https://leetcode.com/problem-list/vrfi9h41/

Q1. Missing Number

Aim:

Find the missing number from `0..n` in an array of size `n`.

Description:

We XOR all numbers `1..n` and all elements of the array. Their XOR gives the missing number.

Full Code:

```
class Solution:
    def missingNumber(self, nums):
        xor1 = 0
        xor2 = 0
        for i in range(len(nums)):
            xor1 = xor1 ^ (i + 1)
            xor2 = xor2 ^ nums[i]
        return xor1 ^ xor2


if __name__ == "__main__":
    nums = [3, 0, 1]
    print("Missing Number:", Solution().missingNumber(nums))
```

Complexity:

- Time: O(n)
- Space: O(1)

Output:

```
Missing Number: 2
```

Q2. Hamming Weight (Number of 1 Bits)

Aim:
Count number of 1s in binary representation of a number.

Description:
Use Brian Kernighan's method: repeatedly clear the lowest set bit using `n & (n-1)`.

Full Code:

```
class Solution:
    def hammingWeight(self, n):
        cnt = 0
        while n > 0:
            cnt += 1
            n = n & (n - 1)
        return cnt


if __name__ == "__main__":
    n = 11   # binary 1011
    print("Hamming Weight:", Solution().hammingWeight(n))
```

Complexity:

- Time: O(k), where k = number of set bits
- Space: O(1)

Output:

```
Hamming Weight: 3
```

Q3. Middle of the Linked List

Aim:
Find the middle node of a linked list.

Description:
Use slow and fast pointers; when fast reaches end, slow is at the middle.

Full Code:

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

class Solution:
    def middleNode(self, head):
        slow = head
        fast = head
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next
        return slow


if __name__ == "__main__":
    head = ListNode(1, ListNode(2, ListNode(3, ListNode(4,
ListNode(5)))))
    mid = Solution().middleNode(head)
    print("Middle Node:", mid.val)
```

Complexity:

- Time: O(n)
- Space: O(1)

Output:

```
Middle Node: 3
```

Q4. Linked List Cycle II

Aim:
Detect the node where a cycle begins in a linked list.

Description:
Use Floyd’s cycle detection algorithm (slow and fast pointers).

Full Code:

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

class Solution:
    def detectCycle(self, head):
        slow = head
        fast = head
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next
            if slow == fast:
                slow = head
                while slow != fast:
                    slow = slow.next
                    fast = fast.next
                return slow
        return None


if __name__ == "__main__":
```

```
    node4 = ListNode(-4)
    node3 = ListNode(0, node4)
    node2 = ListNode(2, node3)
    node1 = ListNode(3, node2)
    node4.next = node2  # create cycle
    cycle_node = Solution().detectCycle(node1)
    print("Cycle starts at:", cycle_node.val if cycle_node
else None)
```

Complexity:

- Time: O(n)
- Space: O(1)

Output:

```
Cycle starts at: 2
```

Q5. Remove Nth Node From End

Aim:

Remove the nth node from the end of a linked list.

Description:

Advance `fast` pointer by `n`, then move both until fast reaches end. Delete `slow.next`.

Full Code:

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

class Solution:
    def removeNthFromEnd(self, head, n):
        if not head or not head.next:
            return None
        slow = head
        fast = head
        for _ in range(n):
            fast = fast.next
        if not fast:
            return head.next
        while fast.next:
            slow = slow.next
            fast = fast.next
        slow.next = slow.next.next
        return head
```

```
if __name__ == "__main__":
    head = ListNode(1, ListNode(2, ListNode(3, ListNode(4, ListNode(5)))))
    new_head = Solution().removeNthFromEnd(head, 2)
    curr = new_head
    while curr:
        print(curr.val, end=" -> ")
        curr = curr.next
    print("None")
```

Complexity:

- Time: O(n)
- Space: O(1)

Output:

