Maximum subarray sum

example: -2, 1, -3, 4, -1, 2, 1, -5, 4

sum = 6

Solution analytics:

brute force:

possible subarrays?

1, 2, 3, 4:

1
1, 2
1, 2, 3
1, 2, 3, 4
2
2, 3
2, 3, 4
3
3, 4
4

1
2
3

i = 0 (1)
- j = 0 => (1) 0 + 1 = 1
- j = 1 => (1, 2) 1 + 2 = 3
- j = 2 => (1, 2, 3) 3 + 3 = 6
- j = 3 => (1, 2, 3, 4) 6 + 4 = 10

i = 1 (2)
- j = 1 => (2) 0 + 2 = 2
- j = 2 => (2, 3) 2 + 3 = 5
- j = 3 => (2, 3, 4) 5 + 4 = 9

i = 2 (3)
- j = 2 => (3) 0 + 3 = 3
- j = 3 => (3, 4) 3 + 4 = 7

i = 3
- j = 3 => (3) = 0 + 3 = 3

max

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxSum = INT_MIN;
        for (int i = 0; i<nums.size();i++) {
            int sum = 0;
            for (int j = i; j<nums.size(); j++) {
                int value = nums[j];
                sum = sum + value;
                if(sum > maxSum) {
                    maxSum = sum;
                }
            }
        }
        return maxSum;
    }
};
```

complexity: $O(n^2)$

O(n)? Kadane's algorithm.

Current Sum: the best sum ending at the current position

Max Sum: the best found so far

Loop over numbers

you decide • it is better to start fresh from this element
• or continue adding to the current sum (subarray)

Decision:

if adding a number drops the sum below that number itself. we start a new sum (new subarray).

we always compare between sum and nums[i].

Manual execution (example): 3, -2, 5, -1

currentSum = maxSum = first el = 3

i=1, el = -2

sum = cSum - 2 = 3 - 2 = 1

1 (sum) > -2 => no => cSum = sum = 1

cSum (1) < maxSum (3), no affect

i = 2, el = 5

sum = cSum + 5 = 1 + 5 = 6

6 (sum) > 5 => no => cSum = sum = 6

cSum (6) > maxSum (3), affect

maxSum = 6

i = 3 = el = -1

sum = cSum - 1 = 6 - 1 = 5

5 (sum) > -1 => no => cSum = sum = 5

cSum (5) < maxSum (6)

no affect

if sum (nums[i]), restart cSum with nums[i] (not in this example)

```
class Solution {
public:
    int maxSubArray(vector<int> &nums) {
        int maxSum = nums[0];
        int currentSum = nums[0];
        for (int i = 1; i < nums.size(); i++) {
            int sum = currentSum + nums[i];
            if (sum < nums[i]) {
                // the added number drops the sum, so we drop the subarray and start a fresh one
                currentSum = nums[i];
            } else {
                // we kepp the subarray
                currentSum = sum;
            }
            if (currentSum > maxSum) {
                maxSum = currentSum;
            }
        }
        return maxSum;
    }
};
```