Exp - 2C

Max - Subarray Sum

* Array = 1, -3, 2, 1, -1

1) Brute

→ computes sum for all possible subarrays & updates maxSum if thisSum > maxSum

→ $O(n^2)$; simple but slow

→ [1] ⟹ 1 , [-3, 2] ⟹ -1 , [-3, 2, 1] ⟹ 0, [2, 1] ⟹ 3
[1, -3] ⟹ -2 , [-3, 2, 1, -1] ⟹ -1, [2] ⟹ 2, [2, 1, -1] ⟹ 2
[1, -3, 2] ⟹ 0 , [1] ⟹ 1, [1, -1] ⟹ 0
[1, -3, 2, 1] ⟹ 1
[-3] ⟹ -3

Max Sum ⟹ 3

2) Divide & Conquer

→ $O(n \log n)$; recursive

→

[1, -3] | [2, 1, -1]

Trace recursion:

[1] ⟹ 1 , [2] ⟹ 2
[-3] ⟹ -3 , [1] ⟹ 1 , [-1] ⟹ -1

3) Kadane's

→ $O(n)$; most eff.

i = 0 : maxSum = 1
i = 1 : thisSum = -2, m_S = 1 ⟹ thisSum = 0
i = 2 : thisSum = 2, m_S = 2
i = 3 : thisSum = 3, m_S = 3 ⟹ [2, 1]
i = 4 : thisSum = 2, m_S = 3