Single Pass vs Two Pass in Coding (with LeetCode Examples)

This document compares **Single Pass** vs **Two Pass** coding strategies with popular LeetCode problems. Each section includes explanations, Java code, and practice links.

■ LeetCode 169 – Majority Element

Problem: Find the element that appears more than n/2 times in an array.

■ **Two Pass Solution:** Count frequencies using HashMap, then check which has count > n/2.

```
class Solution { public int majorityElement(int[] nums) { Map map = new HashMap<>(); for (int num : nums) { map.put(num, map.getOrDefault(num, 0) + 1); } for (Map.Entry entry : map.entrySet()) { if (entry.getValue() > nums.length / 2) { return entry.getKey(); } } return -1; } }
```

■ **Single Pass Solution:** Boyer–Moore Voting Algorithm adjusts candidate in one traversal.

```
class Solution { public int majorityElement(int[] nums) { int candidate = nums[0], count = 1; for (int i = 1; i < nums.length; i++) { if (count == 0) { candidate = nums[i]; count = 1; } else if (nums[i] == candidate) { count++; } else { count--; } } return candidate; } }
```

■ **Practice Link:** https://leetcode.com/problems/majority-element/

■ LeetCode 238 – Product of Array Except Self

Problem: Return array where answer[i] = product of all nums except nums[i].

■ **Two Pass Solution:** First compute prefix products, then suffix products.

```
class Solution { public int[] productExceptSelf(int[] nums) { int n = nums.length; int[] res = new int[n]; res[0] = 1; for (int i = 1; i < n; i++) { res[i] = res[i - 1] * nums[i - 1]; } int suffix = 1; for (int i = n - 1; i >= 0; i--) { res[i] *= suffix; suffix *= nums[i]; } return res; } }
```

■ **Single Pass Solution:** Hard to do in single pass as both prefix and suffix are required.

```
// Typically not possible in strict single pass without extra space.
```

■ **Practice Link:** https://leetcode.com/problems/product-of-array-except-self/

■ Second Largest Element

Problem: Find the second largest element in an array.

■ **Two Pass Solution:** First pass: find max. Second pass: find largest < max.

```
class Solution { public int secondLargest(int[] nums) { int max = Integer.MIN_VALUE; for (int num : nums) max = Math.max(max, num); int second = Integer.MIN_VALUE; for (int num : nums) if (num != max) second = Math.max(second, num); return second; } }
```

■ **Single Pass Solution:** Maintain both max and secondMax in one traversal.

```
class Solution { public int secondLargest(int[] nums) { int max = Integer.MIN_VALUE, second = Integer.MIN_VALUE; for (int num : nums) { if (num > max) { second = max; max = num; } else if (num > second && num != max) { second = num; } } return second; } }
```

■ **Practice Link:** https://leetcode.com/problems/second-largest-digit-in-a-string/

■ LeetCode 75 – Sort Colors (Dutch National Flag)

Problem: Sort an array containing 0, 1, and 2 in-place.

■ **Two Pass Solution:** Count 0s, 1s, 2s (first pass) then rewrite array.

```
class Solution { public void sortColors(int[] nums) { int count0 = 0, count1 = 0, count2 = 0; for (int num : nums) { if (num == 0) count0++; else if (num == 1) count1++; else count2++; } int i = 0; while (count0-- > 0) nums[i++] = 0; while (count1-- > 0) nums[i++] = 1; while (count2-- > 0) nums[i++] = 2; } }
```

■ **Single Pass Solution:** Use Dutch National Flag Algorithm with three pointers (low, mid, high).

```
class Solution { public void sortColors(int[] nums) { int low = 0, mid = 0, high = nums.length - 1; while (mid <= high) { if (nums[mid] == 0) { swap(nums, low, mid); low++; mid++; } else if (nums[mid] == 1) { mid++; } else { swap(nums, mid, high); high--; } } } private void swap(int[] nums, int i, int j) { int temp = nums[i]; nums[i] = nums[j]; nums[j] = temp; } }
```

■ **Practice Link:** https://leetcode.com/problems/sort-colors/

■ LeetCode 41 – First Missing Positive

Problem: Find the smallest missing positive integer in an unsorted array.

■ **Two Pass Solution:** Use HashSet: First pass store all positives, second pass check from 1..n.

```
class Solution { public int firstMissingPositive(int[] nums) { Set set = new HashSet<>(); for (int num : nums) if (num > 0) set.add(num); for (int i = 1; i <= nums.length; i++) { if (!set.contains(i)) return i; } return nums.length + 1; } }
```

■ **Single Pass Solution:** Cyclic sort: place numbers at correct index, then scan once.

```
class Solution { public int firstMissingPositive(int[] nums) { int n = nums.length; for (int i = 0; i < n; i++) { while (nums[i] > 0 && nums[i] <= n && nums[nums[i]-1] != nums[i]) { swap(nums, i, nums[i]-1); } } for (int i = 0; i < n; i++) { if (nums[i] != i + 1) return i + 1; } return n + 1; } private void swap(int[] nums, int i, int j) { int temp = nums[i]; nums[i] = nums[j]; nums[j] = temp; } }
```

■ **Practice Link:** https://leetcode.com/problems/first-missing-positive/