✓ 11. Container With Most Water Medium

```
int maxArea(int[] heights):
    left = 0, right = len(heights) - 1, max = 0 // set initial pointers & initial max
    WHILE left < right: // While the pointers do not meet
        diff = right - left, cap = min(heights[left], heights[right])
        amount = diff * cap // calculate amount
        IF amount > max: max = amount // update max if necessary
        IF heights[left] < heights[right]: left++ // increment left if right is greater
        ELSE: right-- // decrement right otherswise
    Return max
```

Complexity: O(n)

✓ 713. Subarray Product Less Than k Medium

```
int countSubarrays(int[] nums, int k):
    prev = 1, beg = 0, total = 0 // default previous product = 1, beg of window = 0
    FOR i=0; i < len(nums); i++:
        IF nums[i] >= k: // num[i] ≥ k => product * nums[i] ≥ k
            prev = 1, beg = i+1 // reset prev product ∧ beg to next
            continue // skip rest of this iteration
        WHILE nums[i] * prev ≥ k: prev /= nums[beg++] // chop window until
                                                      // product < k
        total += i - beg + 1 // add window length + 1 (for itself) to total
        prev *= num[i] // update prev product
    Return total
```

Complexity O(n)

✓ 438. Find All Anagrams in a String Easy

```
boolean isAnagram(Map<Integer> one, two): // helper function to check if is
                                          // anagram
    return one.equals(two)

List<Integer> findAnagrams(String s, p):
    ret = [] // initialize return variable
    IF len(s) < len(p): return ret // Not possible to contain anagrams
    Map<Integer> letters = {}, temp = {} // define the maps
    FOR char c IN s:
        letters[c] = letters[c]+1 IF c IN letters ELSE 0 // map each letter
                                                         // to # of occurences
    FOR i=0; i < len(p); i++:
        temp[s[i]] = temp[s[i]] + 1 IF s[i] IN temp ELSE 0 // check first
                                                           // window
    FOR i=0; i < len(s) - len(p) +1; i++: // check all windows by sliding
        IF isAnagram(letters, temp):
            ret.add(i) // add index if is anagram
        temp[s[i]]-- // remove first element in the window
        IF i < len(s) - len(p): // check if we have not reached last window
            temp[s[i+len(p)]]++ // add next element to end of window
    Return ret // return list containing all the indices
```

Complexity: O(n)

✓ 209. Minimum Size Subarray Sum Medium

```
int countSubarrays(int[] nums, int target)
    min = MAX, sum = 0, beg = 0 // initial min is constant MAX
    FOR i=0; i < len(nums); i++:
        IF nums[i] ≥ target: return 1 // Return 1 is a single greater or equal
                                      // element is found
        sum += nums[i] // update sum
        IF sum < target: continue // skip rest of this iteration if sum < target
        cur = i - beg + 2 // length of window + 1 (to account for upcoming decrement)
        WHILE sum ≥ target:
            sum -= nums[beg++], cur-- // chop window & decrement cur
        min = cur < min ? cur : min // update min if necessary
    Return min == MAX ? 0 : min // return min if min changed, otherwise 0
```

Complexity O(n)