CS 2009
Design and Analysis of Algorithms

Prepared by: Waheed Ahmed

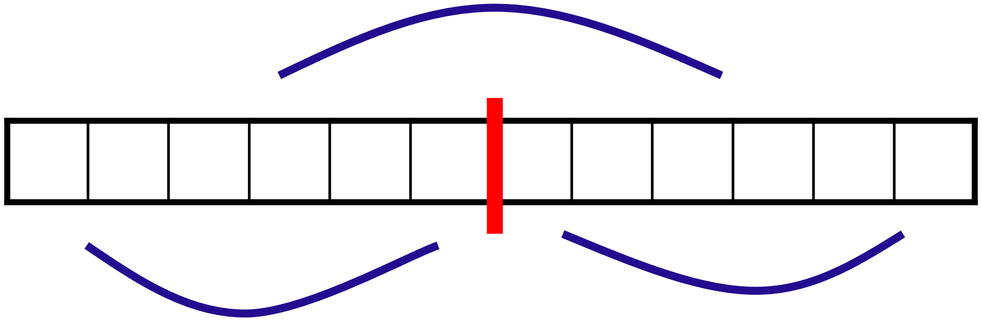
Farrukh Salim Shaikh

Lecture 14:
The Maximum Subarray Problem

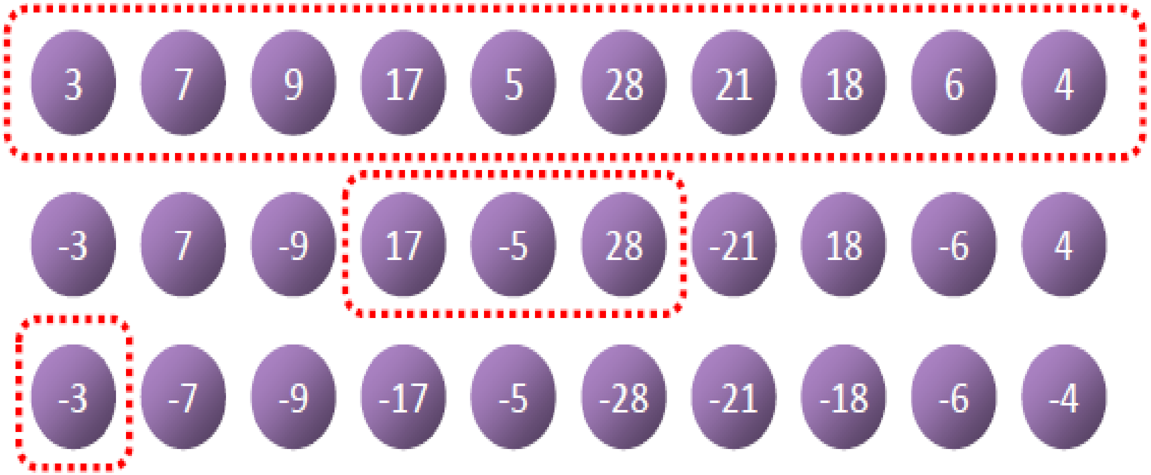
The Maximum Subarray Problem

- *Def:* The maximum subarray problem is the task of finding the largest possible sum of a contiguous subarray, within a given one-dimensional array A[1…n] of numbers.

