Google SWE Intern 2026 Online Assessment - 20道编程题目详解

前言

Google作为全球顶级科技公司，其Software Engineer Intern岗位的Online Assessment以考查基础算法能力和编程思维著称。本题目集合基于历年真实面试经验和最新趋势，精心设计了20道适合实习生级别的编程题目。

这些题目涵盖了数据结构、算法、字符串处理、数组操作等核心知识点，难度从Easy到Medium为主，旨在全面考查实习生候选人的编程基础、逻辑思维和问题解决能力。每道题目都提供了详细的中英文结合解析，帮助求职者深入理解解题思路和最优解法。

题目分类和难度分布

难度分布

- **Easy**: 8道题目 (40%)
- **Medium**: 12道题目 (60%)

知识点覆盖

- **数组和字符串**: 8道题目
- **哈希表和集合**: 4道题目
- **双指针和滑动窗口**: 3道题目
- **递归和分治**: 3道题目
- **动态规划基础**: 2道题目

Problem 1: Two Sum (Easy)

Problem Statement

```
 Given an array of integers nums and an integer target, return indices of the
two
numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not
use the same element twice.

You can return the answer in any order.

Example 1:
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Example 2:
Input: nums = [3,2,4], target = 6
Output: [1,2]

Example 3:
Input: nums = [3,3], target = 6
Output: [0,1]

Constraints:
- 2 <= nums.length <= 10^4
- -10^9 <= nums[i] <= 10^9
- -10^9 <= target <= 10^9
- Only one valid answer exists.
```

解题思路 (Solution Analysis)

这是LeetCode第一题，也是最经典的**哈希表**应用题目。虽然看起来简单，但考查的是对时间复杂度优化的理解。

暴力解法： - 双重循环遍历所有可能的两个数字组合 - 时间复杂度：$O(n^2)$ - 空间复杂度：$O(1)$

哈希表优化： - 使用哈希表存储已遍历的数字和其索引 - 对于当前数字，查找`target - current`是否在哈希表中 - 时间复杂度：$O(n)$ - 空间复杂度：$O(n)$

代码实现 (Code Implementation)

```python
def twoSum(nums, target):
    """
    Find two numbers that add up to target.

    Args:
        nums: List[int] - Array of integers
        target: int - Target sum

    Returns:
        List[int] - Indices of the two numbers
    """
    # Hash map to store number -> index mapping
    num_to_index = {}

    for i, num in enumerate(nums):
        complement = target - num

        # Check if complement exists in hash map
        if complement in num_to_index:
            return [num_to_index[complement], i]

        # Store current number and its index
        num_to_index[num] = i

    # Should never reach here given problem constraints
    return []

# Alternative implementation with more explicit logic
def twoSumVerbose(nums, target):
    """
    More verbose implementation for better understanding.
    """
    seen = {}

    for current_index, current_num in enumerate(nums):
        needed = target - current_num

        if needed in seen:
            return [seen[needed], current_index]
        else:
            seen[current_num] = current_index

    return []

# Test cases
test_cases = [
    ([2, 7, 11, 15], 9),    # [0, 1]
    ([3, 2, 4], 6),         # [1, 2]
    ([3, 3], 6),            # [0, 1]
    ([-1, -2, -3, -4, -5], -8)  # [2, 4]
]

for nums, target in test_cases:
    result = twoSum(nums.copy(), target)
    print(f"nums={nums}, target={target} -> {result}")
```

关键考点 (Key Points)

1. **哈希表应用**: 理解如何用哈希表优化查找
2. **一次遍历**: 边遍历边查找，避免重复工作
3. **边界条件**: 处理负数、重复元素等情况

Problem 2: Valid Anagram (Easy)

Problem Statement

```
 Given two strings s and t, return true if t is an anagram of s, and false
otherwise.

An Anagram is a word or phrase formed by rearranging the letters of a different
word or phrase, typically using all the original letters exactly once.

Example 1:
Input: s = "anagram", t = "nagaram"
Output: true

Example 2:
Input: s = "rat", t = "car"
Output: false

Constraints:
- 1 <= s.length, t.length <= 5 * 10^4
- s and t consist of lowercase English letters only.
```

解题思路 (Solution Analysis)

这是一道经典的**字符串**和**哈希表**问题，考查对字符频次统计的理解。

排序解法： - 将两个字符串排序后比较是否相等 - 时间复杂度：O(n log n) - 空间复杂度：O(1) 或 O(n)（取决于排序算法）

字符计数解法： - 统计两个字符串中每个字符的出现次数 - 比较两个字符计数是否相同 - 时间复杂度：O(n) - 空间复杂度：O(1)（只有26个小写字母）

代码实现 (Code Implementation)

```python
from collections import Counter

def isAnagram(s, t):
    """
    Check if two strings are anagrams using character counting.

    Args:
        s: str - First string
        t: str - Second string

    Returns:
        bool - True if anagrams, False otherwise
    """
    if len(s) != len(t):
        return False

    # Count characters in both strings
    return Counter(s) == Counter(t)

def isAnagramArray(s, t):
    """
    Implementation using array for character counting.
    More efficient for lowercase letters only.
    """
    if len(s) != len(t):
        return False

    # Array to count characters (26 lowercase letters)
    char_count = [0] * 26

    for i in range(len(s)):
        char_count[ord(s[i]) - ord('a')] += 1
        char_count[ord(t[i]) - ord('a')] -= 1

    # All counts should be zero if anagrams
    return all(count == 0 for count in char_count)

def isAnagramSorting(s, t):
    """
    Simple sorting approach.
    """
    return sorted(s) == sorted(t)

# Test cases
test_cases = [
    ("anagram", "nagaram"),     # True
    ("rat", "car"),             # False
    ("listen", "silent"),       # True
    ("hello", "bello"),         # False
    ("a", "ab")                 # False
]

for s, t in test_cases:
    result1 = isAnagram(s, t)
    result2 = isAnagramArray(s, t)
    result3 = isAnagramSorting(s, t)
```

```
    print(f's="{s}", t="{t}" -> Counter: {result1}, Array: {result2}, Sorting:
{result3}')
```

关键考点 (Key Points)

1. **字符统计**: 多种方法统计字符频次
2. **时间复杂度**: 理解不同方法的复杂度差异
3. **空间优化**: 针对特定字符集的优化

Problem 3: Contains Duplicate (Easy)

Problem Statement

```
Given an integer array nums, return true if any value appears at least twice
in
the array, and return false if every element is distinct.

Example 1:
Input: nums = [1,2,3,1]
Output: true

Example 2:
Input: nums = [1,2,3,4]
Output: false

Example 3:
Input: nums = [1,1,1,3,3,2,2,2]
Output: true

Constraints:
- 1 <= nums.length <= 10^5
- -10^9 <= nums[i] <= 10^9
```

解题思路 (Solution Analysis)

这是一道考查**集合(Set)**数据结构的基础题目。

集合解法： - 遍历数组，将元素加入集合 - 如果元素已存在于集合中，说明有重复 - 时间复杂度：O(n) - 空间复杂度：O(n)

排序解法： - 先排序，然后检查相邻元素是否相同 - 时间复杂度：O(n log n) - 空间复杂度：O(1)

代码实现 (Code Implementation)

```python
def containsDuplicate(nums):
    """
    Check if array contains duplicates using set.

    Args:
        nums: List[int] - Array of integers

    Returns:
        bool - True if duplicates exist, False otherwise
    """
    seen = set()

    for num in nums:
        if num in seen:
            return True
        seen.add(num)

    return False

def containsDuplicateSet(nums):
    """
    One-liner using set length comparison.
    """
    return len(nums) != len(set(nums))

def containsDuplicateSorting(nums):
    """
    Sorting approach with O(1) extra space.
    """
    nums.sort()

    for i in range(1, len(nums)):
        if nums[i] == nums[i-1]:
            return True

    return False

# Test cases
test_cases = [
    [1, 2, 3, 1],           # True
    [1, 2, 3, 4],           # False
    [1, 1, 1, 3, 3, 2, 2, 2],  # True
    [1],                    # False
    []                      # False
]

for nums in test_cases:
    result1 = containsDuplicate(nums.copy())
    result2 = containsDuplicateSet(nums.copy())
    result3 = containsDuplicateSorting(nums.copy())
    print(f"nums={nums} -> Set: {result1}, SetLen: {result2}, Sorting:
{result3}")
```

关键考点 (Key Points)

