75 DSA Questions from Leet-Code

BY-Syntax Error

1. Arrays (10 Questions)

1. **1. Two Sum**
2. **121. Best Time to Buy and Sell Stock**
3. **88. Merge Sorted Array**
4. **217. Contains Duplicate**
5. **238. Product of Array Except Self**
6. **53. Maximum Subarray**
7. **15. 3Sum**
8. **56. Merge Intervals**
9. **11. Container With Most Water**
10. **48. Rotate Image**

Solution:

1. Two Sum (LeetCode 1)

Problem: Find two numbers in the array that add up to a specific target.
Solution:

```
python
for i, n in enumerate(nums):
    diff = target - n
    if diff in seen:
        return [seen[diff], i]
    seen[n] = i
```

---

2. Best Time to Buy and Sell Stock (LeetCode 121)

Problem: Maximize profit by choosing one day to buy and another to sell.
Solution:

```
python
min_price, max_profit = inf, 0

for p in prices:
    min_price = min(min_price, p)
```

```
    max_profit = max(max_profit, p - min_price)
return max_profit
```

---

3. Merge Sorted Array (LeetCode 88)

Problem: Merge two sorted arrays into one sorted array.
Solution:

```
python
while m > 0 and n > 0:
    if nums1[m-1] > nums2[n-1]:
        nums1[m+n-1] = nums1[m-1]; m -= 1
    else:
        nums1[m+n-1] = nums2[n-1]; n -= 1
nums1[:n] = nums2[:n]
```

---

4. Contains Duplicate (LeetCode 217)

Problem: Check if an array contains duplicates.
Solution:

```
python
return len(nums) != len(set(nums))
```

---

5. Product of Array Except Self (LeetCode 238)

Problem: Return an array where each element is the product of all other elements.
Solution:

```
python
res = [1]*len(nums)
prefix = 1
for i in range(len(nums)):
    res[i] = prefix
    prefix *= nums[i]
suffix = 1
for i in range(len(nums)-1, -1, -1):
```

```
    res[i] *= suffix
    suffix *= nums[i]
```

---

6. Maximum Subarray (LeetCode 53)

Problem: Find the contiguous subarray with the largest sum.
Solution:

```
python
cur = best = nums[0]
for n in nums[1:]:
    cur = max(n, cur + n)
    best = max(best, cur)
return best
```

---

7. 3Sum (LeetCode 15)

Problem: Find all unique triplets in the array which gives the sum of zero.
Solution:

```
python
nums.sort()
for i in range(len(nums)-2):
    if i > 0 and nums[i] == nums[i-1]: continue
    l, r = i+1, len(nums)-1
    while l < r:
        s = nums[i] + nums[l] + nums[r]
        if s < 0: l += 1
        elif s > 0: r -= 1
        else:
            res.append([nums[i], nums[l], nums[r]])
            l += 1; r -= 1
            while l < r and nums[l] == nums[l-1]: l += 1
```

---

8. Merge Intervals (LeetCode 56)

Problem: Merge overlapping intervals.P
Solution:

```
python
intervals.sort(key=lambda x: x[0])

merged = [intervals[0]]

for s, e in intervals[1:]:

   if s <= merged[-1][1]:

      merged[-1][1] = max(merged[-1][1], e)

   else:

      merged.append([s, e])
```

---

9. Container With Most Water (LeetCode 11)

Problem: Find the maximum water that can be trapped between two lines.
Solution:

```
python
l, r = 0, len(h)-1

best = 0

while l < r:

   area = min(h[l], h[r]) * (r - l)

   best = max(best, area)

   if h[l] < h[r]: l += 1

   else: r -= 1
```

---

10. Rotate Image (LeetCode 48)

Problem: Rotate a matrix 90 degrees clockwise.
Solution:

```
Python
matrix[:] = zip(*matrix[::-1])
```

2. Strings (10 Questions)

1. **20. Valid Parentheses**
2. **125. Valid Palindrome**
3. **242. Valid Anagram**

4. **49. Group Anagrams**
5. **5. Longest Palindromic Substring**
6. **76. Minimum Window Substring**
7. **28. Find the Index of the First Occurrence in a String**
8. **443. String Compression**
9. **14. Longest Common Prefix**
10. **459. Repeated Substring Pattern**

Solution:

