Technique #2: Sliding Window using Two Pointers

Level: Medium

Given an array of positive integers, find a subarray that sums to a given number X.

For e.g, input = [1,2,3,5,2] and X=8, Result = [3,5] (indexes 2,3)

Questions to Clarify:
Q. How to return the result?
A. Return the start and end indices of the subarray.

Q. What to return if the array is empty or null?
A. Return null.

Q. What to return if no subarray is found?
A. Return null.

Q. What to do if there are multiple subarrays?
A. Return any one.

Solution:
The brute force algorithm (going through each subarray) takes O(n^2) time and O(1) space.
We can solve this problem with O(n) time and O(1) space.

We use the technique of using a sliding window and shifting it forward.
We keep two pointers, *start* and *end*. They both start with *0*. We keep track of the sum of subarray *[start..end]*.
If the sum is less than *X*, we advance *end* by 1 and add *a[end]* to the sum. If sum is greater than *X*, we advance start by 1 and subtract *a[start]* from the sum. Keep in mind *start* is always behind or equal to *end*.

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.)

start = 0, end = 0, sum = a[0]

while start is within the array
     if start inched ahead of end:
         bring end back to start, update sum to a[start]

     if (sum < X)
         if cannot expand end further:
             break out, we're done
```

```
        move end forward, increase sum by a[end]
    else if (sum > X)
        reduce sum by a[start], move start forward
    else  // sum == X
        return [start,end]

return null
```

<u>Test Cases:</u>
Edge Cases: empty array, null array
Base Cases: single element (more/less/equal to X)
Regular Cases: two elements, no sum equals to X, etc.

<u>Time Complexity: O(n)</u>

<u>Space Complexity: O(1)</u>

```
public static Pair<Integer> subarraySum(int[] a, int target) {
    if (a == null || a.length == 0)
        return null;

    int start = 0, end = 0, sum = a[0];
    while (start < a.length) {
        if (start > end) { // start inched forward, bring end back to start
            end = start;
            sum = a[start];
        }

        if (sum < target) { // expand to right
            if (end == a.length - 1)
                break; // reached end, cannot expand further

            end++;
            sum = sum + a[end];
        } else if (sum > target) { // contract from left
            sum = sum - a[start];
            start++;
        } else {
            return new Pair<Integer>(start, end);
        }
    }

    return null;
}
```