1. **集合的应用**: 利用集合的唯一性检测重复
2. **一行代码解法**: 理解Python集合的特性
3. **空间时间权衡**: 不同方法的复杂度分析

Problem 4: Maximum Subarray (Easy)

Problem Statement

```
 Given an integer array nums, find the contiguous subarray (containing at least
one number) which has the largest sum and return its sum.

A subarray is a contiguous part of an array.

Example 1:
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: [4,-1,2,1] has the largest sum = 6.

Example 2:
Input: nums = [1]
Output: 1

Example 3:
Input: nums = [5,4,-1,7,8]
Output: 23

Constraints:
- 1 <= nums.length <= 10^5
- -10^4 <= nums[i] <= 10^4
```

解题思路 (Solution Analysis)

这是著名的**Kadane算法**，是动态规划的经典应用。

Kadane算法思想： - 维护当前子数组的最大和 - 如果当前和为负数，重新开始计算 - 记录过程中的全局最大值

算法步骤： 1. 初始化当前最大和为第一个元素 2. 遍历数组，更新当前最大和 3. 如果当前和小于当前元素，重新开始 4. 更新全局最大和

代码实现 (Code Implementation)

```python
def maxSubArray(nums):
    """
    Find maximum sum of contiguous subarray using Kadane's algorithm.

    Args:
        nums: List[int] - Array of integers

    Returns:
        int - Maximum subarray sum
    """
    max_sum = current_sum = nums[0]

    for num in nums[1:]:
        # Either extend existing subarray or start new one
        current_sum = max(num, current_sum + num)
        # Update global maximum
        max_sum = max(max_sum, current_sum)

    return max_sum

def maxSubArrayDP(nums):
    """
    Dynamic programming approach with explicit DP array.
    """
    n = len(nums)
    dp = [0] * n
    dp[0] = nums[0]
    max_sum = nums[0]

    for i in range(1, n):
        dp[i] = max(nums[i], dp[i-1] + nums[i])
        max_sum = max(max_sum, dp[i])

    return max_sum

def maxSubArrayWithIndices(nums):
    """
    Return both maximum sum and the subarray indices.
    """
    max_sum = current_sum = nums[0]
    start = end = 0
    temp_start = 0

    for i in range(1, len(nums)):
        if current_sum < 0:
            current_sum = nums[i]
            temp_start = i
        else:
            current_sum += nums[i]

        if current_sum > max_sum:
            max_sum = current_sum
            start = temp_start
            end = i

    return max_sum, nums[start:end+1]

# Test cases
```

```
test_cases = [
    [-2, 1, -3, 4, -1, 2, 1, -5, 4],  # 6
    [1],                               # 1
    [5, 4, -1, 7, 8],                  # 23
    [-1],                              # -1
    [-2, -1]                           # -1
]

for nums in test_cases:
    result1 = maxSubArray(nums)
    result2 = maxSubArrayDP(nums)
    max_sum, subarray = maxSubArrayWithIndices(nums)
    print(f"nums={nums}")
    print(f"Kadane: {result1}, DP: {result2}, With indices: {max_sum}
{subarray}\n")
```

关键考点 (Key Points)

1. **Kadane算法**: 理解其核心思想和实现
2. **动态规划**: 状态定义和转移方程
3. **负数处理**: 正确处理全为负数的情况

Problem 5: Merge Two Sorted Lists (Easy)

Problem Statement

```
You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists in a one sorted list. The list should be made by splicing
together the nodes of the first two lists.

Return the head of the merged linked list.

Example 1:
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]

Example 2:
Input: list1 = [], list2 = []
Output: []

Example 3:
Input: list1 = [], list2 = [0]
Output: [0]

Constraints:
- The number of nodes in both lists is in the range [0, 50].
- -100 <= Node.val <= 100
- Both list1 and list2 are sorted in non-decreasing order.
```

解题思路 (Solution Analysis)

这是一道经典的**链表**操作题目，考查对链表基本操作的掌握。

迭代解法： - 使用双指针分别指向两个链表 - 比较当前节点值，选择较小的加入结果链表 - 移动对应指针，重复直到一个链表为空 - 将剩余链表直接连接到结果链表

递归解法： - 比较两个链表头节点 - 选择较小的作为当前节点 - 递归处理剩余部分

代码实现 (Code Implementation)

```python
# Definition for singly-linked list
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

    def __repr__(self):
        result = []
        current = self
        while current:
            result.append(str(current.val))
            current = current.next
        return " -> ".join(result)

def mergeTwoLists(list1, list2):
    """
    Merge two sorted linked lists iteratively.

    Args:
        list1: ListNode - Head of first sorted list
        list2: ListNode - Head of second sorted list

    Returns:
        ListNode - Head of merged sorted list
    """
    # Create dummy head for easier manipulation
    dummy = ListNode(0)
    current = dummy

    # Merge while both lists have nodes
    while list1 and list2:
        if list1.val <= list2.val:
            current.next = list1
            list1 = list1.next
        else:
            current.next = list2
            list2 = list2.next
        current = current.next

    # Append remaining nodes
    current.next = list1 or list2

    return dummy.next

def mergeTwoListsRecursive(list1, list2):
    """
    Recursive solution for merging two sorted lists.
    """
    if not list1:
        return list2
    if not list2:
        return list1

    if list1.val <= list2.val:
        list1.next = mergeTwoListsRecursive(list1.next, list2)
        return list1
    else:
        list2.next = mergeTwoListsRecursive(list1, list2.next)
```

```
        return list2

# Helper function to create linked list from array
def create_linked_list(arr):
    """Create linked list from array."""
    if not arr:
        return None

    head = ListNode(arr[0])
    current = head
    for val in arr[1:]:
        current.next = ListNode(val)
        current = current.next

    return head

# Test cases
test_cases = [
    ([1, 2, 4], [1, 3, 4]),     # [1,1,2,3,4,4]
    ([], []),                   # []
    ([], [0]),                  # [0]
    ([1, 2, 3], [4, 5, 6])      # [1,2,3,4,5,6]
]

for arr1, arr2 in test_cases:
    list1 = create_linked_list(arr1)
    list2 = create_linked_list(arr2)

    # Test iterative solution
    result_iter = mergeTwoLists(list1, list2)

    # Recreate lists for recursive test
    list1 = create_linked_list(arr1)
    list2 = create_linked_list(arr2)
    result_rec = mergeTwoListsRecursive(list1, list2)

    print(f"Input: {arr1} + {arr2}")
    print(f"Iterative: {result_iter}")
    print(f"Recursive: {result_rec}\n")
```

关键考点 (Key Points)

1. **链表操作**: 理解链表的基本操作和指针移动
2. **边界条件**: 处理空链表的情况
3. **递归思维**: 理解递归解法的思路

Problem 6: Best Time to Buy and Sell Stock (Easy)

Problem Statement

```
You are given an array prices where prices[i] is the price of a given stock on
day i.

You want to maximize your profit by choosing a single day to buy one stock and
choosing a different day in the future to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot
achieve any profit, return 0.

Example 1:
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit =
6-1 = 5.

Example 2:
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.

Constraints:
- 1 <= prices.length <= 10^5
- 0 <= prices[i] <= 10^4
```

解题思路 (Solution Analysis)

这是一道经典的**贪心算法**题目，考查对最优策略的理解。

核心思想： - 维护到目前为止的最低买入价格 - 计算每天卖出的利润，更新最大利润

算法步骤： 1. 初始化最低价格为第一天价格 2. 遍历价格数组 3. 更新最低价格 4. 计算当天卖出的利润，更新最大利润

代码实现 (Code Implementation)

```
def maxProfit(prices):
    """
    Find maximum profit from single buy-sell transaction.

    Args:
        prices: List[int] - Array of stock prices

    Returns:
        int - Maximum profit
    """
    if not prices or len(prices) < 2:
        return 0

    min_price = prices[0]
    max_profit = 0

    for price in prices[1:]:
        # Update minimum price seen so far
        min_price = min(min_price, price)
        # Calculate profit if selling today
        profit = price - min_price
        # Update maximum profit
        max_profit = max(max_profit, profit)

    return max_profit

def maxProfitDP(prices):
    """
    Dynamic programming approach.
    """
    if not prices:
        return 0

    # dp[i][0] = max profit on day i when not holding stock
    # dp[i][1] = max profit on day i when holding stock
    hold = -prices[0]  # Bought stock on day 0
    sold = 0           # No stock on day 0

    for price in prices[1:]:
        # Either keep not holding, or sell today
        new_sold = max(sold, hold + price)
        # Either keep holding, or buy today
        new_hold = max(hold, -price)  # Can only buy once

        sold, hold = new_sold, new_hold

    return sold

# Test cases
test_cases = [
    [7, 1, 5, 3, 6, 4],     # 5
    [7, 6, 4, 3, 1],        # 0
    [1, 2, 3, 4, 5],        # 4
    [2, 4, 1],              # 2
    [1]                     # 0
]

for prices in test_cases:
```

```
    result1 = maxProfit(prices)
    result2 = maxProfitDP(prices)
    print(f"prices={prices} -> Greedy: {result1}, DP: {result2}")
```

