1. Problem

- Largest Sum Contiguous Subarray

Input: An integer array.

Output: The sum of the contiguous subarray (containing at least one number) which has the largest sum.

Example: Input: A = [-2, 1, -3, $\boxed{4, -1, 2, 1,}$ -5, 4]

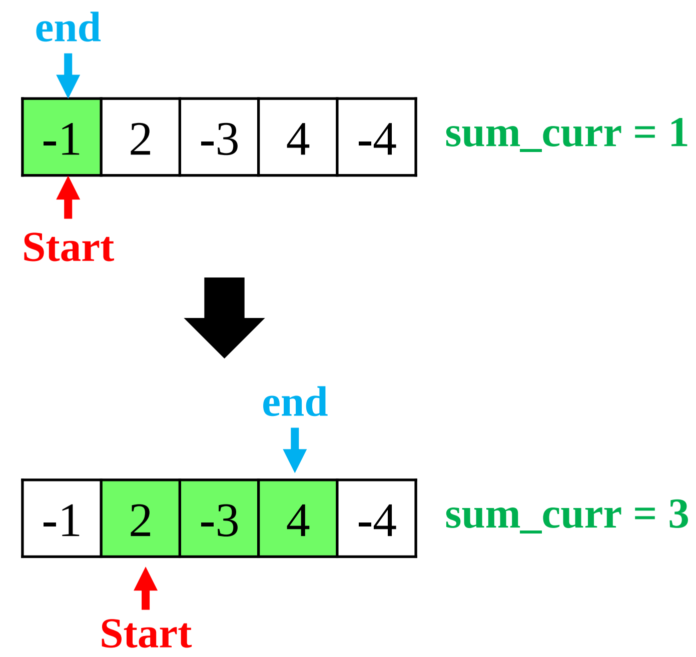
Output: sum_max = 4 – 1 + 2 + 1 = 6

1) Naive approach (Brute Force)

Pseudocode:

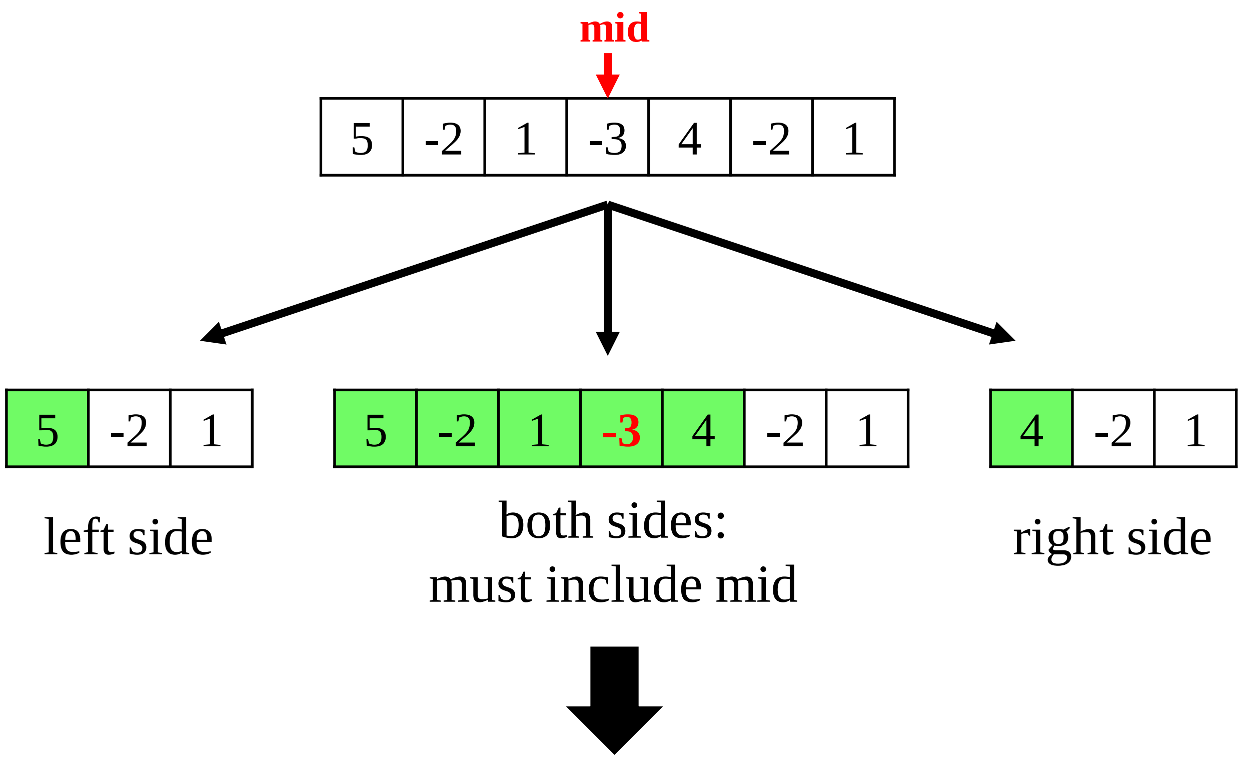
```
maxSubArray(A[0..(n - 1)])

Input: An array with n integers: A
Output: The max sum of contiguous subarray.
sum_max ← -infinity
for start ← 0 to n - 1 do
    for end ← start to n - 1 do
        sum_curr ← sum(A[start to end])
        sum_max ← max(sum_max, sum_curr)
return sum_max
```