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



The Maximum Subarray Problem

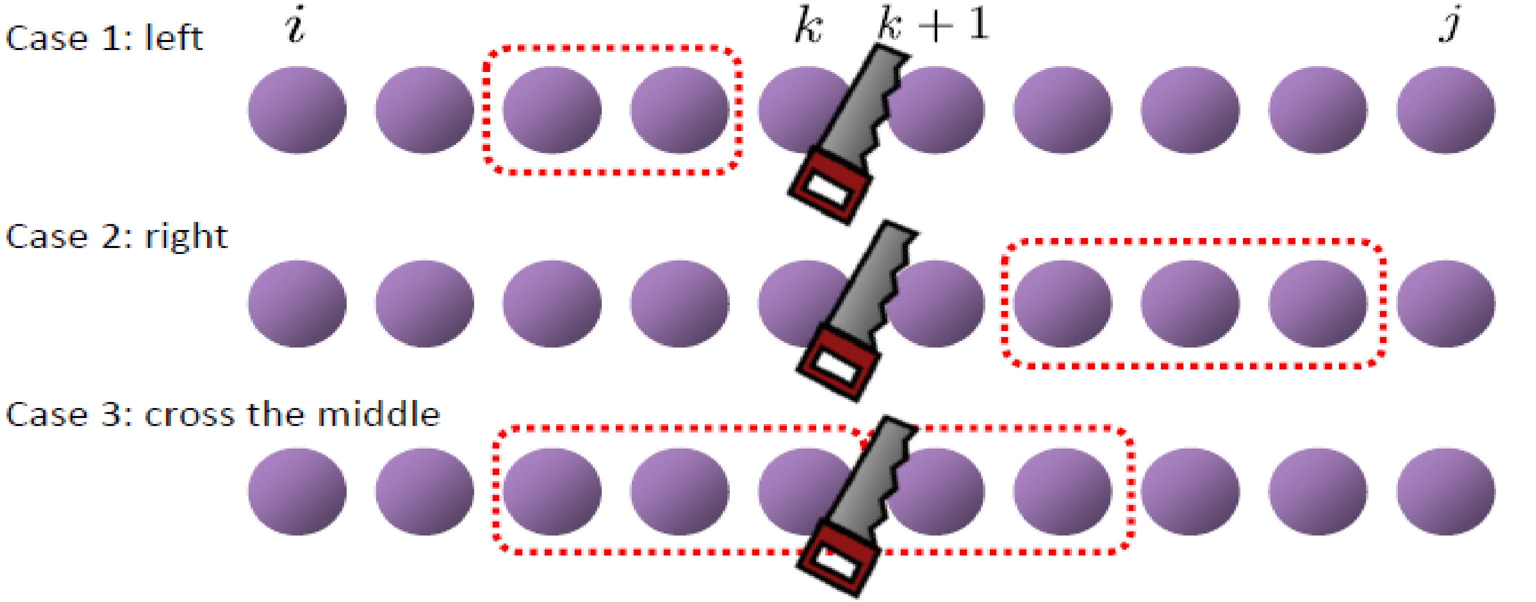


Divide-and-Conquer

- **Base Case (n = 1)**
  - Return itself (maximum subarray)
- **Recursive Case** (n > 1)
  - Divide array into two subarrays.
  - Find maximum sub array recursively
  - **Merge** the results.

Where is Result?

- The maximum subarray for any input must be in one of following cases:



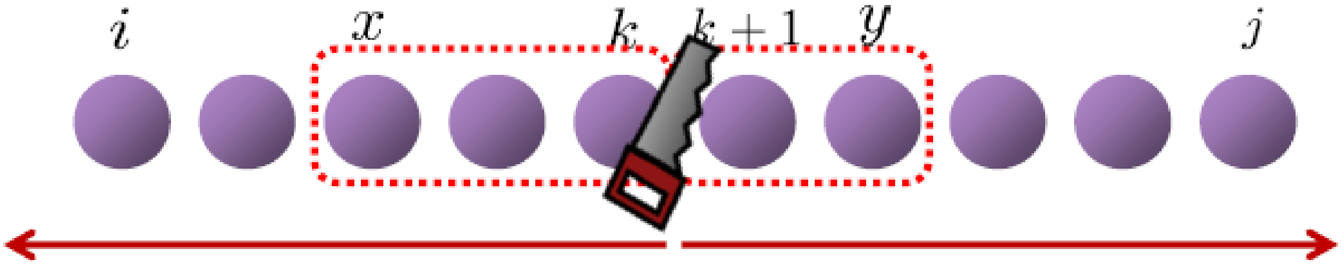
Case 1: `MaxSub(A, i, j) = MaxSub(A, i, k)`

Case 2: `MaxSub(A, i, j) = MaxSub(A, k+1, j)`

Case 3: `MaxSub(A, i, j)` cannot be expressed using `MaxSub`!

Case 3: Cross the Middle

- **Goal: find the maximum subarray that crosses the middle**



(1) Start from the middle to find the left maximum subarray

(2) Start from the middle to find the right maximum subarray

The solution of Case 3 is the combination of (1) and (2)

- **Observation**
  - The sum of $A[x \dots k]$ must be the maximum among $A[i \dots k]$ (left: $i \le k$)
  - The sum of $A[k+1 \dots y]$ must be the maximum among $A[k+1 \dots j]$ (right: $j > k$)
  - Solvable in linear time → $\Theta(n)$