关键考点 (Key Points)

1. **贪心策略**: 在最低点买入，在最高点卖出
2. **一次遍历**: 只需要遍历一次数组
3. **状态维护**: 正确维护最低价格和最大利润

Problem 7: Valid Palindrome (Easy)

Problem Statement

```
 A phrase is a palindrome if, after converting all uppercase letters into
lowercase
letters and removing all non-alphanumeric characters, it reads the same forward
and backward. Alphanumeric characters include letters and numbers.

Given a string s, return true if it is a palindrome, or false otherwise.

Example 1:
Input: s = "A man, a plan, a canal: Panama"
Output: true
Explanation: "amanaplanacanalpanama" is a palindrome.

Example 2:
Input: s = "race a car"
Output: false
Explanation: "raceacar" is not a palindrome.

Example 3:
Input: s = " "
Output: true
Explanation: s is an empty string "" after removing non-alphanumeric
characters.

Constraints:
- 1 <= s.length <= 2 * 10^5
- s consists only of printable ASCII characters.
```

解题思路 (Solution Analysis)

这是一道经典的**双指针**和**字符串处理**题目。

双指针解法： - 使用左右两个指针从字符串两端向中间移动 - 跳过非字母数字字符 - 比较对应字符是否相同（忽略大小写）

预处理解法： - 先过滤出所有字母数字字符并转为小写 - 然后判断处理后的字符串是否是回文

代码实现 (Code Implementation)

```python
def isPalindrome(s):
    """
    Check if string is palindrome using two pointers.

    Args:
        s: str - Input string

    Returns:
        bool - True if palindrome, False otherwise
    """
    left, right = 0, len(s) - 1

    while left < right:
        # Skip non-alphanumeric characters from left
        while left < right and not s[left].isalnum():
            left += 1

        # Skip non-alphanumeric characters from right
        while left < right and not s[right].isalnum():
            right -= 1

        # Compare characters (case insensitive)
        if s[left].lower() != s[right].lower():
            return False

        left += 1
        right -= 1

    return True

def isPalindromePreprocess(s):
    """
    Preprocess string then check palindrome.
    """
    # Filter and convert to lowercase
    cleaned = ''.join(char.lower() for char in s if char.isalnum())

    # Check if cleaned string is palindrome
    return cleaned == cleaned[::-1]

def isPalindromeRecursive(s):
    """
    Recursive solution for palindrome check.
    """
    def is_palindrome_helper(left, right):
        if left >= right:
            return True

        # Skip non-alphanumeric from left
        if not s[left].isalnum():
            return is_palindrome_helper(left + 1, right)

        # Skip non-alphanumeric from right
        if not s[right].isalnum():
            return is_palindrome_helper(left, right - 1)

        # Compare current characters
        if s[left].lower() != s[right].lower():
```

```
            return False

        return is_palindrome_helper(left + 1, right - 1)

    return is_palindrome_helper(0, len(s) - 1)

# Test cases
test_cases = [
    "A man, a plan, a canal: Panama",   # True
    "race a car",                       # False
    " ",                                # True
    "Madam",                           # True
    "No 'x' in Nixon"                  # True
]

for s in test_cases:
    result1 = isPalindrome(s)
    result2 = isPalindromePreprocess(s)
    result3 = isPalindromeRecursive(s)
    print(f's="{s}"')
    print(f'Two pointers: {result1}, Preprocess: {result2}, Recursive:
{result3}\n')
```

关键考点 (Key Points)

1. **双指针技巧**: 从两端向中间移动的经典模式
2. **字符处理**: 正确处理大小写和非字母数字字符
3. **边界条件**: 空字符串和单字符的处理

Problem 8: Climbing Stairs (Easy)

Problem Statement

```
 You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you
climb to the top?

Example 1:
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps

Example 2:
Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step

Constraints:
- 1 <= n <= 45
```

解题思路 (Solution Analysis)

这是一道经典的**动态规划**入门题目，实际上就是斐波那契数列。

递推关系：- `f(n) = f(n-1) + f(n-2)` - 到达第n阶的方法数 = 到达第(n-1)阶的方法数 + 到达第(n-2)阶的方法数

基础情况：- `f(1) = 1`（只有一种方法：走1步）- `f(2) = 2`（两种方法：走1+1步或走2步）

代码实现 (Code Implementation)

```python
def climbStairs(n):
    """
    Calculate number of ways to climb stairs using DP.

    Args:
        n: int - Number of stairs

    Returns:
        int - Number of distinct ways
    """
    if n <= 2:
        return n

    # Use two variables instead of array for O(1) space
    prev2 = 1  # f(1)
    prev1 = 2  # f(2)

    for i in range(3, n + 1):
        current = prev1 + prev2
        prev2 = prev1
        prev1 = current

    return prev1

def climbStairsDP(n):
    """
    Traditional DP approach with array.
    """
    if n <= 2:
        return n

    dp = [0] * (n + 1)
    dp[1] = 1
    dp[2] = 2

    for i in range(3, n + 1):
        dp[i] = dp[i-1] + dp[i-2]

    return dp[n]

def climbStairsRecursive(n):
    """
    Recursive solution with memoization.
    """
    memo = {}

    def climb(n):
        if n in memo:
            return memo[n]

        if n <= 2:
            return n

        memo[n] = climb(n-1) + climb(n-2)
        return memo[n]

    return climb(n)
```

```python
def climbStairsMath(n):
    """
    Mathematical solution using Fibonacci formula.
    """
    import math

    sqrt5 = math.sqrt(5)
    phi = (1 + sqrt5) / 2
    psi = (1 - sqrt5) / 2

    return int((phi**(n+1) - psi**(n+1)) / sqrt5)

# Test cases
test_cases = [1, 2, 3, 4, 5, 10, 20]

for n in test_cases:
    result1 = climbStairs(n)
    result2 = climbStairsDP(n)
    result3 = climbStairsRecursive(n)
    result4 = climbStairsMath(n)
    print(f"n={n} -> Optimized: {result1}, DP: {result2}, Recursive: {result3},
Math: {result4}")
```

关键考点 (Key Points)

1. **动态规划**: 理解状态转移方程
2. **空间优化**: 从O(n)优化到O(1)
3. **斐波那契数列**: 识别问题的本质

Problem 9: Binary Search (Easy)

Problem Statement

```
 Given an array of integers nums which is sorted in ascending order, and an integer
target, write a function to search target in nums. If target exists, then return
its index. Otherwise, return -1.

You must write an algorithm with O(log n) runtime complexity.

Example 1:
Input: nums = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4

Example 2:
Input: nums = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1

Constraints:
- 1 <= nums.length <= 10^4
- -10^4 < nums[i], target < 10^4
- All the integers in nums are unique.
- nums is sorted in ascending order.
```

解题思路 (Solution Analysis)