1. Valid Parentheses (LeetCode 20)

```
python
stack = []

pairs = {')': '(', ']': '[', '}': '{'}

for c in s:

   if c in pairs:

      if not stack or stack.pop() != pairs[c]: return False

   else:

      stack.append(c)

return not stack
```

---

2. Valid Palindrome (LeetCode 125)

```
python
filtered = [c.lower() for c in s if c.isalnum()]

return filtered == filtered[::-1]
```

---

3. Valid Anagram (LeetCode 242)

```
python
return Counter(s) == Counter(t)
```

---

4. Group Anagrams (LeetCode 49)

```
python
res = defaultdict(list)

for w in strs:
```

```
    key = tuple(sorted(w))
    res[key].append(w)
return list(res.values())
```

---

5. Longest Palindromic Substring (LeetCode 5)

```
python
for i in range(len(s)):
    for a, b in [(i, i), (i, i+1)]:
        while a >= 0 and b < len(s) and s[a] == s[b]:
            if b - a + 1 > len(ans): ans = s[a:b+1]
            a -= 1; b += 1
```

---

6. Minimum Window Substring (LeetCode 76)

```
python
need, missing = Counter(t), len(t)
l = start = end = 0
for r, ch in enumerate(s, 1):
    if need[ch] > 0: missing -= 1
    need[ch] -= 1
    while missing == 0:
        if not end or r - l < end - start: start, end = l, r
        need[s[l]] += 1
        if need[s[l]] > 0: missing += 1
        l += 1
```

---

7. Find the Index of the First Occurrence (LeetCode 28)

```
python
for i in range(len(haystack) - len(needle) + 1):
    if haystack[i:i+len(needle)] == needle:
        return i
return -1
```

---

8. String Compression (LeetCode 443)

```
python
i = write = 0

while i < len(chars):

   ch, count = chars[i], 0

   while i < len(chars) and chars[i] == ch:

      i += 1; count += 1

   chars[write] = ch; write += 1

   if count > 1:

      for c in str(count):

         chars[write] = c; write += 1
```

---

9. Longest Common Prefix (LeetCode 14)

```
python
prefix = strs[0]

for s in strs[1:]:

   while not s.startswith(prefix):

      prefix = prefix[:-1]

return prefix
```

---

10. Repeated Substring Pattern (LeetCode 459)

```
Python
return s in (s + s)[1:-1]
```

Linked Lists (8 Questions)

1. **206. Reverse Linked List**
2. **21. Merge Two Sorted Lists**
3. **19. Remove Nth Node From End of List**
4. **141. Linked List Cycle**
5. **2. Add Two Numbers**
6. **160. Intersection of Two Linked Lists**
7. **234. Palindrome Linked List**
8. **25. Reverse Nodes in k-Group**

Soluition:

Linked Lists

1. Reverse Linked List (LeetCode 206)

```
python
prev, curr = None, head

while curr:

    nxt = curr.next

    curr.next = prev

    prev = curr

    curr = nxt

return prev
```

---

2. Merge Two Sorted Lists (LeetCode 21)

```
python
dummy = tail = ListNode()

while l1 and l2:

    if l1.val < l2.val:

        tail.next, l1 = l1, l1.next

    else:

        tail.next, l2 = l2, l2.next

    tail = tail.next

tail.next = l1 or l2

return dummy.next
```

---

3. Remove Nth Node From End of List (LeetCode 19)

```
python
dummy = ListNode(0, head)

fast = slow = dummy

for _ in range(n): fast = fast.next

while fast.next:
```

```
    fast, slow = fast.next, slow.next
slow.next = slow.next.next
return dummy.next
```

---

4. Linked List Cycle (LeetCode 141)

```
python
slow = fast = head
while fast and fast.next:
    slow, fast = slow.next, fast.next.next
    if slow == fast: return True
return False
```

---

5. Add Two Numbers (LeetCode 2)

```
python
dummy = curr = ListNode()
carry = 0
while l1 or l2 or carry:
    v1 = l1.val if l1 else 0
    v2 = l2.val if l2 else 0
    carry, val = divmod(v1 + v2 + carry, 10)
    curr.next = ListNode(val)
    curr = curr.next
    l1 = l1.next if l1 else None
    l2 = l2.next if l2 else None
return dummy.next
```