The Maximum Subarray Problem - Example

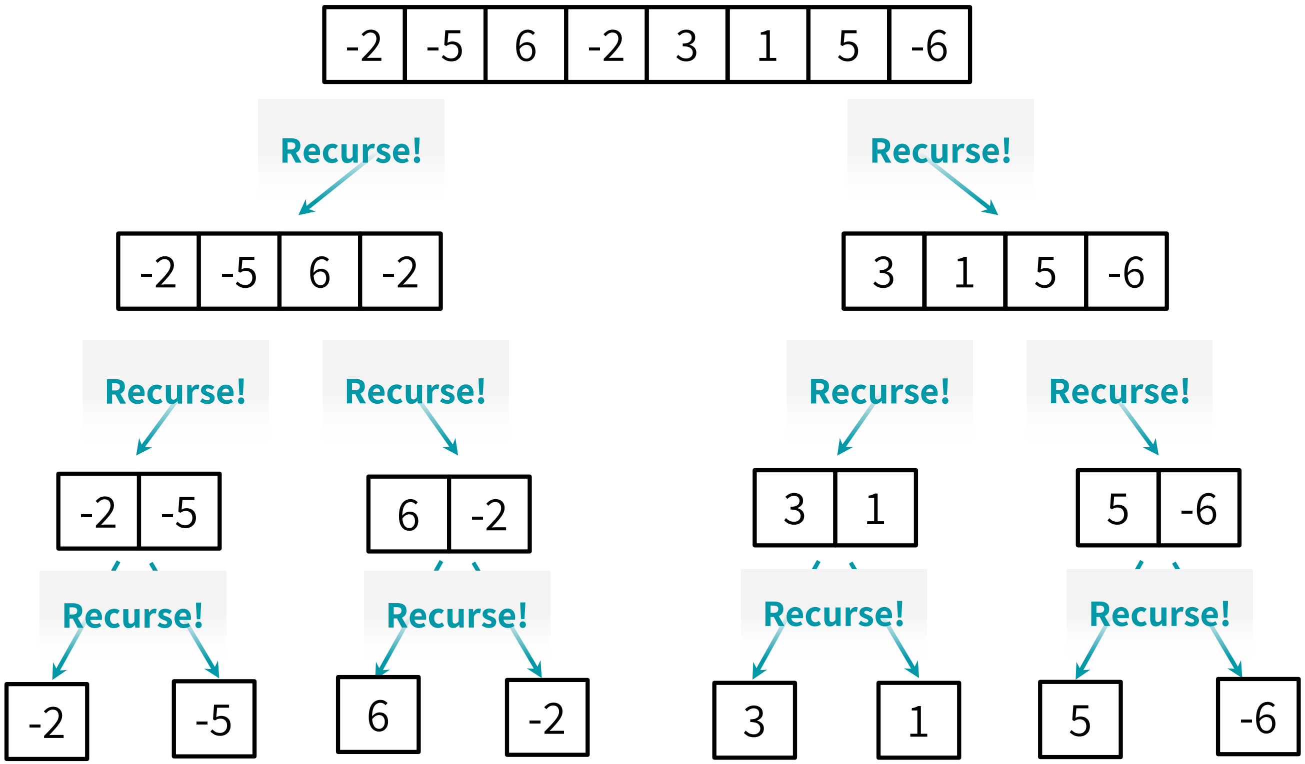
-2	-5	6	-2	3	1	5	-6

What is maximum subarray sum of this array

Maximum subarray sum is

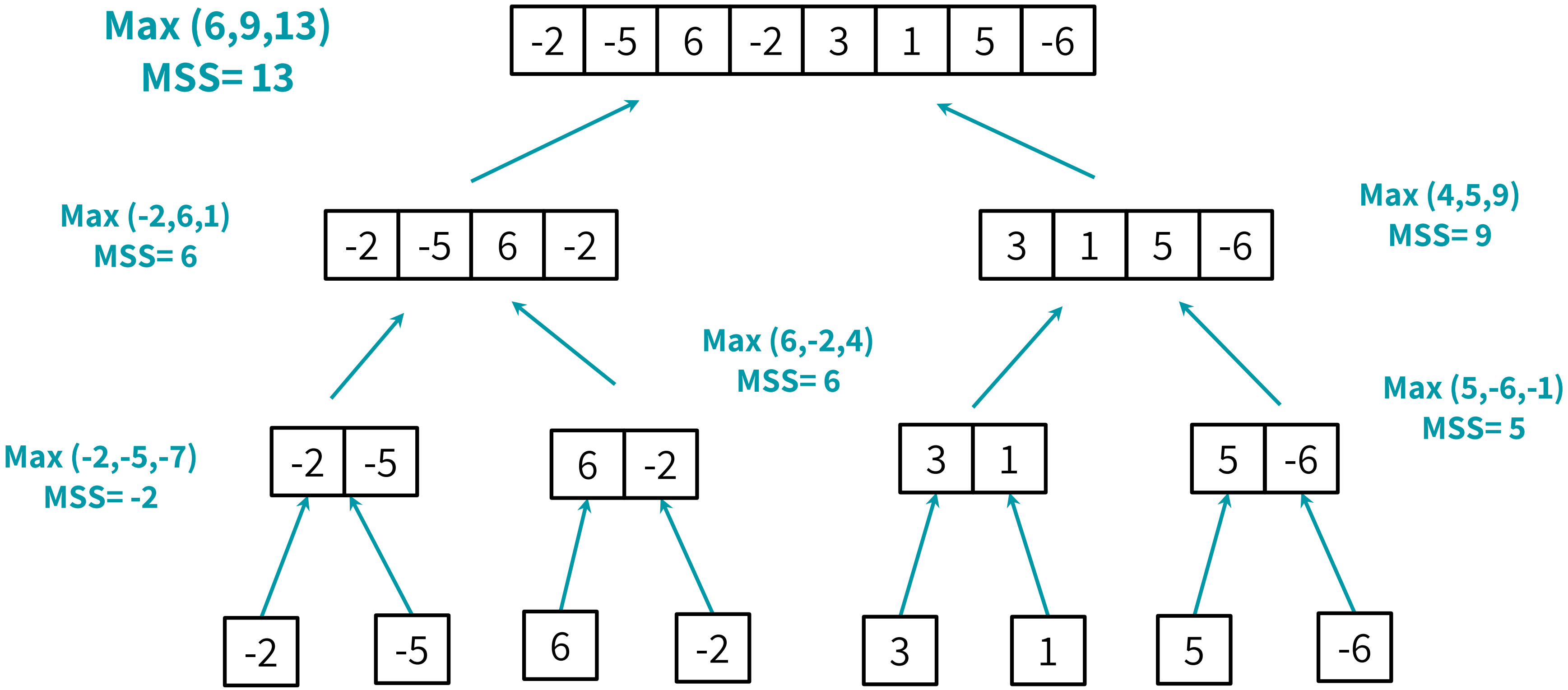
6-2+3+1+5 = 13

MaxSubArray: RECURSIVE CALLS



This is where we hit our base case!

MaxSubArray: RECURSIVE CALLS



Divide and Conquer Solution

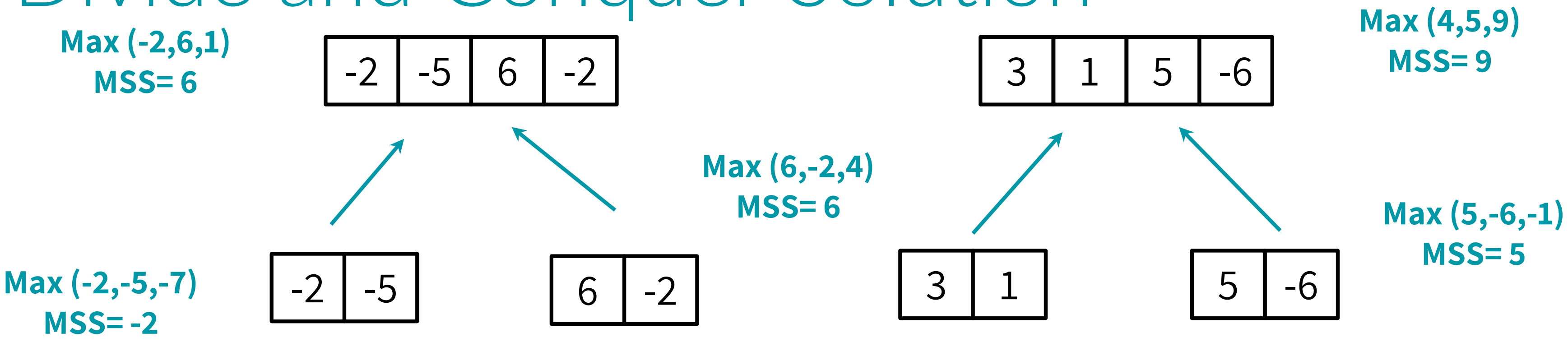
```
MaxSubarray(A, i, j)
  if i == j // base case
    return (i, j, A[i])
  else // recursive case
    k = floor((i + j) / 2)
    (l_low, l_high, l_sum) = MaxSubarray(A, i, k)
    (r_low, r_high, r_sum) = MaxSubarray(A, k+1, j)
    (c_low, c_high, c_sum) = MaxCrossSubarray(A, i, k, j)

  if l_sum >= r_sum and l_sum >= c_sum // case 1
    return (l_low, l_high, l_sum)
  else if r_sum >= l_sum and r_sum >= c_sum // case 2
    return (r_low, r_high, r_sum)
  else // case 3
    return (c_low, c_high, c_sum)
```

