

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

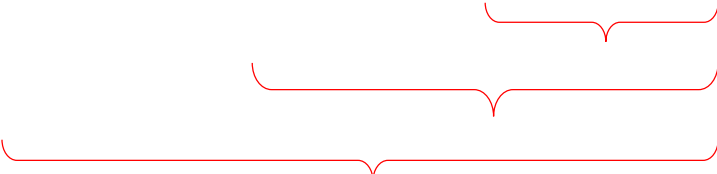
- Maximum SubArray is from index 4 till index 7 with sum 8


Brute Force Solution for Maximum Sub Array Sum

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 1

 All sub arrays ending at index 2

 All sub arrays ending at index 3

 All sub arrays ending at index 4

Brute Force Solution $O(n^2)$

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

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

Brute Force Solution

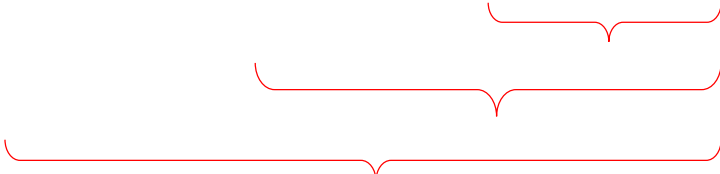
Dry Run

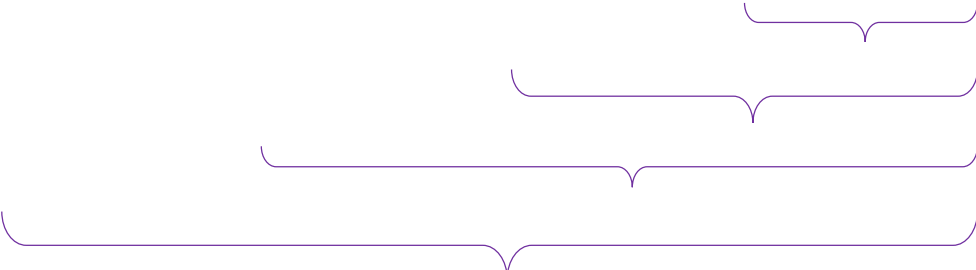
$O(n^2)$ Solution for Maximum Sub Array Sum

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 1

 All sub arrays ending at index 2

 All sub arrays ending at index 3

 All sub arrays ending at index 4

$O(n^2)$ Solution for Maximum Sub Array Sum

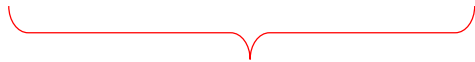
i	1	2	3	4	5	6	7	8
A[i]	-6	2	-4	1	3	-1	5	-1



All sub arrays ending at index 1



All sub arrays ending at index 2



All sub arrays ending at index 3

$O(n^2)$ 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 1



All sub arrays ending at index 2



All sub arrays ending at index 3

What is a sub problem here?

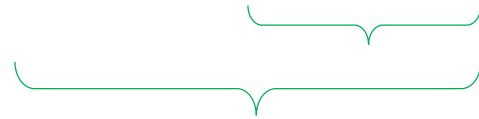
All subarrays ending at index i is a sub problem of original problem.

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



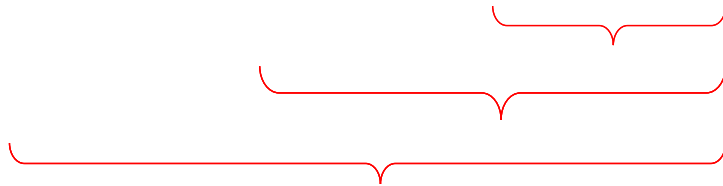
All sub arrays ending at index 1

$$\text{MaxSum}[1] = A[1] = -6$$



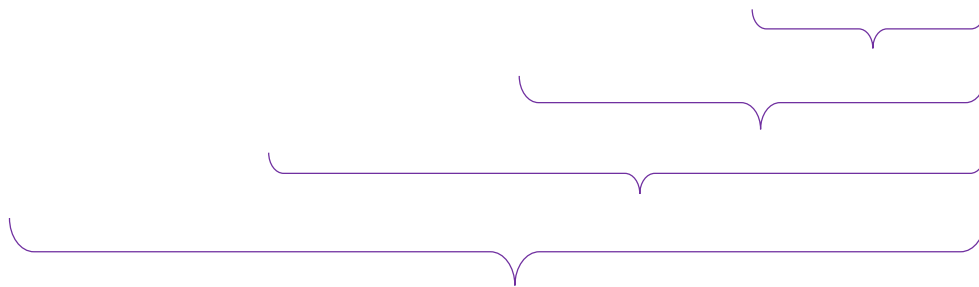
All sub arrays ending at index 2

$$\text{MaxSum}[2] = \text{Max} \begin{cases} A[2] = 2 \\ A[1] + A[2] = -4 \end{cases}$$