```
1 -> 2 -> 3 -> 5 -> None
```

Q6. Merge Two Sorted Lists

Aim:
Merge two sorted linked lists into a single sorted list.

Description:
Compare nodes one by one using two pointers and a dummy head.

Full Code:

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


class Solution:
    def mergeTwoLists(self, list1, list2):
        ptr1 = list1
        ptr2 = list2
        dummy = ListNode(0)
        curr = dummy
        while ptr1 and ptr2:
            if ptr1.val <= ptr2.val:
                curr.next = ptr1
                ptr1 = ptr1.next
            else:
                curr.next = ptr2
                ptr2 = ptr2.next
            curr = curr.next
        curr.next = ptr1 if ptr1 else ptr2
        return dummy.next
```

```
if __name__ == "__main__":
    l1 = ListNode(1, ListNode(2, ListNode(4)))
    l2 = ListNode(1, ListNode(3, ListNode(5)))
    merged = Solution().mergeTwoLists(l1, l2)
    while merged:
        print(merged.val, end=" -> ")
        merged = merged.next
    print("None")
```

Complexity:

- Time: O(m+n)
- Space: O(1)

Output:

```
1 -> 1 -> 2 -> 3 -> 4 -> 5 -> None
```

Q7. Daily Temperatures

Aim:
Find how many days you must wait for a warmer temperature.

Description:
Use a stack to track indices of decreasing temperatures, traverse from right to left.

Full Code:

```
class Solution:
    def dailyTemperatures(self, temp):
        st = []
        n = len(temp)
        res = [0] * n
        for i in range(n - 1, -1, -1):
            while st and temp[i] >= temp[st[-1]]:
                st.pop()
            res[i] = 0 if not st else st[-1] - i
            st.append(i)
        return res


if __name__ == "__main__":
    temps = [73, 74, 75, 71, 69, 72, 76, 73]
    print("Result:", Solution().dailyTemperatures(temps))
```

Complexity:

- Time: O(n)

- Space: O(n)

Output:

```
Result: [1, 1, 4, 2, 1, 1, 0, 0]
```

Q8. Find Median from Data Stream

Aim:

Design a structure to add numbers and find median dynamically.

Description:

Use two heaps: max-heap (`left`) for smaller half, min-heap (`right`) for larger half. Balance them.

Full Code:

```
import heapq


class MedianFinder:
    def __init__(self):
        self.left = []   # max-heap (store negatives)
        self.right = []  # min-heap

    def addNum(self, num):
        heapq.heappush(self.left, -num)
        if self.left and self.right and -self.left[0] >
self.right[0]:
            ele = -heapq.heappop(self.left)
            heapq.heappush(self.right, ele)
        if len(self.left) > len(self.right) + 1:
            ele = -heapq.heappop(self.left)
            heapq.heappush(self.right, ele)
        if len(self.right) > len(self.left) + 1:
            ele = heapq.heappop(self.right)
            heapq.heappush(self.left, -ele)

    def findMedian(self):
```

```
        if len(self.right) == len(self.left):
            return (-self.left[0] + self.right[0]) / 2
        elif len(self.left) > len(self.right):
            return -self.left[0]
        else:
            return self.right[0]


if __name__ == "__main__":
    mf = MedianFinder()
    mf.addNum(1)
    mf.addNum(2)
    print("Median:", mf.findMedian())
    mf.addNum(3)
    print("Median:", mf.findMedian())
```

Complexity:

- Time: O(log n) per insertion, O(1) for median
- Space: O(n)

Output:

```
Median: 1.5
Median: 2
```

Q9. Rotate Image (Matrix 90°)

Aim:
Rotate a matrix by 90° clockwise in-place.

Description:
First transpose the matrix, then reverse each row.

Full Code:

```
class Solution:
    def rotate(self, matrix):
        for i in range(len(matrix) - 1):
            for j in range(i + 1, len(matrix)):
                matrix[i][j], matrix[j][i] = matrix[j][i],
matrix[i][j]
        for row in matrix:
            row.reverse()


if __name__ == "__main__":
    mat = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]
    Solution().rotate(mat)
    for row in mat:
        print(row)
```

