167. Two sum II - Input array is sorted.

Ip: numbers = [2, 7, 11, 15] target = 9

Op: [1, 2]

Explanation: The sum of 2 and 7 is 9. Therefore, index1 = 1, index2 = 2 we return [1, 2].

Brute force: (Beginner):

```
class Solution {
    public int[] twoSum (int[] numbers, int target) {
        for (int i = 0; i < numbers.length ; i++) {
            for (int j = i+1 ; j < numbers.length ; j++) {
                if (numbers[i] + numbers[j] == target) {
                    return new int[] {i+1, j+1} ;
                                    // 1:- indexed.
                }
            }
        }
        return new int[][];
    }
}
```

Time → $O(n^2)$ space → $O(1)$

optimal approach (Two pointers)

* left → start of array
* right → end of array

Logic:

1. If numbers[left] + numbers[right] == target → answer found
2. If the sum is too big, move right leftward
3. If the sum is too small, move left rightward.

```
class Solution {
public int[] twoSum(int[] numbers, int target) {
    int left = 0;
    int right = numbers.length - 1;
while (left < right) {
    int sum = numbers[left] + numbers[right];
if (sum == target) {
    return new int[] {left+1, right+1}; // 1-indexed.
} else if (sum < target) {
    left++;
} else {
    right--;
    }
  }
  return new int[] {};
 }
}
```

Dry Run:

numbers = [2, 7, 11, 15] target = 9

1. left = 0(2), right = 3(15) → sum = 17 > 9 →
   move right → 2
2. left = 0(2), right = 2(11) → sum = 13 > 9 →
   move right → 1
3. left = 0(2), right = 1(7) → sum = 9 →
   → answer = [1, 2].