All sub arrays
ending at index 3

$$\text{MaxSum}[3] = \text{Max} \begin{cases} A[3] = -4 \\ A[2] + A[3] = 2 \\ A[1] + A[2] + A[3] = -8 \end{cases}$$



All sub arrays
ending at
Index 4

$$\text{MaxSum}[4] = \text{Max} \begin{cases} A[4] = 1 \\ A[3] + A[4] = -3 \\ A[2] + A[3] + A[4] = -1 \\ A[1] + A[2] + A[3] + A[4] = -7 \end{cases}$$

$\text{MaxSum}[i]$ = Maximum sum out of all subarrays that
must end at index i

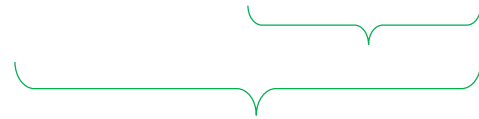
Dynamic Programming Algorithm (Kadane's Algorithm)

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



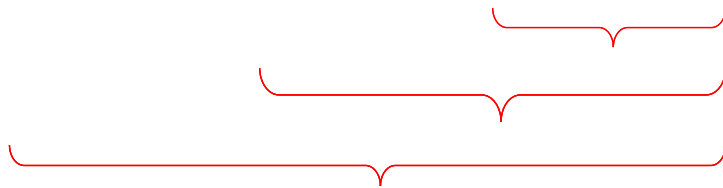
All sub arrays ending at index 1

$$\text{MaxSum}[1] = A[1] = -6$$



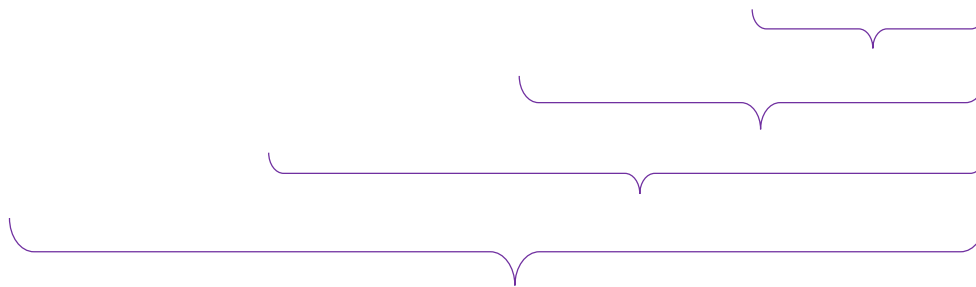
All sub arrays ending at index 2

$$\text{MaxSum}[2] = \text{Max} \begin{cases} A[2] = 2 \\ A[1] + A[2] = -4 \end{cases}$$



All sub arrays
ending at index 3

$$\text{MaxSum}[3] = \text{Max} \begin{cases} A[3] = -4 \\ A[2] + A[3] = 2 \\ A[1] + A[2] + A[3] = -8 \end{cases}$$



All sub arrays
ending at
Index 4

$$\text{MaxSum}[4] = \text{Max} \begin{cases} A[4] = 1 \\ A[3] + A[4] = -3 \\ A[2] + A[3] + A[4] = -1 \\ A[1] + A[2] + A[3] + A[4] = -7 \end{cases}$$

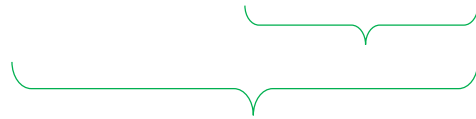
Can you divide the problem into subproblems such that solution to a bigger subproblem uses solution from smaller subproblem

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



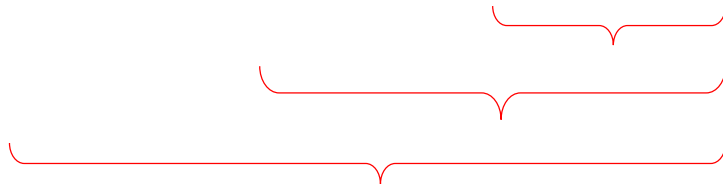
All sub arrays ending at index 1

$$\text{MaxSum}[1] = A[1] = -6$$



All sub arrays ending at index 2

$$\text{MaxSum}[2] = \text{Max} \begin{cases} A[2] = 2 \\ A[1] + A[2] = -4 \end{cases}$$



