum = 12
j = 10

	1	2	3	4	5	6	7	8	9	10
	2	-3	5	-1	-2	-4	10	7	-2	-3
range of MaxSum							🟩	🟩		

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 17
ThisSum = 12
j = 10

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

range of MaxSum

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 17
ThisSum = 12
j = 10

	1	2	3	4	5	6	7	8	9	10
	2	-3	5	-1	-2	-4	10	7	-2	-3
range of MaxSum							🟩	🟩		

A Linear-time Algorithm

- FIND-MAXIMUM-SUBARRAY4(A)

```
MaxSum = 0
ThisSum = 0
for j = 1 to n
    ThisSum = ThisSum + A[ j ]
    if ThisSum > MaxSum
        MaxSum = ThisSum
    else if ThisSum < 0
        ThisSum = 0
return MaxSum
```

MaxSum = 17
ThisSum = 12
j = 11

return 17

1	2	3	4	5	6	7	8	9	10
2	-3	5	-1	-2	-4	10	7	-2	-3

range of MaxSum

A Linear-time Algorithm

- Linear time (i.e., $O(n)$) algorithm would be best
- Running time is proportional to amount of input
- It makes only one pass through the data
- If the array is on a disk or is being transmitted over the Internet, it can be read sequentially, and there is no need to store any part of it in main memory
- At any point in time, the algorithm can correctly give an answer to the subsequence problem for the data it has already read: We call online algorithm