这是**二分查找**的标准实现，是必须掌握的基础算法。

算法思想： - 每次比较中间元素与目标值 - 根据比较结果缩小搜索范围 - 重复直到找到目标或搜索范围为空

关键点： - 正确处理边界条件 - 避免整数溢出 - 循环不变量的维护

代码实现 (Code Implementation)

```python
def search(nums, target):
    """
    Binary search implementation.

    Args:
        nums: List[int] - Sorted array
        target: int - Target value to search

    Returns:
        int - Index of target or -1 if not found
    """
    left, right = 0, len(nums) - 1

    while left <= right:
        # Avoid overflow: mid = (left + right) // 2
        mid = left + (right - left) // 2

        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1

def searchRecursive(nums, target):
    """
    Recursive binary search implementation.
    """
    def binary_search(left, right):
        if left > right:
            return -1

        mid = left + (right - left) // 2

        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            return binary_search(mid + 1, right)
        else:
            return binary_search(left, mid - 1)

    return binary_search(0, len(nums) - 1)

def searchLeftBound(nums, target):
    """
    Find leftmost position where target can be inserted.
    """
    left, right = 0, len(nums)

    while left < right:
        mid = left + (right - left) // 2

        if nums[mid] < target:
            left = mid + 1
        else:
            right = mid
```

```
    return left if left < len(nums) and nums[left] == target else -1

def searchRightBound(nums, target):
    """
    Find rightmost position where target can be inserted.
    """
    left, right = 0, len(nums)

    while left < right:
        mid = left + (right - left) // 2

        if nums[mid] <= target:
            left = mid + 1
        else:
            right = mid

    return left - 1 if left > 0 and nums[left - 1] == target else -1

# Test cases
test_cases = [
    ([-1, 0, 3, 5, 9, 12], 9),     # 4
    ([-1, 0, 3, 5, 9, 12], 2),     # -1
    ([5], 5),                      # 0
    ([1, 3, 5, 7, 9], 1),          # 0
    ([1, 3, 5, 7, 9], 9)           # 4
]

for nums, target in test_cases:
    result1 = search(nums, target)
    result2 = searchRecursive(nums, target)
    result3 = searchLeftBound(nums, target)
    result4 = searchRightBound(nums, target)
    print(f"nums={nums}, target={target}")
    print(f"Standard: {result1}, Recursive: {result2}, Left: {result3}, Right:
{result4}\n")
```

关键考点 (Key Points)

1. **边界处理**: `left <= right` vs `left < right`
2. **中点计算**: 避免整数溢出的写法
3. **变体问题**: 左边界、右边界查找

Problem 10: Reverse Linked List (Easy)

Problem Statement

```
 Given the head of a singly linked list, reverse the list, and return the new
head.

Example 1:
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]

Example 2:
Input: head = [1,2]
Output: [2,1]

Example 3:
Input: head = []
Output: []

Constraints:
- The number of nodes in the list is the range [0, 5000].
- -5000 <= Node.val <= 5000
```

解题思路 (Solution Analysis)

这是一道经典的**链表操作**题目，考查对指针操作的理解。

迭代解法： - 使用三个指针：prev, current, next - 逐个反转链表中的指针方向

递归解法： - 递归到链表末尾 - 在回溯过程中反转指针

代码实现 (Code Implementation)

```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

    def to_list(self):
        """Convert linked list to Python list for easy display."""
        result = []
        current = self
        while current:
            result.append(current.val)
            current = current.next
        return result

def reverseList(head):
    """
    Reverse linked list iteratively.

    Args:
        head: ListNode - Head of the linked list

    Returns:
        ListNode - New head of reversed list
    """
    prev = None
    current = head

    while current:
        next_temp = current.next  # Store next node
        current.next = prev       # Reverse the link
        prev = current           # Move prev forward
        current = next_temp      # Move current forward

    return prev  # prev is the new head

def reverseListRecursive(head):
    """
    Reverse linked list recursively.
    """
    # Base case
    if not head or not head.next:
        return head

    # Recursively reverse the rest
    new_head = reverseListRecursive(head.next)

    # Reverse current connection
    head.next.next = head
    head.next = None

    return new_head

def reverseListStack(head):
    """
    Reverse using stack (for educational purposes).
    """
    if not head:
        return None
```

```
    stack = []
    current = head

    # Push all nodes to stack
    while current:
        stack.append(current)
        current = current.next

    # Pop and reconnect
    new_head = stack.pop()
    current = new_head

    while stack:
        current.next = stack.pop()
        current = current.next

    current.next = None
    return new_head

# Helper function to create linked list
def create_list(arr):
    """Create linked list from array."""
    if not arr:
        return None

    head = ListNode(arr[0])
    current = head
    for val in arr[1:]:
        current.next = ListNode(val)
        current = current.next

    return head

# Test cases
test_cases = [
    [1, 2, 3, 4, 5],    # [5,4,3,2,1]
    [1, 2],             # [2,1]
    [],                 # []
    [1]                 # [1]
]

for arr in test_cases:
    # Test iterative solution
    head1 = create_list(arr)
    reversed1 = reverseList(head1)
    result1 = reversed1.to_list() if reversed1 else []

    # Test recursive solution
    head2 = create_list(arr)
    reversed2 = reverseListRecursive(head2)
    result2 = reversed2.to_list() if reversed2 else []

    # Test stack solution
    head3 = create_list(arr)
    reversed3 = reverseListStack(head3)
    result3 = reversed3.to_list() if reversed3 else []

    print(f"Original: {arr}")
    print(f"Iterative: {result1}, Recursive: {result2}, Stack: {result3}\n")
```

关键考点 (Key Points)

1. **指针操作**: 正确处理三个指针的移动
2. **递归思维**: 理解递归的执行过程
3. **边界条件**: 空链表和单节点链表的处理

Problem 11: Move Zeroes (Easy)

Problem Statement

```
 Given an integer array nums, move all 0's to the end of it while maintaining
the
relative order of the non-zero elements.

Note that you must do this in-place without making a copy of the array.

Example 1:
Input: nums = [0,1,0,3,12]
Output: [1,3,12,0,0]

Example 2:
Input: nums = [0]
Output: [0]

Constraints:
- 1 <= nums.length <= 10^4
- -2^31 <= nums[i] <= 2^31 - 1
```

解题思路 (Solution Analysis)

这是一道经典的**双指针**题目，考查数组的原地操作。

双指针解法： - 使用快慢指针，慢指针指向下一个非零元素应该放置的位置 - 快指针遍历数组，遇到非零元素就与慢指针位置交换 - 最后将剩余位置填充为0

时间复杂度: O(n) **空间复杂度**: O(1)

代码实现 (Code Implementation)

```python
def moveZeroes(nums):
    """
    Move all zeros to end while maintaining relative order.

    Args:
        nums: List[int] - Array to modify in-place
    """
    # Two pointers approach
    slow = 0  # Points to position for next non-zero element

    # Move all non-zero elements to front
    for fast in range(len(nums)):
        if nums[fast] != 0:
            nums[slow] = nums[fast]
            slow += 1

    # Fill remaining positions with zeros
    while slow < len(nums):
        nums[slow] = 0
        slow += 1

def moveZeroesSwap(nums):
    """
    Alternative approach using swapping.
    """
    slow = 0

    for fast in range(len(nums)):
        if nums[fast] != 0:
            nums[slow], nums[fast] = nums[fast], nums[slow]
            slow += 1

def moveZeroesOptimal(nums):
    """
    Most optimal approach - only swap when necessary.
    """
    slow = 0

    for fast in range(len(nums)):
        if nums[fast] != 0:
            if slow != fast:  # Only swap if positions are different
                nums[slow], nums[fast] = nums[fast], nums[slow]
            slow += 1

# Test cases
test_cases = [
    [0, 1, 0, 3, 12],       # [1,3,12,0,0]
    [0],                    # [0]
    [1, 2, 3],              # [1,2,3]
    [0, 0, 1],              # [1,0,0]
    [1, 0, 2, 0, 3, 0, 4]   # [1,2,3,4,0,0,0]
]

for nums in test_cases:
    original = nums.copy()
    moveZeroes(nums)
    print(f"Original: {original} -> Result: {nums}")
```

关键考点 (Key Points)

1. **双指针技巧**: 快慢指针的经典应用
2. **原地操作**: 不使用额外空间修改数组
3. **相对顺序**: 保持非零元素的原始顺序

Problem 12: Intersection of Two Arrays II (Easy)

Problem Statement

```
 Given two integer arrays nums1 and nums2, return an array of their
intersection.
Each element in the result must appear as many times as it shows in both arrays
and you may return the result in any order.

Example 1:
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2,2]

Example 2:
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [4,9]
Explanation: [9,4] is also accepted.

Constraints:
- 1 <= nums1.length, nums2.length <= 1000
- 0 <= nums1[i], nums2[i] <= 1000
```

解题思路 (Solution Analysis)

