Technique: Kadane's Algorithm

Level: Medium

Given an array of integers that can be both +ve and -ve, find the contiguous subarray with the largest sum.

For example: [1,2,-1,2,-3,2,-5] -> first 4 elements have the largest sum. Return (0,3)

Questions to Clarify:
Q. How do you want the output?
A. Return the value of the maximum sum.

Q. Do empty arrays count as a subarray?
A. No, the subarray should have at least 1 element.

Q. But what if the input array is empty or null?
A. Throw an exception.

Solution:
The brute force solution is to iterate through each subarray by using two for loops, and calculating the sum of each subarray.

```
for i -> 0 to a.length
    sum = 0
    for j -> i to a.length
          sum = sum + a[j]
          check if sum is max and save if so
```

This takes $O(n^2)$ time and $O(1)$ space. It is good to mention this approach to the interviewer.

We can however, do this problem in $O(n)$ time and $O(1)$ space.

We use Kadane's algorithm. If we know the max sum ending at *a[i - 1]*,
we can calculate the max sum ending at *a[i]*.

```
max_sum_ending_at[i] = Max(max_sum_ending_at[i-1] + a[i], a[i])
```

This is because there are only 2 candidates for max sum ending at *i* - either *a[i]*
as a single array, or *a[i]* with the array ending at *a[i - 1]*.

This is an example of building on top of a subproblem. We know the solution of subproblem *a[i]*. We used that information to calculate the solution for subproblem *a[i+1]*. This also forms the basis for Dynamic Programming, which we cover in later sections.

Pseudocode:

```
(Note: Never write pseudocode in an actual interview. Unless you're writing a
few lines quickly to plan out your solution. Your actual solution should be in
a real language and use good syntax.)

result = a[0], maxEndingHere = a[0]
for i -> 1 to a.length - 1
    maxEndingHere = max(maxEndingHere + a[i], a[i])
    result = max(result, maxEndingHere)
return result
```

Test Cases:
Edge Cases: empty array, null array
Base Cases: single element (+ve, 0, -ve), two elements
Regular Cases: all -ve, all +ve, mix -ve and +ve, all 0s

Time Complexity: O(n)

Space Complexity: O(1)

```
/*
 * Brute force solution - using nested loop
 */
public static Integer maximumSumSubarrayBruteForce(int[] a) {
    if (a == null || a.length == 0)
        throw new IllegalArgumentException("Input array is empty or null");

    int maxSum = a[0];

    for (int i = 0; i < a.length; i++) {
        int sum = 0;
        for (int j = i; j < a.length; j++) {
            sum = sum + a[j];
            maxSum = Math.max(maxSum, sum);
        }
    }

    return maxSum;
}

/*
 * Kadane's algorithm solution
 */
public static Integer maximumSumSubarray(int[] a) {
    if (a == null || a.length == 0)
        throw new IllegalArgumentException("Input array is empty or null");
```

```
    int maxSum = a[0], maxEndingHere = a[0];

    for (int i = 1; i < a.length; i++) {
        maxEndingHere = Math.max(maxEndingHere + a[i], a[i]);
        maxSum = Math.max(maxSum, maxEndingHere);
    }

    return maxSum;
}
```