---

6. Intersection of Two Linked Lists (LeetCode 160)

```
python
a, b = headA, headB
while a != b:
    a = a.next if a else headB
```

```
    b = b.next if b else headA
return a
```

---

7. Palindrome Linked List (LeetCode 234)

```
python
slow = fast = head
while fast and fast.next:
    slow, fast = slow.next, fast.next.next
prev = None
while slow:
    nxt = slow.next
    slow.next = prev
    prev = slow
    slow = nxt
first, second = head, prev
while second:
    if first.val != second.val: return False
    first, second = first.next, second.next
return True
```

---

8. Reverse Nodes in k-Group (LeetCode 25)

```
Python
def reverse_k(head, k):
    prev, curr = None, head
    for _ in range(k):
        if not curr: return head
        curr = curr.next
    curr = head
    for _ in range(k):
        nxt = curr.next
        curr.next = prev
        prev, curr = curr, nxt
    head.next = reverse_k(curr, k)
    return prev
```

4. Stacks and Queues (6 Questions)

1. **232. Implement Queue using Stacks**
2. **225. Implement Stack using Queues**
3. **155. Min Stack**
4. **739. Daily Temperatures**
5. **150. Evaluate Reverse Polish Notation**
6. **496. Next Greater Element Is**
7. **503. Next Greater Element II**
8. **622.Circular Queue**

Solution:

Stacks and Queues

---

1. Implement Queue Using Stacks (LeetCode 232)

```
Python
def push(x):
    s1.append(x)

def pop():
    while len(s1) > 1:
        s2.append(s1.pop())
    val = s1.pop()
    while s2:
        s1.append(s2.pop())
    return val
```

---

2. Implement Stack Using Queues (LeetCode 225)

```
Python
def push(x):
    q.append(x)
    for _ in range(len(q) - 1):
        q.append(q.popleft())
```

```
def pop():
    return q.popleft()
```

---

3. Min Stack (LeetCode 155)

```
Python
def push(x):
    stack.append(x)
    if not min_stack or x <= min_stack[-1]:
        min_stack.append(x)

def pop():
    val = stack.pop()
    if val == min_stack[-1]:
        min_stack.pop()
```

---

4. Daily Temperatures (LeetCode 739)

```
Python
for i, t in enumerate(temps):
    while stack and temps[stack[-1]] < t:
        idx = stack.pop()
        res[idx] = i - idx
    stack.append(i)
```

---

5. Evaluate Reverse Polish Notation (LeetCode 150)

```
Python
for token in tokens:
    if token not in '+-*/':
        stack.append(int(token))
    else:
        b, a = stack.pop(), stack.pop()
        if token == '+': stack.append(a + b)
        elif token == '-': stack.append(a - b)
        elif token == '*': stack.append(a * b)
        else: stack.append(int(a / b))
```

---

6. Next Greater Element I (LeetCode 496)

```
Python
for num in nums2:
    while stack and num > stack[-1]:
        val = stack.pop()
        next_greater[val] = num
    stack.append(num)
```

---

7. Next Greater Element II (LeetCode 503)

```
Python
for i in range(2 * n - 1, -1, -1):
    while stack and nums[stack[-1]] <= nums[i % n]:
        stack.pop()
    res[i % n] = nums[stack[-1]] if stack else -1
    stack.append(i % n)
```

---

8. Circular Queue (LeetCode 622)

```
Python
def enQueue(val):
    if (rear + 1) % size == front:
        return False
    queue[rear] = val
    rear = (rear + 1) % size
    return True

def deQueue():
    if front == rear:
        return False
    front = (front + 1) % size
    return True
```

5. Binary Search (6 Questions)

1. **704. Binary Search**
2. **34. Find First and Last Position of Element in Sorted Array**
3. **74. Search a 2D Matrix**
4. **33. Search in Rotated Sorted Array**
5. **81. Search in Rotated Sorted Array II**
6. **162. Find Peak Element**

Solution:

704. Binary Search

```
python
l, r = 0, len(nums) - 1

while l <= r:

   mid = (l + r) // 2

   if nums[mid] == target: return mid
```

```
    elif nums[mid] < target: l = mid + 1
    else: r = mid - 1
return -1
```

