✓844. Backspace String Compare Easy

```
boolean backspaceCompare (String s, String t):
    char[] one = s.toCharArray(), two = t.toCharArray() // convert to arrays
    int one_p = 0, two_p = 0 // set pointers respectively
    FOR i=0; i<len(one); i++:
        IF one[i] != '#':
            one[one_p++] = one[i] // store char & increment pointer
        ELSE IF i!=0: one_p-- // decrement pointer ≥0

    FOR i=0; i<len(two); i++:
        IF two[i] != '#':
            two[two_p++] = two[i] // store char & increment pointer
        ELSE IF i!=0: two_p-- // decrement pointer ≥0
    IF one_p != two_p: // different lengths => not same string
        Return False
    FOR i=0; i<one_p; i++:
        IF one[i] != two[i]: // different letter @ same index => not same
            Return False
    Return True // If nothing contradicts => same String
```

Complexity O(n)

✓ 986. Interval List Intersections Medium

```
int[][] intervalIntersection(int[][] first, int[][] second):
    List<int[]> ret = []  // initialize return variable
    int i=0, j=0  // pointer to each list of intervals
    WHILE i < len(first) AND j < len(second):
        lo = max(first[i][0], second[j][0])  // Find front cap index
        hi = min(first[i][1], second[j][1])  // Find end cap index
        IF lo ≤ hi:  // if the two indices meet or can wrap
            ret.add({lo, high})  // Add this pair of indices
        IF first[i][1] < second[j][1]:
            i++  // increase first pointer if second dominates
        ELSE j++  // increase second pointe if first dominates
    Return ret.toArray(new int[len(ret)][])
```

Complexity O(n)

✓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)