Visualization:

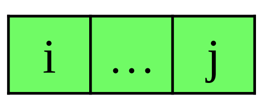


	Time complexity	Space complexity	How to get sum
Naive sum	$O(n^3)$	O(1)	iteratively
Cumulatively	$O(n^2)$	$O(n)$ or O(1)	using cumulative sum

2) Divide and conquer



return max(left, both, right)



Max sum subarray

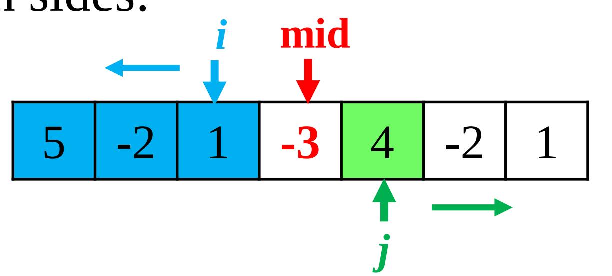
2) Divide and conquer

maxSubArray(A[0..(n - 1)])

Input: An array with n integers: A
Output: The max sum of contiguous subarray.

```
return findMax(A, 0, n - 1)
```

Both sides:



both_left = 4   both_right = 4

both_max = both_left + A[mid] + both_right

findMax(A, start, end)

Input: An array A, start index and end index
Output: The max sum of contiguous subarray.

```
mid ← (start + end) / 2
left_max ← findMax(A, left, mid - 1)
right_max ← findMax(A, mid + 1, right)
both_left ← 0; both_right ← 0; sum_curr = 0
for i ← mid - 1 to 0 do
    sum_curr ← sum_curr + A[i]
    both_left ← max(both_left, sum_curr)
sum_curr = 0
for j ← mid + 1 to n - 1 do
    sum_curr ← sum_curr + A[j]
    both_right ← max(both_right, sum_curr)
both_max ← both_left + A[mid] + both_right
return max(both_max, left_max, right_max)
```

2) Divide and conquer

Time complexity:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(\frac{n}{2}) + \mathrm{O}(n) & \text{if } n > 1 \end{cases}$$

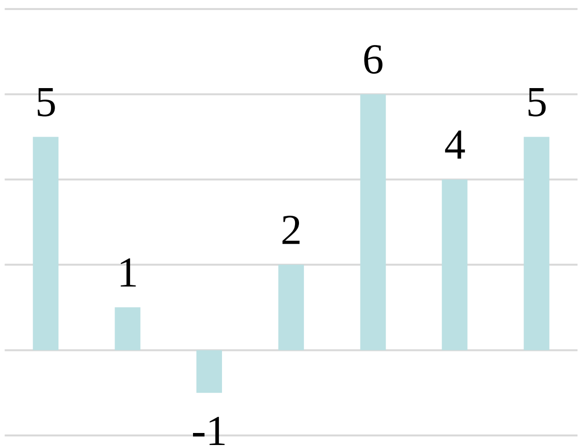
$$\text{Solving, } T(n) \in O(n \log n)$$

Space complexity:

Extra space is $\Theta(\log n)$ stack space for recursion

3) Kadane's Algorithm

A:	5	-4	-2	3	4	-2	1
Cumulative sum:	5	1	-1	2	6	4	5



A negative value in cumulative sum array is not worth keeping.

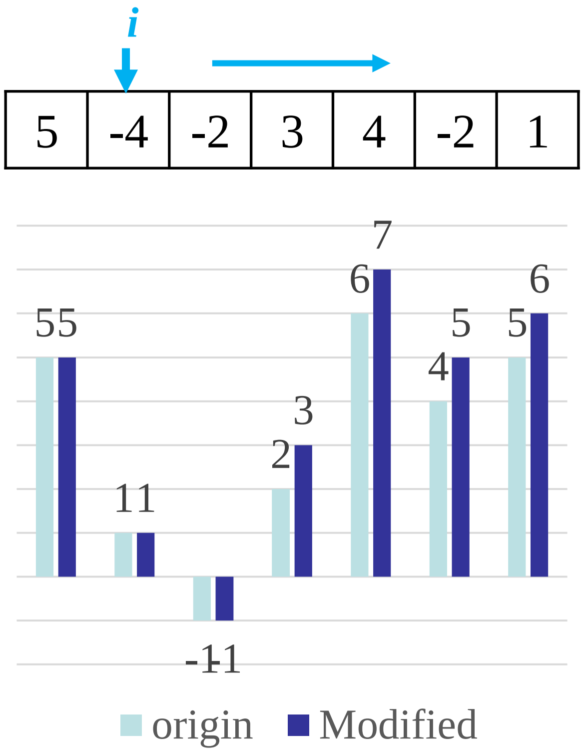
Once the cumulative sum becomes negative, reset it to 0.