All sub arrays ending at index 3

$$\text{MaxSum}[3] = \text{Max} \begin{cases} A[3] \\ A[2] + A[3] \\ A[1] + A[2] + A[3] \end{cases}$$

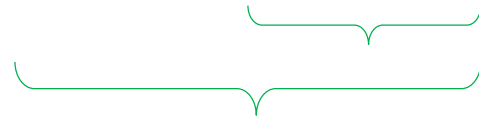
$$\text{MaxSum}[3] = \text{Max} \begin{cases} A[3] \\ \text{Max} \begin{cases} A[2] \\ A[1] + A[2] \end{cases} + A[3] \end{cases}$$

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



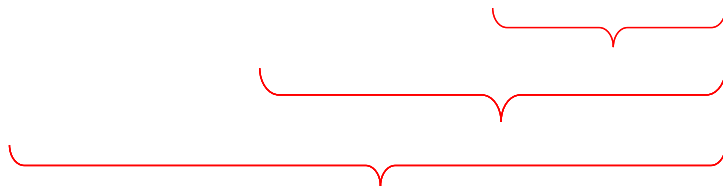
All sub arrays ending at index 1

$$\text{MaxSum}[1] = A[1] = -6$$



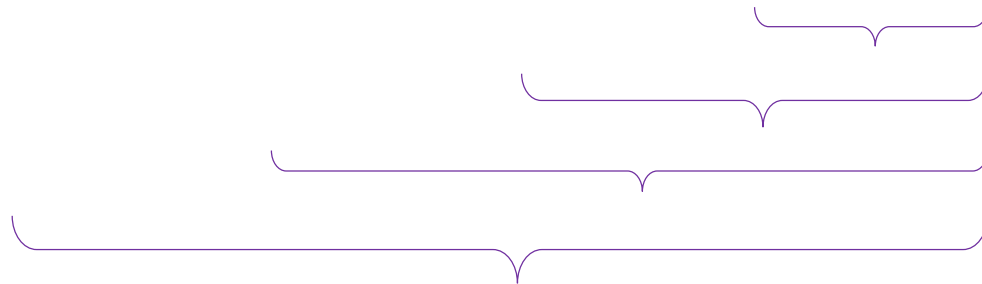
All sub arrays ending at index 2

$$\text{MaxSum}[2] = \text{Max} \begin{cases} A[2] = 2 \\ A[1] + A[2] = -4 \end{cases}$$



All sub arrays
ending at index 3

$$\text{MaxSum}[3] = \text{Max} \begin{cases} A[3] = -4 \\ A[2] + A[3] = 2 \\ A[1] + A[2] + A[3] = -8 \end{cases}$$



All sub arrays
ending at
Index 4

$$\text{MaxSum}[4] = \text{Max} \begin{cases} A[4] \\ A[3] + A[4] \\ A[2] + A[3] + A[4] \\ A[1] + A[2] + A[3] + A[4] \end{cases}$$

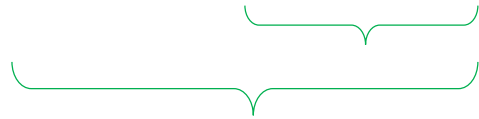
$$\text{MaxSum}[4] = \text{Max} \begin{cases} A[4] \\ \text{Max} \begin{cases} A[3] \\ A[2] + A[3] \\ A[1] + A[2] + A[3] \end{cases} + A[4] \end{cases}$$

