Q8 Maximum subarray sum. (Kadane's Algorithm)

i/p → {5, 4, -1, 7, 8}

o/p → 23

Sub arrays can be defined as part of array which is contiguous.

All possible sub-arrays

{5} → 5

{5,4} → 9

{5,4,-1} → 8

{5,4,-1,7} → 15

{5,4,-1,7,8} → 23] maximum sum.

{4} → 4

{4,-1} → 3

{4,-1,7} → 10

{4,-1,7,8} → 18

{-1} → -1

{-1,7} → 6

{-1,7,8} → 14

{7} → 7

{7,8} → 15

{8} → 8

The above approach is the brute force as we are finding all the possible sub-arrays. We need to find the optimized approach as here 3 loops would be required & hence $O(n^3)$ is the time complexity. We need to find the solution in O(n) approach & here Kadane's algorithm is required.

Q8 Maximum subarray sum. (Kadane's Algorithm)

i/p → {5, 4, -1, 7, 8}

o/p → 23

Sub arrays can be defined as part of array which is contiguous.

All possible sub-arrays

{5} → 5

{5,4} → 9

{5,4,-1} → 8

{5,4,-1,7} → 15

{5,4,-1,7,8} → 23] maximum sum.

{4} → 4

{4,-1} → 3

{4,-1,7} → 10

{4,-1,7,8} → 18

{-1} → -1

{-1,7} → 6

{-1,7,8} → 14

{7} → 7

{7,8} → 15

{8} → 8

The above approach is the brute force as we are finding all the possible sub-arrays. We need to find the optimized approach as here 3 loops would be required & hence $O(n^3)$ is the time complexity. We need to find the solution in O(n) approach & here Kadane's algorithm is required.

Dry run + Approach

{-2, 1, -3, 4, -1, 2, 1, -5, 4}

i ↗

Initially sum = 0, maxi = -2 ← arr[0]

1) i = 0
sum = sum + arr[i]
sum = 0 + (-2) = -2
Now update maxi
maxi = max(sum, maxi) = max(-2, -2) = -2
Now as sum < 0, make sum = 0.

2) i = 1
sum = sum + arr[i]
sum = 0 + 1 = 1 ← sum
maxi = max(sum, maxi) = max(-2, 1) = 1
↳ maxi

Sum < 0 → False & hence do nothing

3) i = 2
sum = sum + arr[i]
sum = 1 + (-3) = -2
maxi = max(1, -2) = 1
sum < 0 → True & hence sum = 0

4) i = 3
sum = 0 + 4 = 4
maxi = max(1, 4) = 4
sum < 0 → False & hence do nothing.

5) i = 4

sum = 4 + (-1) = 3
maxi = max (4, 3) = 4
sum < 0 → False & hence do nothing

6) i = 5
sum = 3 + 2 = 5
maxi = max (4, 5) = 5
sum < 0 → False & hence do nothing

7) i = 6
sum = 5 + 1 = 6
maxi = max (5, 6) = 6
sum < 0 → False & hence do nothing

8) i = 7
sum = 6 - 5 = 1
maxi = max (6, 1) = 6
sum < 0 → False & hence do nothing

9) i = 8
sum = 1 + 4 = 5
maxi = max (6, 5) = 6
sum < 0 → False & hence do nothing.

Hence we have traversed the whole array & at last just return maxi.

Steps
1) sum = 0, maxi = nums[0];
2) Traverse array
   * sum = sum + arr[i]
   * maxi = max (sum, maxi)
   * sum < 0 → sum = 0

3) Return maxi.

Code

```
int maxSubArraySum (vector <int> &nums) {
        // Initial steps
    int sum = 0;
    int maxi = nums [0];
    // Traversing array
    for (int i=0 ; i< nums.size() ; i++) {
        // Step-2 (1st step)
        sum = sum + nums [i];
        // Step-2 (2nd step)
        maxi = max (sum, maxi);
        // Step-1 (3rd step)
        if (sum < 0) {
            sum = 0;
        }
    } // Step-3
    return maxi;
}
```

Time complexity = O(n)
Space complexity = O(1)