3) Kadane's Algorithm

maxSubArray(A[0..(n - 1)])

Input: An array A, start index and end index
Output: The max sum of contiguous subarray.

```
sum_max ← A[0]; sum_curr ← A[0]
for i ← 1 to n - 1 do
    if sum_curr < 0 then
        sum_curr = A[i]
    else
        sum_curr = A[i] + sum_curr
    sum_max = max(sum_curr, sum_max)
return sum_max
```



Time complexity: $O(n)$

Space complexity: $O(1)$

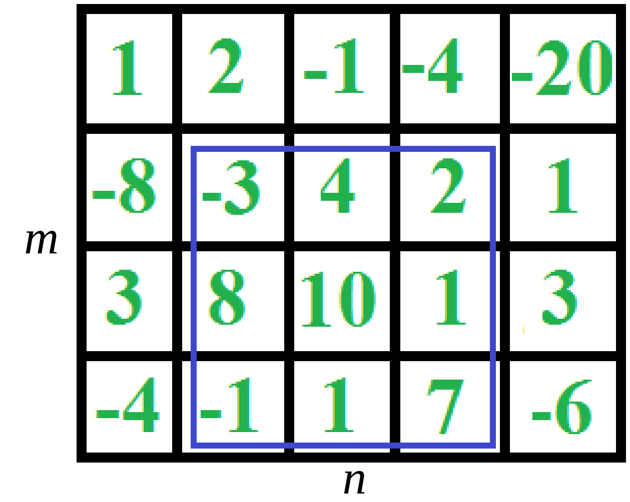
	Time complexity	Space complexity
Best Naive brute force	O(n^2)	O(1)
Divide and conquer	O($n$log$n$)	O(log$n$)
Kadane's Algorithm	O(n)	O(1)

- Largest sum rectangle in a 2D matrix

Input: An m by n 2D integer matrix.

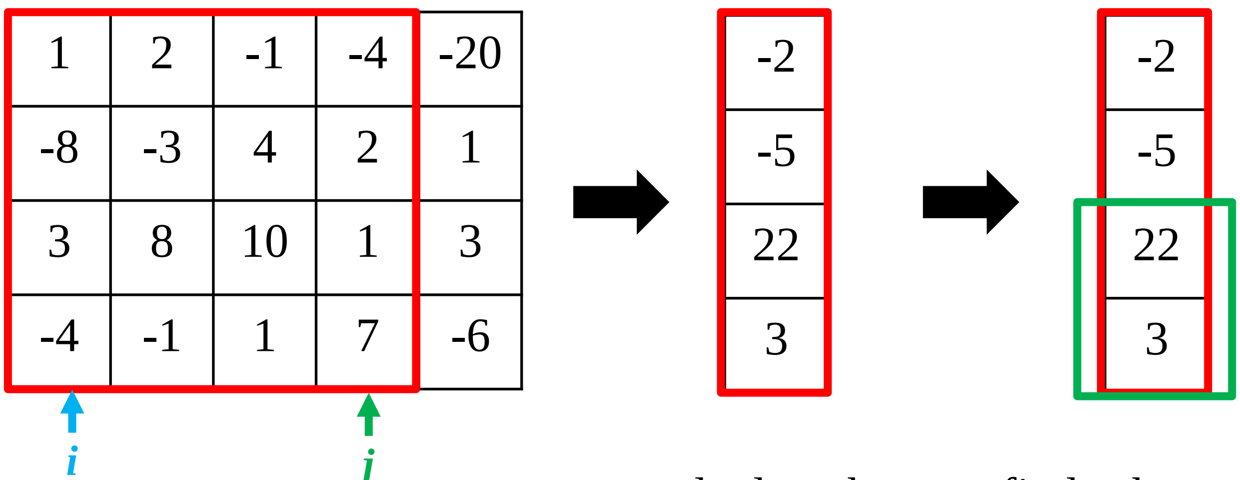
Output: The maimum sum of the contiguous submatrix

Worst brute force: $O(m^3n^3)$



https://www.geeksforgeeks.org/maximum-sum-rectangle-in-a-2d-matrix-dp-27/

Kadane's Algorithm:



Iterate and fix columns

calculate the sum of rows

find subarray with maximum sum

Time complexity: $O(mn^2)$