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



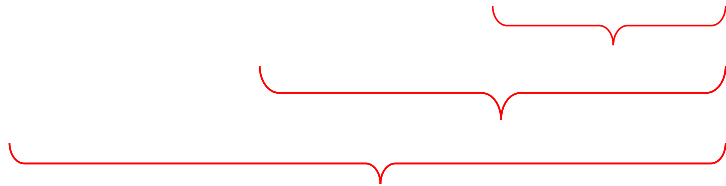
All sub arrays ending at index 1

$$\text{MaxSum}[1] = A[1] = -6$$



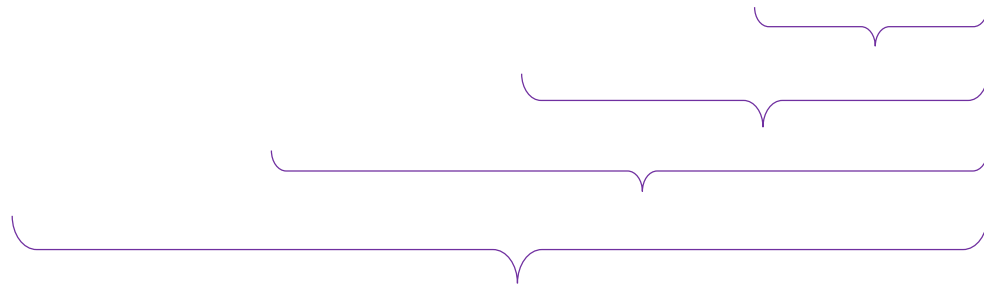
All sub arrays ending at index 2

$$\text{MaxSum}[2] = \text{Max} \begin{cases} A[2] = 2 \\ A[1] + A[2] = -4 \end{cases}$$



All sub arrays
ending at index 3

$$\text{MaxSum}[3] = \text{Max} \begin{cases} A[3] = -4 \\ A[2] + A[3] = -2 \\ A[1] + A[2] + A[3] = -8 \end{cases}$$



All sub arrays
ending at
Index 4

$$\text{MaxSum}[4] = \text{Max} \begin{cases} A[4] \\ A[3] + A[4] \\ A[2] + A[3] + A[4] \\ A[1] + A[2] + A[3] + A[4] \end{cases}$$

$$\text{MaxSum}[4] = \text{Max} \begin{cases} A[4] \\ \text{MaxSum}[3] + A[4] \end{cases}$$

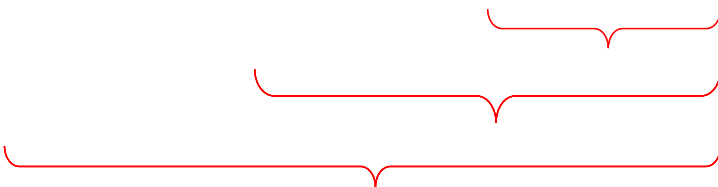
i	1	2	3	4	5	6	7	8
A[i]	-6	2	-4	1	3	-1	5	-1

 All sub arrays ending at index 1

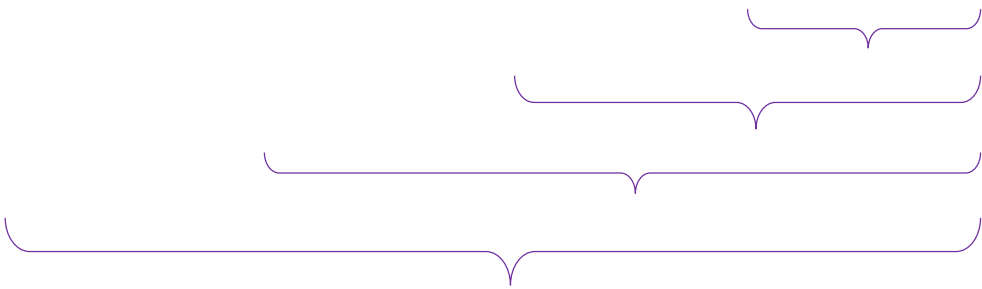
$\text{MaxSum}[1] = A[1] = -6$

 All sub arrays ending at index 2

$\text{MaxSum}[2] = \text{Max} \begin{cases} A[2] \\ \text{MaxSum}[1] + A[2] \end{cases}$

 All sub arrays ending at index 3

$\text{MaxSum}[3] = \text{Max} \begin{cases} A[3] \\ \text{MaxSum}[2] + A[3] \end{cases}$



$\text{MaxSum}[4] = \text{Max} \begin{cases} A[4] \\ \text{MaxSum}[3] + A[4] \end{cases}$

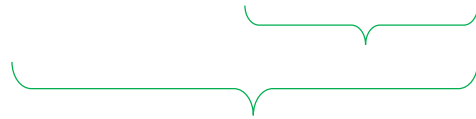
$\text{MaxSum}[5] = \text{Max} \begin{cases} A[5] \\ \text{MaxSum}[4] + A[5] \end{cases}$