这是一道考查**哈希表**和**双指针**的题目。

哈希表解法： - 统计一个数组中每个元素的频次 - 遍历另一个数组，如果元素在哈希表中且频次大于0，加入结果并减少频次

排序+双指针解法： - 将两个数组排序 - 使用双指针比较元素，相等时加入结果

代码实现 (Code Implementation)

```python
from collections import Counter

def intersect(nums1, nums2):
    """
    Find intersection of two arrays with duplicates.

    Args:
        nums1: List[int] - First array
        nums2: List[int] - Second array

    Returns:
        List[int] - Intersection with duplicates
    """
    # Count elements in nums1
    count = Counter(nums1)
    result = []

    # Check each element in nums2
    for num in nums2:
        if count[num] > 0:
            result.append(num)
            count[num] -= 1

    return result

def intersectTwoPointers(nums1, nums2):
    """
    Two pointers approach after sorting.
    """
    nums1.sort()
    nums2.sort()

    i = j = 0
    result = []

    while i < len(nums1) and j < len(nums2):
        if nums1[i] < nums2[j]:
            i += 1
        elif nums1[i] > nums2[j]:
            j += 1
        else:
            result.append(nums1[i])
            i += 1
            j += 1

    return result

def intersectOptimized(nums1, nums2):
    """
    Optimized to use smaller array for counting.
    """
    # Ensure nums1 is the smaller array
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1

    count = Counter(nums1)
    result = []
```

```
    for num in nums2:
        if count[num] > 0:
            result.append(num)
            count[num] -= 1

    return result

# Test cases
test_cases = [
    ([1, 2, 2, 1], [2, 2]),           # [2,2]
    ([4, 9, 5], [9, 4, 9, 8, 4]),     # [4,9] or [9,4]
    ([1, 2, 3], [4, 5, 6]),           # []
    ([1, 1], [1, 1, 1])               # [1,1]
]

for nums1, nums2 in test_cases:
    result1 = intersect(nums1.copy(), nums2.copy())
    result2 = intersectTwoPointers(nums1.copy(), nums2.copy())
    result3 = intersectOptimized(nums1.copy(), nums2.copy())
    print(f"nums1={nums1}, nums2={nums2}")
    print(f"Counter: {result1}, Two pointers: {result2}, Optimized:
{result3}\n")
```

关键考点 (Key Points)

1. **频次统计**: 使用Counter或字典统计元素频次
2. **双指针**: 排序后的双指针遍历
3. **空间优化**: 选择较小数组进行计数

Problem 13: Plus One (Easy)

Problem Statement

```
 You are given a large integer represented as an integer array digits, where each
digits[i] is the ith digit of the integer. The digits are ordered from most
significant to least significant in left-to-right order. The large integer does
not contain any leading zeros.

Increment the large integer by one and return the resulting array of digits.

Example 1:
Input: digits = [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123. Incrementing by one gives
123 + 1 = 124.

Example 2:
Input: digits = [4,3,2,1]
Output: [4,3,2,2]
Explanation: The array represents the integer 4321. Incrementing by one gives
4321 + 1 = 4322.

Example 3:
Input: digits = [9]
Output: [1,0]
Explanation: The array represents the integer 9. Incrementing by one gives 9 +
1 = 10.

Constraints:
- 1 <= digits.length <= 100
- 0 <= digits[i] <= 9
- digits does not contain any leading zeros except for the number 0 itself.
```

解题思路 (Solution Analysis)

这是一道考查**数组操作**和**进位处理**的题目。

核心思想： - 从最后一位开始处理进位 - 如果当前位小于9，直接加1返回 - 如果当前位是9，变为0并继续处理进位 - 如果所有位都是9，需要在最前面添加1

代码实现 (Code Implementation)

```python
def plusOne(digits):
    """
    Add one to number represented as digit array.

    Args:
        digits: List[int] - Array representing a number

    Returns:
        List[int] - Result after adding one
    """
    # Process from right to left
    for i in range(len(digits) - 1, -1, -1):
        if digits[i] < 9:
            digits[i] += 1
            return digits
        digits[i] = 0

    # If we reach here, all digits were 9
    return [1] + digits

def plusOneRecursive(digits):
    """
    Recursive approach for plus one.
    """
    def add_carry(index):
        if index < 0:
            return [1]  # All digits were 9

        if digits[index] < 9:
            digits[index] += 1
            return digits
        else:
            digits[index] = 0
            return add_carry(index - 1)

    return add_carry(len(digits) - 1)

def plusOneString(digits):
    """
    Convert to string, add one, convert back (for comparison).
    Note: This approach may not work for very large numbers.
    """
    # Convert to integer
    num = int(''.join(map(str, digits)))
    # Add one
    num += 1
    # Convert back to digit array
    return [int(d) for d in str(num)]

# Test cases
test_cases = [
    [1, 2, 3],          # [1,2,4]
    [4, 3, 2, 1],       # [4,3,2,2]
    [9],                # [1,0]
    [9, 9, 9],          # [1,0,0,0]
    [1, 9, 9],          # [2,0,0]
    [0]                 # [1]
]
```

```
for digits in test_cases:
    original = digits.copy()
    result1 = plusOne(digits.copy())
    result2 = plusOneRecursive(digits.copy())
    result3 = plusOneString(digits.copy())
    print(f"Original: {original}")
    print(f"Iterative: {result1}, Recursive: {result2}, String: {result3}\n")
```

关键考点 (Key Points)

1. **进位处理**: 正确处理数字加法的进位
2. **边界条件**: 全为9的特殊情况
3. **数组操作**: 从右到左的遍历和修改

Problem 14: Single Number (Easy)

Problem Statement

```
 Given a non-empty array of integers nums, every element appears twice except
for
one. Find that single one.

You must implement a solution with a linear runtime complexity and use only
constant extra space.

Example 1:
Input: nums = [2,2,1]
Output: 1

Example 2:
Input: nums = [4,1,2,1,2]
Output: 4

Example 3:
Input: nums = [1]
Output: 1

Constraints:
- 1 <= nums.length <= 3 * 10^4
- -3 * 10^4 <= nums[i] <= 3 * 10^4
- Each element in the array appears twice except for one element which appears
only once.
```

解题思路 (Solution Analysis)

这是一道经典的**位运算**题目，考查对XOR操作的理解。

XOR性质： - `a ^ a = 0`（任何数与自己异或为0） - `a ^ 0 = a`（任何数与0异或为自己） - XOR满足交换律和结合律

算法思想： - 将所有数字进行XOR操作 - 相同的数字会相互抵消变为0 - 最后剩下的就是只出现一次的数字

代码实现 (Code Implementation)

```python
def singleNumber(nums):
    """
    Find single number using XOR operation.

    Args:
        nums: List[int] - Array with one unique element

    Returns:
        int - The single number
    """
    result = 0
    for num in nums:
        result ^= num
    return result

def singleNumberFunctional(nums):
    """
    Functional programming approach using reduce.
    """
    from functools import reduce
    import operator
    return reduce(operator.xor, nums, 0)

def singleNumberSet(nums):
    """
    Using set for comparison (uses O(n) space).
    """
    return 2 * sum(set(nums)) - sum(nums)

def singleNumberMath(nums):
    """
    Mathematical approach using sum.
    """
    unique_nums = set(nums)
    return 2 * sum(unique_nums) - sum(nums)

# Demonstration of XOR properties
def demonstrate_xor():
    """
    Demonstrate XOR properties for educational purposes.
    """
    print("XOR Properties Demonstration:")
    print(f"5 ^ 5 = {5 ^ 5}")  # 0
    print(f"7 ^ 0 = {7 ^ 0}")  # 7
    print(f"3 ^ 5 ^ 3 = {3 ^ 5 ^ 3}")  # 5
    print(f"1 ^ 2 ^ 3 ^ 2 ^ 1 = {1 ^ 2 ^ 3 ^ 2 ^ 1}")  # 3
    print()

# Test cases
test_cases = [
    [2, 2, 1],              # 1
    [4, 1, 2, 1, 2],        # 4
    [1],                    # 1
    [7, 3, 7],              # 3
    [1, 2, 3, 4, 1, 2, 3]   # 4
]

demonstrate_xor()
```

```
for nums in test_cases:
    result1 = singleNumber(nums)
    result2 = singleNumberFunctional(nums)
    result3 = singleNumberSet(nums)
    result4 = singleNumberMath(nums)
    print(f"nums={nums}")
    print(f"XOR: {result1}, Functional: {result2}, Set: {result3}, Math:
{result4}\n")
```

关键考点 (Key Points)

1. **位运算**: 理解XOR操作的性质和应用
2. **数学性质**: 利用数学关系解决问题
3. **空间复杂度**: O(1)空间的重要性

Problem 15: Happy Number (Easy)

Problem Statement

```
Write an algorithm to determine if a number n is happy.

A happy number is a number defined by the following process:
- Starting with any positive integer, replace the number by the sum of the
squares of its digits.
- Repeat the process until the number equals 1 (where it will stay), or it
loops endlessly in a cycle which does not include 1.
- Those numbers for which this process ends in 1 are happy.

Return true if n is a happy number, and false if it is not.

Example 1:
Input: n = 19
Output: true
Explanation:
1² + 9² = 82
8² + 2² = 68
6² + 8² = 100
1² + 0² + 0² = 1

Example 2:
Input: n = 2
Output: false

Constraints:
- 1 <= n <= 2^31 - 1
```