---

34. Find First and Last Position of Element in Sorted Array

```
python
def findBound(isFirst):
    l, r, ans = 0, len(nums) - 1, -1
    while l <= r:
        mid = (l + r) // 2
        if nums[mid] == target:
            ans = mid
            if isFirst: r = mid - 1
            else: l = mid + 1
        elif nums[mid] < target: l = mid + 1
        else: r = mid - 1
    return ans


return [findBound(True), findBound(False)]
```

---

74. Search a 2D Matrix

```
python
rows, cols = len(matrix), len(matrix[0])
l, r = 0, rows * cols - 1
while l <= r:
    mid = (l + r) // 2
    val = matrix[mid // cols][mid % cols]
    if val == target: return True
    elif val < target: l = mid + 1
    else: r = mid - 1
return False
```

---

33. Search in Rotated Sorted Array

```python
l, r = 0, len(nums) - 1

while l <= r:

    mid = (l + r) // 2

    if nums[mid] == target: return mid

    if nums[l] <= nums[mid]:

        if nums[l] <= target < nums[mid]: r = mid - 1

        else: l = mid + 1

    else:

        if nums[mid] < target <= nums[r]: l = mid + 1

        else: r = mid - 1

return -1
```

---

81. Search in Rotated Sorted Array II

```python
l, r = 0, len(nums) - 1

while l <= r:

    mid = (l + r) // 2

    if nums[mid] == target: return True

    if nums[l] == nums[mid] == nums[r]:

        l += 1; r -= 1

    elif nums[l] <= nums[mid]:

        if nums[l] <= target < nums[mid]: r = mid - 1

        else: l = mid + 1

    else:

        if nums[mid] < target <= nums[r]: l = mid + 1

        else: r = mid - 1

return False
```

---

162. Find Peak Element

```
Python
l, r = 0, len(nums) - 1
while l < r:
    mid = (l + r) // 2
    if nums[mid] > nums[mid + 1]: r = mid
    else: l = mid + 1
return l
```

6. Trees (8 Questions)

1. **104. Maximum Depth of Binary Tree**
2. **100. Same Tree**
3. **101. Symmetric Tree**
4. **144. Binary Tree Preorder Traversal**
5. **94. Binary Tree Inorder Traversal**
6. **145. Binary Tree Postorder Traversal**
7. **102. Binary Tree Level Order Traversal**
8. **110. Balanced Binary Tree**

Solution:

104. Maximum Depth of Binary Tree

```
python
if not root: return 0

return 1 + max(dfs(root.left), dfs(root.right))
```

---

100. Same Tree

```
python
if not p and not q: return True

if not p or not q or p.val != q.val: return False

return same(p.left, q.left) and same(p.right, q.right)
```

---

101. Symmetric Tree

```python
def isMirror(t1, t2):
    if not t1 and not t2: return True
    if not t1 or not t2 or t1.val != t2.val: return False
    return isMirror(t1.left, t2.right) and isMirror(t1.right, t2.left)
```

---

144. Binary Tree Preorder Traversal

```python
if not root: return []
stack, res = [root], []
while stack:
    node = stack.pop()
    res.append(node.val)
    if node.right: stack.append(node.right)
    if node.left: stack.append(node.left)
return res
```

---

94. Binary Tree Inorder Traversal

```python
stack, res = [], []
cur = root
while cur or stack:
    while cur:
        stack.append(cur)
        cur = cur.left
    cur = stack.pop()
    res.append(cur.val)
    cur = cur.right
return res
```

---

145. Binary Tree Postorder Traversal

```
python
if not root: return []

stack, res = [root], []

while stack:

   node = stack.pop()

   res.insert(0, node.val)

   if node.left: stack.append(node.left)

   if node.right: stack.append(node.right)

return res
```

---

102. Binary Tree Level Order Traversal

```
python
if not root: return []

q, res = [root], []

while q:

   level = []

   for _ in range(len(q)):

      node = q.pop(0)

      level.append(node.val)

      if node.left: q.append(node.left)

      if node.right: q.append(node.right)

   res.append(level)

return res
```

---

110. Balanced Binary Tree

```
python
def height(node):
    if not node: return 0
    left, right = height(node.left), height(node.right)
    if left == -1 or right == -1 or abs(left - right) > 1:
        return -1
    return 1 + max(left, right)
return height(root) != -1
```

7. Recursion and Backtracking (7 Questions)

1. **39. Combination Sum**
2. **46. Permutations**
3. **78. Subsets**
4. **51. N-Queens**
5. **17. Letter Combinations of a Phone Number**
6. **90. Subsets II**
7. **37. Sudoku Solver**