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



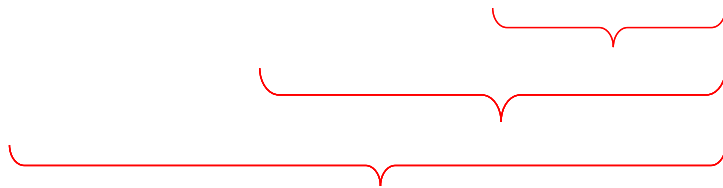
All sub arrays ending at index 1

$$\text{MaxSum}[1] = A[1] = -6$$



All sub arrays ending at index 2

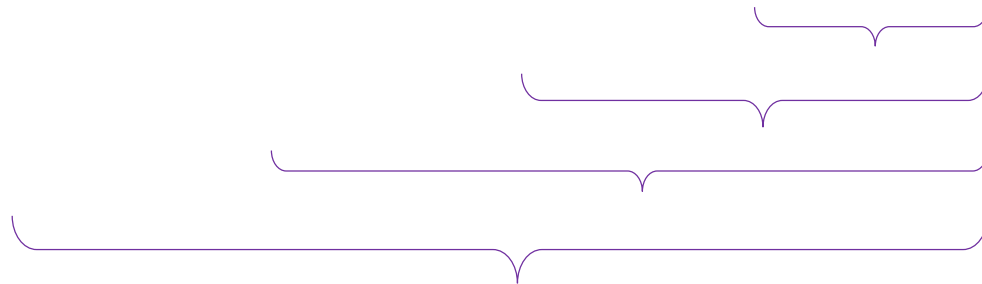
$$\text{MaxSum}[2] = \text{Max} \begin{cases} A[2] \\ \text{MaxSum}[1] + A[2] \end{cases}$$



All sub arrays

ending at index 3

$$\text{MaxSum}[3] = \text{Max} \begin{cases} A[3] \\ \text{MaxSum}[2] + A[3] \end{cases}$$



$$\text{MaxSum}[4] = \text{Max} \begin{cases} A[4] \\ \text{MaxSum}[3] + A[4] \end{cases}$$

Recursion for DP Solution

$$\text{MaxSum}[i] = \text{Max} (A[i] + \text{MaxSum}[i-1], A[i])$$

$$\text{MaxSum}[5] = \text{Max} \begin{cases} A[5] \\ \text{MaxSum}[4] + A[5] \end{cases}$$

Brute Force $O(n^2)$ Solution

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

DP $O(n)$ Solution

```
1.  MaxSubArraySum(A,n)
2.  {
3.    globalSum = A[1]
4.    MaxSum[1] = A[1]
5.    for (i = 2 to n)
6.    {
7.      if (MaxSum[i-1] + A[i] > A[i] )
8.        MaxSum[i] = MaxSum[i-1] + A[i]
9.      else
10.        MaxSum[i] = A[i]
11.      globalSum = Max (globalSum, MaxSum[i])
12.    }
13.    return globalSum
14. }
```

O(n) Dynamic Programming Algorithm (Kadane's Algorithm)

1. MaxSubArraySum(A,n)

2. { globalSum = A[1]

3. MaxSum[1] = A[1]

4. for (i = 2 to n)

5. if (MaxSum[i-1] + A[i] > A[i])

6. MaxSum[i] = MaxSum[i-1] + A[i]

7. else

8. MaxSum[i] = A[i]

9. If (globalSum < MaxSum[i])

10. globalSum = MaxSum[i]

11. globalEnd = i

12. return globalSum

13.}

Recursion for DP Solution

MaxSum[i] = Max (A[i] + MaxSum[i-1] , A[i])

O(n) Dynamic Programming Algorithm (Kadane's Algorithm)

```
1. MaxSubArraySum(A,n)
2. {  globalSum = A[1]
3.    MaxSum[1] = A[1]
4.    for (i = 2 to n)
5.        if (MaxSum[i-1] + A[i] > A[i] )
6.            MaxSum[i] = MaxSum[i-1] + A[i]
7.        else
8.            MaxSum[i] = A[i]
9.        If (globalSum < MaxSum[i])
10.            globalSum = MaxSum[i]
11.            globalEnd = i
12.    return globalSum
13.}
```

Task 1:

This algorithm keeps track of end of Max sub array in line 11. Modify this algorithm to keep track of start of Max sub array

Task 2

- Dry run brute force $O(n^2)$ 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]	2	-4	3	4	-3	5	-5	6	-1

Task 3

- Dry run Kadane's algorithm on following array and show all working. Show all values of MaxSum[i] array.

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

Task 4

- Can you write the dynamic programming solution of this problem that takes $O(1)$ memory (without array of MaxSum) and $O(n)$ time?
- If yes, write the pseudocode.