解题思路 (Solution Analysis)

这是一道考查**循环检测**的题目，可以使用**哈希集合**或**快慢指针**来解决。

哈希集合解法： - 使用集合记录已经出现过的数字 - 如果出现重复，说明进入循环，返回false - 如果到达1，返回true

快慢指针解法： - 类似于检测链表中的环 - 快指针每次计算两次，慢指针每次计算一次 - 如果有环，快慢指针会相遇

代码实现 (Code Implementation)

```python
def isHappy(n):
    """
    Check if number is happy using set to detect cycle.

    Args:
        n: int - Number to check

    Returns:
        bool - True if happy, False otherwise
    """
    def get_sum_of_squares(num):
        """Calculate sum of squares of digits."""
        total = 0
        while num > 0:
            digit = num % 10
            total += digit * digit
            num //= 10
        return total

    seen = set()

    while n != 1 and n not in seen:
        seen.add(n)
        n = get_sum_of_squares(n)

    return n == 1

def isHappyTwoPointers(n):
    """
    Check if number is happy using two pointers (Floyd's cycle detection).
    """
    def get_sum_of_squares(num):
        total = 0
        while num > 0:
            digit = num % 10
            total += digit * digit
            num //= 10
        return total

    slow = fast = n

    while True:
        slow = get_sum_of_squares(slow)
        fast = get_sum_of_squares(get_sum_of_squares(fast))

        if fast == 1:
            return True
        if slow == fast:
            return False

def isHappyRecursive(n, seen=None):
    """
    Recursive approach with memoization.
    """
    if seen is None:
        seen = set()

    if n == 1:
```

```
        return True
    if n in seen:
        return False

    seen.add(n)

    # Calculate sum of squares
    total = 0
    while n > 0:
        digit = n % 10
        total += digit * digit
        n //= 10

    return isHappyRecursive(total, seen)

def get_sum_of_squares_string(n):
    """
    Alternative implementation using string conversion.
    """
    return sum(int(digit) ** 2 for digit in str(n))

# Test cases with step-by-step demonstration
def demonstrate_happy_number(n):
    """
    Demonstrate the happy number calculation process.
    """
    print(f"Checking if {n} is happy:")
    seen = set()
    original_n = n

    while n != 1 and n not in seen:
        seen.add(n)
        digits = [int(d) for d in str(n)]
        squares = [d**2 for d in digits]
        new_n = sum(squares)
        print(f"{n} -> {' + '.join(f'{d}²' for d in digits)} = {' +
' .join(map(str, squares))} = {new_n}")
        n = new_n

    if n == 1:
        print(f"✓ {original_n} is happy!")
        return True
    else:
        print(f"✗ {original_n} enters cycle at {n}")
        return False

# Test cases
test_cases = [19, 2, 7, 10, 1, 23]

for n in test_cases:
    result1 = isHappy(n)
    result2 = isHappyTwoPointers(n)
    result3 = isHappyRecursive(n)
    print(f"n={n} -> Set: {result1}, Two pointers: {result2}, Recursive:
{result3}")

print("\nDetailed demonstration:")
demonstrate_happy_number(19)
print()
demonstrate_happy_number(2)
```

关键考点 (Key Points)

1. **循环检测**: 使用集合或快慢指针检测循环
2. **数字处理**: 计算各位数字的平方和
3. **算法优化**: 不同方法的空间复杂度对比

Problem 16: Count Primes (Medium)

Problem Statement

```
 Given an integer n, return the number of prime numbers that are less than n.

Example 1:
Input: n = 10
Output: 4
Explanation: There are 4 prime numbers less than 10, they are 2, 3, 5, 7.

Example 2:
Input: n = 0
Output: 0

Example 3:
Input: n = 1
Output: 0

Constraints:
- 0 <= n <= 5 * 10^6
```

解题思路 (Solution Analysis)

这是一道经典的**埃拉托斯特尼筛法(Sieve of Eratosthenes)**题目。

埃拉托斯特尼筛法： 1. 创建一个布尔数组，初始化所有数为质数 2. 从2开始，将每个质数的倍数标记为合数 3. 重复直到处理完所有数字 4. 统计剩余的质数个数

优化： - 只需要检查到√n - 跳过偶数（除了2） - 使用位操作优化空间

代码实现 (Code Implementation)

```python
def countPrimes(n):
    """
    Count prime numbers less than n using Sieve of Eratosthenes.

    Args:
        n: int - Upper bound (exclusive)

    Returns:
        int - Number of primes less than n
    """
    if n <= 2:
        return 0

    # Initialize boolean array
    is_prime = [True] * n
    is_prime[0] = is_prime[1] = False  # 0 and 1 are not prime

    # Sieve of Eratosthenes
    for i in range(2, int(n**0.5) + 1):
        if is_prime[i]:
            # Mark all multiples of i as composite
            for j in range(i*i, n, i):
                is_prime[j] = False

    # Count primes
    return sum(is_prime)

def countPrimesOptimized(n):
    """
    Optimized version that skips even numbers.
    """
    if n <= 2:
        return 0
    if n <= 3:
        return 1  # Only 2 is prime

    # Only track odd numbers (except 2)
    # Index i represents number 2*i+3
    size = (n - 3) // 2 + 1
    is_prime = [True] * size

    # Sieve for odd numbers only
    for i in range(int((n**0.5 - 3) // 2) + 1):
        if is_prime[i]:
            # The number represented by index i is 2*i+3
            num = 2 * i + 3
            # Mark multiples starting from num^2
            start = (num * num - 3) // 2
            for j in range(start, size, num):
                is_prime[j] = False

    # Count primes: 2 + odd primes
    return 1 + sum(is_prime)

def countPrimesNaive(n):
    """
    Naive approach for comparison (inefficient for large n).
    """
```

```
    def is_prime(num):
        if num < 2:
            return False
        for i in range(2, int(num**0.5) + 1):
            if num % i == 0:
                return False
        return True

    return sum(1 for i in range(2, n) if is_prime(i))

def getPrimes(n):
    """
    Return list of all primes less than n.
    """
    if n <= 2:
        return []

    is_prime = [True] * n
    is_prime[0] = is_prime[1] = False

    for i in range(2, int(n**0.5) + 1):
        if is_prime[i]:
            for j in range(i*i, n, i):
                is_prime[j] = False

    return [i for i in range(n) if is_prime[i]]

# Performance comparison
import time

def benchmark_methods(n):
    """
    Compare performance of different methods.
    """
    methods = [
        ("Sieve", countPrimes),
        ("Optimized", countPrimesOptimized),
    ]

    if n <= 1000:  # Only test naive method for small n
        methods.append(("Naive", countPrimesNaive))

    results = {}
    for name, method in methods:
        start_time = time.time()
        result = method(n)
        end_time = time.time()
        results[name] = (result, end_time - start_time)
        print(f"{name}: {result} primes, {end_time - start_time:.6f} seconds")

    return results

# Test cases
test_cases = [0, 1, 2, 3, 10, 100, 1000]

for n in test_cases:
    result1 = countPrimes(n)
    result2 = countPrimesOptimized(n)
    primes = getPrimes(n) if n <= 30 else []
    print(f"n={n} -> Sieve: {result1}, Optimized: {result2}")
    if primes:
        print(f"Primes: {primes}")
```

```
    print()

# Performance benchmark
print("Performance Benchmark (n=10000):")
benchmark_methods(10000)
```

关键考点 (Key Points)

1. **埃拉托斯特尼筛法**: 理解筛法的原理和实现
2. **算法优化**: 只检查到√n，跳过偶数等优化
3. **时间复杂度**: O(n log log n) vs O(n√n)

Problem 17: Power of Three (Easy)

Problem Statement

```
 Given an integer n, return true if it is a power of three. Otherwise, return
false.

An integer n is a power of three, if there exists an integer x such that n ==
3^x.

Example 1:
Input: n = 27
Output: true
Explanation: 27 = 3^3

Example 2:
Input: n = 0
Output: false
Explanation: There is no x where 3^x = 0.

Example 3:
Input: n = -1
Output: false
Explanation: There is no x where 3^x = (-1).

Constraints:
- -2^31 <= n <= 2^31 - 1
```

解题思路 (Solution Analysis)