Solution:

39. Combination Sum

```
python
def backtrack(start, target, path):
    if target == 0:
        res.append(path[:])
        return
    for i in range(start, len(candidates)):
        if candidates[i] <= target:
            path.append(candidates[i])
            backtrack(i, target - candidates[i], path)
            path.pop()
```

---

46. Permutations

```
python
def backtrack(path, used):
    if len(path) == len(nums):
        res.append(path[:])
        return
    for i in range(len(nums)):
        if not used[i]:
            used[i] = True
            path.append(nums[i])
            backtrack(path, used)
            path.pop()
```

```
        used[i] = False
```

---

78. Subsets

```
python
def backtrack(index, path):
    res.append(path[:])
    for i in range(index, len(nums)):
        path.append(nums[i])
        backtrack(i + 1, path)
        path.pop()
```

---

51. N-Queens

```
python
def backtrack(row):
    if row == n:
        res.append([''.join(r) for r in board])
        return
    for col in range(n):
        if col in cols or (row + col) in diag1 or (row - col) in diag2:
            continue
        board[row][col] = 'Q'
        cols.add(col); diag1.add(row + col); diag2.add(row - col)
        backtrack(row + 1)
        board[row][col] = '.'
        cols.remove(col); diag1.remove(row + col); diag2.remove(row - col)
```

---

17. Letter Combinations of a Phone Number

```
python
def backtrack(index, path):
    if index == len(digits):
        res.append(''.join(path))
```

```
    return
  for ch in mapping[digits[index]]:
    path.append(ch)
    backtrack(index + 1, path)
    path.pop()
```

---

90. Subsets II

```
python
def backtrack(index, path):
  res.append(path[:])
  for i in range(index, len(nums)):
    if i > index and nums[i] == nums[i - 1]:
      continue
    path.append(nums[i])
    backtrack(i + 1, path)
    path.pop()
```

---

37. Sudoku Solver

```
Python
def backtrack():
    for r in range(9):
        for c in range(9):
            if board[r][c] == '.':
                for ch in '123456789':
                    if valid(r, c, ch):
                        board[r][c] = ch
                        if backtrack(): return True
                        board[r][c] = '.'
                return False
    return True
```

8. Dynamic Programming (10 Questions)

1. **70. Climbing Stairs**
2. **198. House Robber**
3. **322. Coin Change**
4. **300. Longest Increasing Subsequence**
5. **1143. Longest Common Subsequence**
6. **62. Unique Paths**
7. **5. Longest Palindromic Substring**

8. **718. Maximum Length of Repeated Subarray**
9. **416. Partition Equal Subset Sum**
10. **53. Maximum Subarray**

Solution:

70. Climbing Stairs

```
python
dp = [0] * (n + 1)

dp[0], dp[1] = 1, 1

for i in range(2, n + 1):

   dp[i] = dp[i - 1] + dp[i - 2]
```

---

198. House Robber

```
python
if not nums: return 0

if len(nums) == 1: return nums[0]

dp = [0] * len(nums)

dp[0], dp[1] = nums[0], max(nums[0], nums[1])

for i in range(2, len(nums)):

   dp[i] = max(dp[i - 1], dp[i - 2] + nums[i])
```

---

322. Coin Change

```
python
dp = [float('inf')] * (amount + 1)

dp[0] = 0

for coin in coins:

   for i in range(coin, amount + 1):

      dp[i] = min(dp[i], dp[i - coin] + 1)
```

---

300. Longest Increasing Subsequence

```
python
dp = [1] * len(nums)
```

```
for i in range(len(nums)):
    for j in range(i):
        if nums[i] > nums[j]:
            dp[i] = max(dp[i], dp[j] + 1)
```

1143. Longest Common Subsequence

```
python
dp = [[0]*(len(t)+1) for _ in range(len(s)+1)]
for i in range(1, len(s)+1):
    for j in range(1, len(t)+1):
        if s[i-1] == t[j-1]:
            dp[i][j] = dp[i-1][j-1] + 1
        else:
            dp[i][j] = max(dp[i-1][j], dp[i][j-1])
```

---

62. Unique Paths

```
python
dp = [[1]*n for _ in range(m)]
for i in range(1, m):
    for j in range(1, n):
        dp[i][j] = dp[i-1][j] + dp[i][j-1]
```