Divide

Conquer

Combine

Divide and Conquer Solution



Divide

```
(l_low, l_high, l_sum) = MaxSubarray(A, i, k)
(r_low, r_high, r_sum) = MaxSubarray(A, k+1, j)
(c_low, c_high, c_sum) = MaxCrossSubarray(A, i, k, j)
```

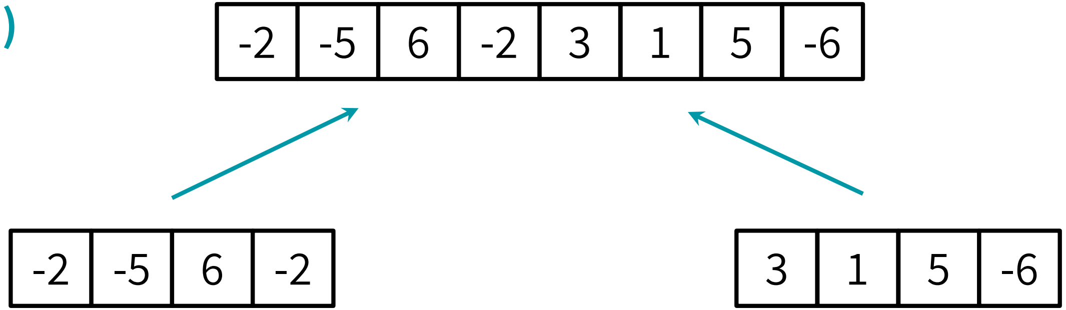
Conquer

```
if l_sum >= r_sum and l_sum >= c_sum // case 1
  return (l_low, l_high, l_sum)
else if r_sum >= l_sum and r_sum >= c_sum // case 2
  return (r_low, r_high, r_sum)
else // case 3
  return (c_low, c_high, c_sum)
```

Combine

Divide and Conquer Solution

Max (6,9,13)
MSS= 13



Max (-2,6,1)
MSS= 6

Max (4,5,9)
MSS= 9

Divide

```
(l_low, l_high, l_sum) = MaxSubarray(A, i, k)
(r_low, r_high, r_sum) = MaxSubarray(A, k+1, j)
(c_low, c_high, c_sum) = MaxCrossSubarray(A, i, k, j)
```

Conquer

```
if l_sum >= r_sum and l_sum >= c_sum // case 1
  return (l_low, l_high, l_sum)
else if r_sum >= l_sum and r_sum >= c_sum // case 2
  return (r_low, r_high, r_sum)
else // case 3
  return (c_low, c_high, c_sum)
```

Combine

Divide and Conquer Solution

```
MaxCrossSubarray(A, i, k, j)
  left_sum = -∞
  sum=0
  for p = k downto i
    sum = sum + A[p]
    if sum > left_sum
      left_sum = sum
      max_left = p

  right_sum = -∞
  sum=0
  for q = k+1 to j
    sum = sum + A[q]
    if sum > right_sum
      right_sum = sum
      max_right = q
  return (max_left, max_right, left_sum + right_sum)
```

$O(k - i + 1)$

$O(j - k)$

$= O(j - i + 1)$

Divide and Conquer Solution

```
MaxSubarray(A, i, j)
  if i == j // base case
    return (i, j, A[i])
  else // recursive case
    k = floor((i + j) / 2)
    (l_low, l_high, l_sum) = MaxSubarray(A, i, k)
    (r_low, r_high, r_sum) = MaxSubarray(A, k+1, j)
    (c_low, c_high, c_sum) = MaxCrossSubarray(A, i, k, j)

  if l_sum >= r_sum and l_sum >= c_sum // case 1
    return (l_low, l_high, l_sum)
  else if r_sum >= l_sum and r_sum >= c_sum // case 2
    return (r_low, r_high, r_sum)
  else // case 3
    return (c_low, c_high, c_sum)
```

$O(1)$

$T(k-i+1)$

$T(j-k)$

$O(j-i+1)$

$O(1)$

$O(1)$

$O(1)$

Divide and Conquer Solution

1. Divide

- Divide a list of size *n* into 2 subarrays of size *n/2* — $\Theta(1)$

2. Conquer

- <u>Recursive case ($n > 1$)</u> — $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$
  - find **MaxSub** for each subarrays
- <u>Base case ($n = 1$)</u> — $\Theta(1)$
  - Return itself
- Find **MaxCrossSub** for the original list — $\Theta(n)$

3. Combine

- Pick the subarray with the maximum sum among 3 subarrays — $\Theta(1)$

- $T(n)$ = time for running `MaxSubarray(A, i, j)` with $j - i + 1 = n$

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n) & \text{if } n \geq 2 \end{cases}$$