这是一道考查**数学**和**循环**的题目，有多种解法。

循环除法： - 不断除以3，检查是否能整除 - 最后结果应该为1

递归解法： - 递归检查n/3是否为3的幂

数学解法： - 在给定范围内，最大的3的幂是3^19 = 1162261467 - 如果n是3的幂，那么最大3的幂应该能被n整除

代码实现 (Code Implementation)

```python
import math

def isPowerOfThree(n):
    """
    Check if number is power of three using iterative division.

    Args:
        n: int - Number to check

    Returns:
        bool - True if power of three, False otherwise
    """
    if n <= 0:
        return False

    while n % 3 == 0:
        n //= 3

    return n == 1

def isPowerOfThreeRecursive(n):
    """
    Recursive approach to check power of three.
    """
    if n <= 0:
        return False
    if n == 1:
        return True
    if n % 3 != 0:
        return False

    return isPowerOfThreeRecursive(n // 3)

def isPowerOfThreeMath(n):
    """
    Mathematical approach using largest power of 3 in 32-bit range.
    """
    if n <= 0:
        return False

    # Largest power of 3 in 32-bit signed integer range
    max_power_of_3 = 3 ** 19  # 1162261467

    return max_power_of_3 % n == 0

def isPowerOfThreeLog(n):
    """
    Using logarithm to check (may have precision issues).
    """
    if n <= 0:
        return False

    # Calculate log_3(n) = log(n) / log(3)
    log_result = math.log10(n) / math.log10(3)

    # Check if result is close to an integer
    return abs(log_result - round(log_result)) < 1e-10
```

```python
def isPowerOfThreeString(n):
    """
    Convert to base 3 and check pattern (educational purpose).
    """
    if n <= 0:
        return False

    # Convert to base 3
    base3 = ""
    temp = n
    while temp > 0:
        base3 = str(temp % 3) + base3
        temp //= 3

    # Power of 3 in base 3 should be "1" followed by zeros
    return base3[0] == '1' and all(c == '0' for c in base3[1:])

def generatePowersOfThree(max_val):
    """
    Generate all powers of 3 up to max_val.
    """
    powers = []
    power = 1
    while power <= max_val:
        powers.append(power)
        power *= 3
    return powers

# Test cases
test_cases = [27, 0, -1, 1, 9, 45, 81, 243, 244]

# Generate powers of 3 for reference
powers_of_3 = generatePowersOfThree(10000)
print(f"Powers of 3 up to 10000: {powers_of_3}\n")

for n in test_cases:
    result1 = isPowerOfThree(n)
    result2 = isPowerOfThreeRecursive(n)
    result3 = isPowerOfThreeMath(n)
    result4 = isPowerOfThreeLog(n)
    result5 = isPowerOfThreeString(n)

    print(f"n={n}")
    print(f"Iterative: {result1}, Recursive: {result2}, Math: {result3}")
    print(f"Log: {result4}, String: {result5}")
    print(f"Expected: {n in powers_of_3}\n")

# Demonstrate base 3 conversion
print("Base 3 representations:")
for power in [1, 3, 9, 27, 81]:
    base3 = ""
    temp = power
    while temp > 0:
        base3 = str(temp % 3) + base3
        temp //= 3
    print(f"{power} in base 3: {base3}")
```

关键考点 (Key Points)

1. **多种解法**: 循环、递归、数学、对数等不同方法
2. **边界条件**: 负数、0、1的特殊处理
3. **数学性质**: 利用3的幂的数学特性

Problem 18: Fizz Buzz (Easy)

Problem Statement

```
 Given an integer n, return a string array answer (1-indexed) where:
- answer[i] == "FizzBuzz" if i is divisible by 3 and 5.
- answer[i] == "Fizz" if i is divisible by 3.
- answer[i] == "Buzz" if i is divisible by 5.
- answer[i] == i (as a string) if none of the above conditions are true.

Example 1:
Input: n = 3
Output: ["1","2","Fizz"]

Example 2:
Input: n = 5
Output: ["1","2","Fizz","4","Buzz"]

Example 3:
Input: n = 15
Output:
["1","2","Fizz","4","Buzz","Fizz","7","8","Fizz","Buzz","11","Fizz","13","14","Fi

Constraints:
- 1 <= n <= 10^4
```

解题思路 (Solution Analysis)

这是一道经典的**条件判断**题目，看似简单但有很多优化和扩展的方法。

基础解法： - 遍历1到n，对每个数字检查整除条件 - 按优先级返回对应字符串

优化解法： - 使用字符串拼接避免多次条件判断 - 使用映射表处理多个条件

代码实现 (Code Implementation)

```
def fizzBuzz(n):
    """
    Generate FizzBuzz sequence up to n.

    Args:
        n: int - Upper bound (inclusive)

    Returns:
        List[str] - FizzBuzz sequence
    """
    result = []

    for i in range(1, n + 1):
        if i % 15 == 0:  # Divisible by both 3 and 5
            result.append("FizzBuzz")
        elif i % 3 == 0:
            result.append("Fizz")
        elif i % 5 == 0:
            result.append("Buzz")
        else:
            result.append(str(i))

    return result

def fizzBuzzStringConcat(n):
    """
    Using string concatenation approach.
    """
    result = []

    for i in range(1, n + 1):
        output = ""

        if i % 3 == 0:
            output += "Fizz"
        if i % 5 == 0:
            output += "Buzz"

        if not output:
            output = str(i)

        result.append(output)

    return result

def fizzBuzzMapping(n):
    """
    Using mapping for extensibility.
    """
    # Mapping of divisor to string
    mappings = {3: "Fizz", 5: "Buzz"}
    result = []

    for i in range(1, n + 1):
        output = ""

        for divisor, word in mappings.items():
            if i % divisor == 0:
```

```
                output += word

        if not output:
            output = str(i)

        result.append(output)

    return result

def fizzBuzzExtended(n, rules=None):
    """
    Extended version that accepts custom rules.

    Args:
        n: int - Upper bound
        rules: dict - Custom divisor to string mapping
    """
    if rules is None:
        rules = {3: "Fizz", 5: "Buzz"}

    result = []

    for i in range(1, n + 1):
        output = ""

        # Sort by divisor to ensure consistent order
        for divisor in sorted(rules.keys()):
            if i % divisor == 0:
                output += rules[divisor]

        if not output:
            output = str(i)

        result.append(output)

    return result

def fizzBuzzOneLiner(n):
    """
    One-liner implementation using list comprehension.
    """
    return [
        "FizzBuzz" if i % 15 == 0 else
        "Fizz" if i % 3 == 0 else
        "Buzz" if i % 5 == 0 else
        str(i)
        for i in range(1, n + 1)
    ]

def fizzBuzzCounter(n):
    """
    Using counter approach to avoid modulo operations.
    """
    result = []
    fizz_count = buzz_count = 0

    for i in range(1, n + 1):
        fizz_count += 1
        buzz_count += 1
        output = ""

        if fizz_count == 3:
```

```
            output += "Fizz"
            fizz_count = 0

        if buzz_count == 5:
            output += "Buzz"
            buzz_count = 0

        if not output:
            output = str(i)

        result.append(output)

    return result

# Test cases
test_cases = [3, 5, 15, 20]

for n in test_cases:
    result1 = fizzBuzz(n)
    result2 = fizzBuzzStringConcat(n)
    result3 = fizzBuzzMapping(n)
    result4 = fizzBuzzOneLiner(n)
    result5 = fizzBuzzCounter(n)

    print(f"n={n}")
    print(f"Basic: {result1}")
    print(f"All methods produce same result: {result1 == result2 == result3 ==
result4 == result5}")
    print()

# Extended example with custom rules
print("Extended FizzBuzz with custom rules:")
custom_rules = {3: "Fizz", 5: "Buzz", 7: "Bang"}
extended_result = fizzBuzzExtended(21, custom_rules)
print(f"With rules {custom_rules}: {extended_result}")
```

关键考点 (Key Points)

1. **条件判断**: 正确处理多个条件的优先级
2. **代码扩展性**: 设计易于扩展的代码结构
3. **性能优化**: 避免重复的模运算

Problem 19: Excel Sheet Column Number (Easy)

Problem Statement

```
 Given a string columnTitle that represents the column title as appears in an
Excel
sheet, return its corresponding column number.

For example:
A -> 1
B -> 2
C -> 3
...
Z -> 26
AA -> 27
AB -> 28
...

Example 1:
Input: columnTitle = "A"
Output: 1

Example 2:
Input: columnTitle = "AB"
Output: 28

Example 3:
Input: columnTitle = "ZY"
Output: 701

Constraints:
- 1 <= columnTitle.length <= 7
- columnTitle consists only of uppercase English letters.
- columnTitle is in the range ["A", "FXSHRXW"].
```