---

5. Longest Palindromic Substring

```
python
dp = [[False]*n for _ in range(n)]
start, max_len = 0, 1
for i in range(n-1, -1, -1):
    for j in range(i, n):
        if s[i] == s[j] and (j - i < 2 or dp[i+1][j-1]):
```

```
        dp[i][j] = True
        if j - i + 1 > max_len:
            start, max_len = i, j - i + 1
```

---

718. Maximum Length of Repeated Subarray

```
python
dp = [[0]*(len(B)+1) for _ in range(len(A)+1)]
res = 0
for i in range(1, len(A)+1):
    for j in range(1, len(B)+1):
        if A[i-1] == B[j-1]:
            dp[i][j] = dp[i-1][j-1] + 1
            res = max(res, dp[i][j])
```

---

416. Partition Equal Subset Sum

```
python
target = sum(nums) // 2
dp = [False] * (target + 1)
dp[0] = True
for num in nums:
    for i in range(target, num - 1, -1):
        dp[i] = dp[i] or dp[i - num]
```

---

53. Maximum Subarray

```
Python
cur = res = nums[0]
for num in nums[1:]:
    cur = max(num, cur + num)
    res = max(res, cur)
```

9. Graphs (6 Questions)

1. **133. Clone Graph**
2. **200. Number of Islands**
3. **207. Course Schedule**
4. **785. Is Graph Bipartite?**
5. **994. Rotting Oranges**
6. **323. Number of Connected Components in an Undirected Graph**

Solution:

133. Clone Graph

```
python
def dfs(node):
    if node in visited:
        return visited[node]
    clone = Node(node.val)
    visited[node] = clone
    for nei in node.neighbors:
        clone.neighbors.append(dfs(nei))
    return clone
```

---

200. Number of Islands

```
python
def dfs(r, c):
    if r < 0 or c < 0 or r >= rows or c >= cols or grid[r][c] != '1':
        return
    grid[r][c] = '0'
    for dr, dc in [(1,0), (-1,0), (0,1), (0,-1)]:
        dfs(r + dr, c + dc)
```

---

207. Course Schedule

```
python
def dfs(course):
    if course in visiting: return False
    if course in visited: return True
```

```
    visiting.add(course)
    for pre in graph[course]:
        if not dfs(pre): return False
    visiting.remove(course)
    visited.add(course)
    return True
```

---

785. Is Graph Bipartite?

```
Python
def dfs(node, color_val):
    color[node] = color_val
    for nei in graph[node]:
        if nei in color:
            if color[nei] == color_val:
                return False
        else:
            if not dfs(nei, 1 - color_val):
                return False
    return True
```

---

994. Rotting Oranges

```
python
while queue:
    for _ in range(len(queue)):
        r, c = queue.popleft()
        for dr, dc in [(1,0), (-1,0), (0,1), (0,-1)]:
            nr, nc = r + dr, c + dc
            if 0 <= nr < m and 0 <= nc < n and grid[nr][nc] == 1:
                grid[nr][nc] = 2
                queue.append((nr, nc))
    minutes += 1
```

---

323. Number of Connected Components in an Undirected Graph

```
Python
def dfs(node):
    for nei in graph[node]:
```

```
        if nei not in visited:
            visited.add(nei)
            dfs(nei)
```

10. Bit Manipulation (4 Questions)

1. **136. Single Number**
2. **190. Reverse Bits**
3. **191. Number of 1 Bits**
4. **268. Missing Number**

Solution:

136. Single Number

```
Python
res = 0
for num in nums:
    res ^= num
```

Explanation: XOR operation cancels out numbers that appear twice, leaving only the number that appears once.

---

190. Reverse Bits

```
Python
res = 0
for i in range(32):
    res = (res << 1) | (n & 1)
    n >>= 1
```

Explanation: Process each bit of the number, shifting the result left and appending the current bit of `n` to the result.

---

191. Number of 1 Bits

```
Python
count = 0
while n:
    n &= (n - 1)
    count += 1
```

Explanation: Count the number of set bits (1s) by repeatedly masking the least significant bit and shifting the number right.

---

268. Missing Number

```
Python
res = len(nums)
for i in range(len(nums)):
    res ^= i ^ nums[i]
```