Two-pointer technique

Problems that involve two pointers [list, i, j] which otherwise would be very slow...

Variation 1

One slow-runner and the other fast-runner

```
... f() {
    ...
    while (i < n) {
        if (some condition) {
            j += 1;
        }
        i += 1;
    }
    ...
}
```

Variation 2

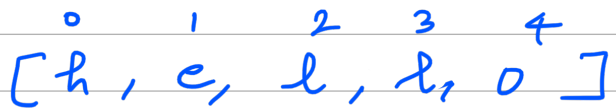
One pointer starts from the beginning while the other pointer starts from the end...

They move towards each other until they both meet.

```
... f() {
    ...
    i = 0, j = n-1;
    while (i < j) {
        i += 1;
        j -= 1;
    }
    ...
}
```

Classical problem of reversing a string-

```
void reverse (char[] str) {
    int half = str.length/2;
    for(int i=0; i< half; i+=1) {
        swap(str, i, n-i-1);
    }
}
```




```
void swap(char[] str, int i, int j) {
    char temp = str[i];
    str[i] = str[j];
    str[j] = temp;
}
```

Using two-pointer technique-

```
void reverse (char[] str) {
    int i=0, j = str.length-1;
    while (i< j) {
        swap(str, i, j);
        i+=1;
        j-=1;
    }
}
```

① → Print squared sorted array-

Arr[]
N
-ve and +ve integers
print the sorted output

[-7, -2, 3, 4, 6]
49 4 9 16 36

[4, 9, 16, 36, 49]

#977 Leetcode

```
int[] sortedSquares(int[] nums) {

    int i=0,
        j= nums.length-1;

    int[] result = new int[nums.length];

    int k = nums.length -1;

    while (i<=j) {

        if (Math.abs(nums[i])>
            Math.abs(nums[j])) {

            result[k] = nums[i] x
                        nums[i];

            i+=1;

        } else {
```

```
result[k] = nums[j] *
            nums[j];
        j--;
    }
    k++;
}
return result;
}
```

T = O(n) n is the length of the array
S = O(n)

(2) → Longest distinct characters in a string —
#3 leetcode

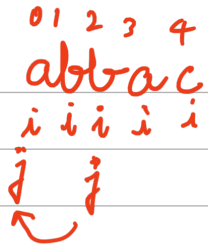
S = abcabcbb → 3 | S = bbbbbb → 1 | S = pwwkew → 3

- longest part without a repeating character

```
0 1 2 3 4 5 6 7
a b c a b c b b
i i i i i i i i
  j j j j j
```

Map
- Collection of key-value pairs

```
{
  a : 0,
  b : 1,
  c : 1,
```



{
a: 0,
b: 2,
c: 4
}

```
int lengthOfLongestSubstring(String s) {
    if (s == null || s.length() == 0) {
        return 0;
    }

    Map<Character, Integer> map = new HashMap();
    int j = 0;

    int maxLen = Integer.MIN_VALUE;

    for (int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);
        if (map.containsKey(c)) {

            if (map.get(c) >= j) {

                j = map.get(c) + 1;

            }
        }

        maxLen = Math.max(maxLen,
                          i - j + 1);

        map.put(c, i);

    }
    return maxLen;
}
```

T: O(N)
S: O(N)

③ → Sum subarray — → arr of all positive integers

	0	1	2	3	4
arr:	2	3	2	5	5

sum = 12

subarray whose sum >= 12;
get its length

If there are multiple such subarrays;
take the minimum length.

#209 leetcode

nums = [2, 3, 1, 2, 4, 3], target = 7
(i, j)

nums = [2, 3, 1, 2, 4, 3], target = 7 ✓

```
int minSubArrayLen(int target, int[] nums) {
    int i = 0,
        j = 0,
        sum = 0;
    int minLen = Integer.MAX_VALUE;
    while (i < nums.length) {

        sum += nums[i];

        while (sum >= target) {
```

```
                minLen = Math.min(minLen,
                                  i-j+1);

                sum -= nums[j];

                j += 1;

            }

            i += 1;

        }

        return  minLen == Integer.MAX_VALUE ?
                0 : minLen;

    }
```

T = O(N)
S = O(1)

④ Remove duplicates from sorted array—

#26

nums = [0, 0, 1, 1, 1, 2, 2, 3, 3, 4]

→ 5 0 1 2 3 4

```
int removeDuplicates(int[] nums) {
    int i = 0;
    while (i < nums.length) {
```

```
while (i < nums.length-1 &&
       nums[i] == nums[i+1]) {

    i += 1;

}

swap(i, j, nums);

j += 1;

i += 1;

    }

    return j;

}
```

T = O(N)
S = O(1)

⑤ Input array is sorted - Two sum - #167

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

output → [1, 2]

index - 1

nums = [2, 3, 4] target = 6

output → [1, 3]

```
int[] twoSum(int[] numbers, int target) {

    int left = 0,
        right = numbers.length - 1;

    while (left < right) {

        int sum = numbers[left] +
                  numbers[right];

        if( sum == target ) {
            return new int[] {
                left + 1,
                right + 1
            };
        } else if (sum > target) {
            right -= 1;
        } else {
            left += 1;
        }

    }
    return new int[] {
        -1, -1
    };

}
```

T = O(N)
S = O(1)

⑥ Valid palindrome –

s = "A man, a plan, a canal: Panama"
true

amanaplanacanalpanama

```
boolean isPalindrome(String s) {

    int left = 0;
        right = s.length()-1;

    while (left < right) {

        Character leftChar =
            Character.toLowerCase(s.charAt(left));

        boolean isLeftValid =
            Character.isLetterOrDigit(leftChar);

        rightChar

        if( isLeftValid && isRightValid)
        {
            if(leftChar != rightChar){

                return false;

            }

            left += 1;
            right -= 1;
        } else if (!isLeftValid) {
            left += 1;

        } else if (!isRightValid) {
```

```
            right --;
        }
    }
    return true;
}
```

T = O(N)
S = O(1)

#11 container with most water

#186 Reverse words in a string II

#189 Rotate Array

#238 Product of Array except itself