解题思路 (Solution Analysis)

这是一道**进制转换**题目，类似于26进制转换，但有特殊性。

核心思想： - Excel列号是一种特殊的26进制系统 - 没有0，从A(1)到Z(26) - 每一位的权重是26的幂次

算法步骤： 1. 从右到左处理每个字符 2. 将字符转换为对应数字(A=1, B=2, ..., Z=26) 3. 乘以对应的权重(26^0, 26^1, 26^2, ...) 4. 累加得到最终结果

代码实现 (Code Implementation)

```python
def titleToNumber(columnTitle):
    """
    Convert Excel column title to number.

    Args:
        columnTitle: str - Excel column title

    Returns:
        int - Corresponding column number
    """
    result = 0

    for char in columnTitle:
        # Convert character to number (A=1, B=2, ..., Z=26)
        digit = ord(char) - ord('A') + 1
        # Multiply previous result by 26 and add current digit
        result = result * 26 + digit

    return result

def titleToNumberExplicit(columnTitle):
    """
    More explicit version showing the base-26 conversion.
    """
    result = 0
    length = len(columnTitle)

    for i, char in enumerate(columnTitle):
        digit = ord(char) - ord('A') + 1
        power = length - i - 1
        result += digit * (26 ** power)

    return result

def titleToNumberRecursive(columnTitle):
    """
    Recursive approach.
    """
    if not columnTitle:
        return 0

    # Process last character
    last_char = columnTitle[-1]
    last_digit = ord(last_char) - ord('A') + 1

    # Recursively process remaining characters
    remaining = titleToNumberRecursive(columnTitle[:-1])

    return remaining * 26 + last_digit

def numberToTitle(columnNumber):
    """
    Reverse operation: convert number to Excel column title.
    """
    result = ""

    while columnNumber > 0:
        # Adjust for 1-based indexing
```

```
        columnNumber -= 1
        remainder = columnNumber % 26
        result = chr(ord('A') + remainder) + result
        columnNumber //= 26

    return result

def demonstrateConversion(columnTitle):
    """
    Demonstrate step-by-step conversion process.
    """
    print(f"Converting '{columnTitle}' to number:")
    result = 0

    for i, char in enumerate(columnTitle):
        digit = ord(char) - ord('A') + 1
        power = len(columnTitle) - i - 1
        contribution = digit * (26 ** power)
        result += contribution

        print(f"  {char} = {digit}, position {power}: {digit} × 26^{power} =
{contribution}")

    print(f"  Total: {result}")
    return result

# Test cases
test_cases = ["A", "AB", "ZY", "AAA", "FXSHRXW"]

for title in test_cases:
    result1 = titleToNumber(title)
    result2 = titleToNumberExplicit(title)
    result3 = titleToNumberRecursive(title)

    # Verify reverse conversion
    reverse = numberToTitle(result1)

    print(f"Title: '{title}'")
    print(f"Methods: {result1}, {result2}, {result3} (all same: {result1 ==
result2 == result3})")
    print(f"Reverse: {result1} -> '{reverse}' (correct: {reverse == title})")
    print()

# Detailed demonstration
print("Detailed conversion process:")
demonstrateConversion("ZY")
print()

# Show pattern for first few columns
print("Excel column pattern:")
for i in range(1, 30):
    title = numberToTitle(i)
    print(f"{i:2d} -> {title}")
```

关键考点 (Key Points)

1. **进制转换**: 理解特殊的26进制系统

2. **字符处理**: ASCII码转换和字符映射
3. **逆向思维**: 理解正向和反向转换的关系

Problem 20: Majority Element (Easy)

Problem Statement

```
Given an array nums of size n, return the majority element.

The majority element is the element that appears more than ⌊n / 2⌋ times. You
may
assume that the majority element always exists in the array.

Example 1:
Input: nums = [3,2,3]
Output: 3

Example 2:
Input: nums = [2,2,1,1,1,2,2]
Output: 2

Constraints:
- n == nums.length
- 1 <= n <= 5 * 10^4
- -10^9 <= nums[i] <= 10^9
```

解题思路 (Solution Analysis)

这是一道经典的**Boyer-Moore投票算法**题目，有多种解法。

Boyer-Moore投票算法： - 维护一个候选者和计数器 - 遇到相同元素计数器+1，不同元素计数器-1 - 计数器为0时更换候选者 - 最后的候选者就是多数元素

其他解法： - 哈希表统计频次 - 排序后取中位数 - 分治算法

代码实现 (Code Implementation)

```python
from collections import Counter

def majorityElement(nums):
    """
    Find majority element using Boyer-Moore voting algorithm.

    Args:
        nums: List[int] - Array of integers

    Returns:
        int - The majority element
    """
    candidate = None
    count = 0

    # Phase 1: Find candidate
    for num in nums:
        if count == 0:
            candidate = num
        count += 1 if num == candidate else -1

    # Phase 2: Verify candidate (not needed given problem constraints)
    # count = sum(1 for num in nums if num == candidate)
    # return candidate if count > len(nums) // 2 else None

    return candidate

def majorityElementHashMap(nums):
    """
    Using hash map to count frequencies.
    """
    count = Counter(nums)
    return count.most_common(1)[0][0]

def majorityElementSorting(nums):
    """
    Sort and return middle element.
    """
    nums.sort()
    return nums[len(nums) // 2]

def majorityElementDivideConquer(nums):
    """
    Divide and conquer approach.
    """
    def majority_rec(left, right):
        # Base case
        if left == right:
            return nums[left]

        # Divide
        mid = (left + right) // 2
        left_majority = majority_rec(left, mid)
        right_majority = majority_rec(mid + 1, right)

        # Conquer
        if left_majority == right_majority:
            return left_majority
```

```
        # Count occurrences in current range
        left_count = sum(1 for i in range(left, right + 1) if nums[i] ==
left_majority)
        right_count = sum(1 for i in range(left, right + 1) if nums[i] ==
right_majority)

        return left_majority if left_count > right_count else right_majority

    return majority_rec(0, len(nums) - 1)

def demonstrateVoting(nums):
    """
    Demonstrate Boyer-Moore voting algorithm step by step.
    """
    print(f"Demonstrating voting algorithm on {nums}:")
    candidate = None
    count = 0

    for i, num in enumerate(nums):
        old_candidate, old_count = candidate, count

        if count == 0:
            candidate = num
        count += 1 if num == candidate else -1

        print(f"Step {i+1}: num={num}, candidate: {old_candidate}->{candidate},
count: {old_count}->{count}")

    print(f"Final candidate: {candidate}\n")
    return candidate

# Test cases
test_cases = [
    [3, 2, 3],
    [2, 2, 1, 1, 1, 2, 2],
    [1],
    [1, 1, 1, 2, 2],
    [6, 5, 5]
]

for nums in test_cases:
    result1 = majorityElement(nums)
    result2 = majorityElementHashMap(nums)
    result3 = majorityElementSorting(nums.copy())  # Copy because sorting
modifies array
    result4 = majorityElementDivideConquer(nums)

    print(f"nums={nums}")
    print(f"Voting: {result1}, HashMap: {result2}, Sorting: {result3}, D&C:
{result4}")
    print(f"All same: {result1 == result2 == result3 == result4}")
    print()

# Detailed demonstration
print("Detailed voting algorithm demonstration:")
demonstrateVoting([2, 2, 1, 1, 1, 2, 2])

# Performance analysis
import time

def benchmark_methods(nums):
```

```
    """
    Compare performance of different methods.
    """
    methods = [
        ("Boyer-Moore", majorityElement),
        ("HashMap", majorityElementHashMap),
        ("Sorting", lambda x: majorityElementSorting(x.copy())),
        ("Divide & Conquer", majorityElementDivideConquer)
    ]

    for name, method in methods:
        start_time = time.time()
        result = method(nums)
        end_time = time.time()
        print(f"{name}: {result}, {end_time - start_time:.6f} seconds")

# Large test case for performance comparison
large_nums = [1] * 25000 + [2] * 24999
print(f"\nPerformance benchmark (array size: {len(large_nums)}):")
benchmark_methods(large_nums)
```

关键考点 (Key Points)

1. **Boyer-Moore算法**: 理解投票算法的原理和实现
2. **多种解法**: 掌握不同方法的优缺点
3. **时间空间复杂度**: 各种方法的复杂度分析
