## Maximum Subarray Sum Problem

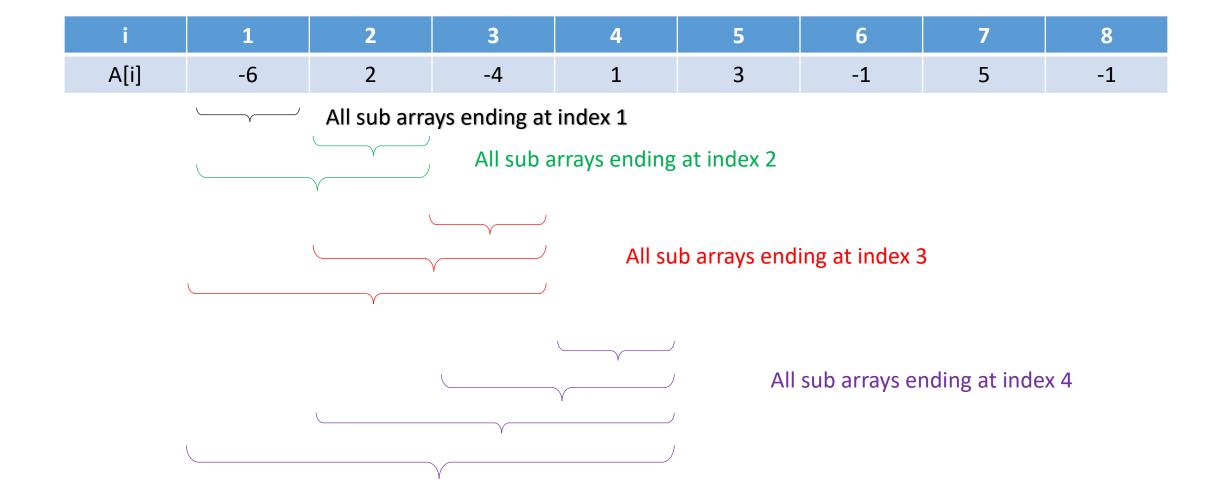
#### Maximum Subarray Problem

- The maximum subarray problem is the task of finding the contiguous subarray within a one-dimensional array of numbers which has the largest sum.
- Example

i	1	2	3	4	5	6	7	8
A[i]	-6	2	-4	1	3	-1	5	-1

Maximum SubArray is from index 4 till index 7 with sum 8

#### Brute Force Solution for Maximum Sub Array Sum



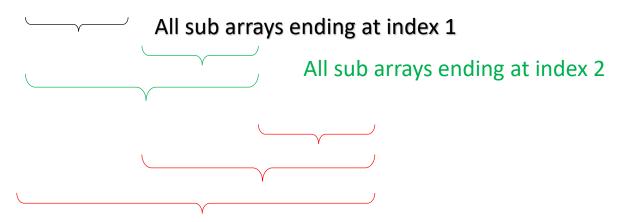
## Brute Force Solution O(n<sup>2</sup>)

```
MaxSubArraySum(A, n)
1.
2.
3.
     MaxSum = -infinity
     for(i = 1 to n)
4.
5.
        subArraySum = 0
6.
         for (j = i to 1)
7.
8.
            subArraySum += A[j]
9.
10.
            MaxSum = Max (MaxSum, subAayraySum)
11.
12. }
13.
     return MaxSum
14.
```

# Brute Force Solution Dry Run

#### O(n<sup>2</sup>) Solution for Maximum Sub Array Sum

i	1	2	3	4	5	6	7	8
A[i]	-6	2	-4	1	3	-1	5	-1



All sub arrays ending at index 3

## Divide and Conquer Solution

i	1	2	3	4	5	6	7	8
A[i]	-6	2	-4	1	3	-1	5	-1

## Divide and Conquer Solution

i	1	2	3	4	5	6	7	8
A[i]	-6	2	-4	1	3	-1	5	-1

## Divide and Conquer Solution O(n lg n)

- Divide array in two equal halves
- SubArrays can be divide into 3 categories
  - Left subarray (start and end index in left half of array)
  - Right subarray (start and end index in right half of array)
  - Crossing subarray (start in left and end in right half of array)

- Left and right subarray sum is calculated using recursion
- Crossing subarray sum is computed using a linear time function

### Divide and Conquer

```
MaxSubArraySum(A, I, r)
            return array[l]
   If (I == r)
   m = (l+r)/2
                    MaxSubArraySum(A, I, m)
    return Max
                    MaxSubArraySum(A, m+1, r)
                    MaxCrossingSum(A, I, m, r)
```

```
MaxCrossingSum(A, I, m, r)
 sum = 0
 for(i = m to I)
       sum = sum + A[i]
       leftSum = Max (leftSum, sum)
 sum = 0
 for(i = m+1 \text{ to } r)
       sum = sum + A[i]
       rightSum = Max (rightSum, sum)
 return leftSum + rightSum
```

```
MaxSubArraySum(A, I, r) [-3, 5, 2, -4, -6, 3, 2, -3]
  If (I == r) return array[I]
  m = (l+r)/2
```

```
MaxSubArraySum(A, I, r) [-3, 5, 2, -4]
     If (I == r) return array[I]
     m = (l+r)/2
     return Max = 7  \begin{bmatrix} [5] = 5 = MaxSubArraySum(A, I, m) \\ [2] = 2 = MaxSubArraySum(A, m+1, r) \\ [5, 2] = 7 = MaxCrossingSum(A, I, m, r)
```

```
MaxSubArraySum(A, I, r) [ -6, 3, 2, -3]
     If (I == r) return array[I]
     m = (l+r)/2
     return Max = 5  \begin{bmatrix} [3] = 3 = MaxSubArraySum(A, I, m) \\ [2] = 2 = MaxSubArraySum(A, m+1, r) \\ [3, 2] = 5 = MaxCrossingSum(A, I, m, r)
```

```
MaxSubArraySum(A, I, r) [ -6, 3]
     If (I == r) return array[I]
     m = (l+r)/2
     [-6] = -6 = MaxSubArraySum(A, I, m)

return Max = 3  [3] = 3 = MaxSubArraySum(A, m+1, r)

[-6, 3] = -3 = MaxCrossingSum(A, I, m, r)
```

• Dry run brute force O(n²) algorithm on following array and show all working. Show all values of MaxSum[i] array. MaxSum[i] array stores maximum sum out of all subarrays ending at index i.

i	1	2	3	4	5	6	7	8	9
A[i]	3	-5	1	5	-4	5	-6	7	-2

• Dry run Divide and Conquer algorithm for Maximum Subarray Sum on following array and show all working.

i	1	2	3	4	5	6	7	8	9
A[i]	3	-5	1	5	-4	5	-6	7	-2

 The pseudocode on Slide 4 returns maximum subarray sum. Modify the pseudocode to also keep track of start and end index of maximum subarray.

The pseudocode on Slide 11 returns maximum crossing subarray sum.
 Modify the pseudocode to also keep track of start and end index of maximum crossing subarray.