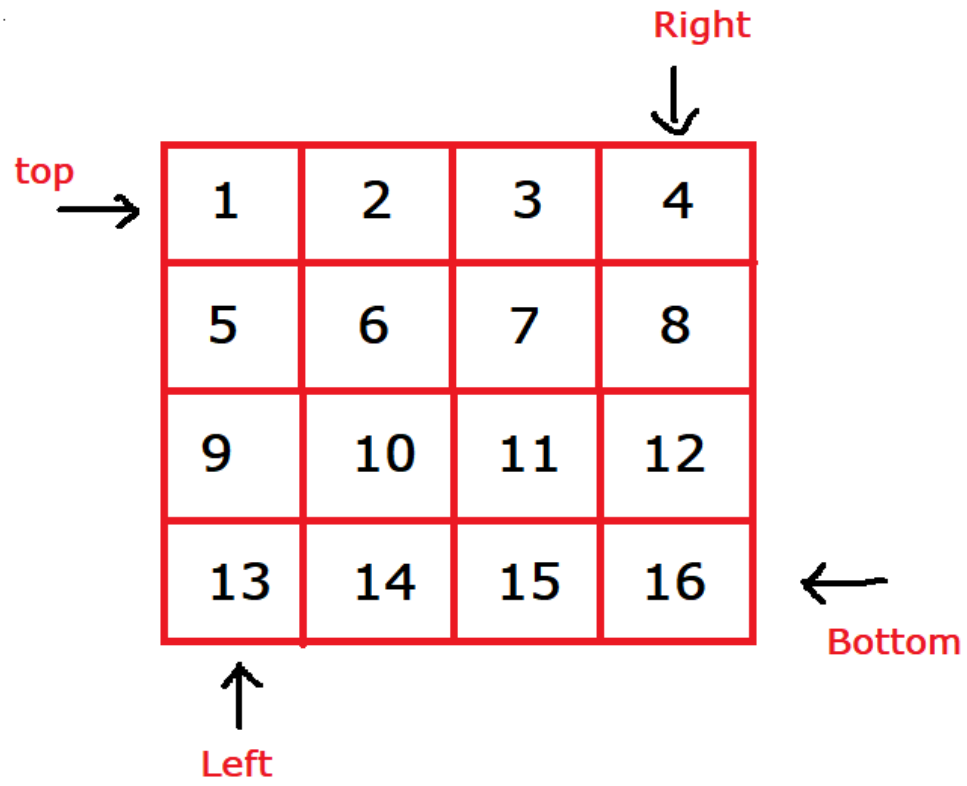
Complexity:

- Time: O(n²)
- Space: O(1)

Output:

```
[7, 4, 1]
[8, 5, 2]
[9, 6, 3]
```

Q10 — Spiral Matrix Traversal



Aim

Return matrix elements in spiral order.

Description

Traverse layer by layer using `top, bottom, left, right` boundaries.

Full Code

```
class Solution:
    def spiralOrder(self, matrix):
        if not matrix or not matrix[0]:
            return []
        n = len(matrix)
        m = len(matrix[0])
        top, bottom, left, right = 0, n - 1, 0, m - 1

        ans = []
        while left <= right and top <= bottom:
```

```
            # right
            for i in range(left, right + 1):
                ans.append(matrix[top][i])
            top += 1
            # down
            for i in range(top, bottom + 1):
                ans.append(matrix[i][right])
            right -= 1
            # left
            if top <= bottom:
                for i in range(right, left - 1, -1):
                    ans.append(matrix[bottom][i])
                bottom -= 1
            # up
            if left <= right:
                for i in range(bottom, top - 1, -1):
                    ans.append(matrix[i][left])
                left += 1
        return ans


if __name__ == "__main__":
    mat = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]
    print("Spiral Order:", Solution().spiralOrder(mat))
```

Complexity

- Time: O(N×M)

- Space: O(1) (output excluded)

Sample Output

```
Spiral Order: [1, 2, 3, 6, 9, 8, 7, 4, 5]
```

Q11 — Set Matrix Zeroes

Aim

If an element is 0, set its entire row and column to 0 (in-place).

Description

Use first row/column as markers; use `col0` to track first column.

Full Code

```
class Solution:
    def setZeroes(self, matrix):
        if not matrix or not matrix[0]:
            return
        n = len(matrix)
        m = len(matrix[0])
        col0 = 1

        # mark rows/cols
        for i in range(n):
            for j in range(m):
                if matrix[i][j] == 0:
                    matrix[i][0] = 0
                    if j != 0:
                        matrix[0][j] = 0
                    else:
                        col0 = 0

        # use marks to set zeroes (except first row/col)
        for i in range(1, n):
            for j in range(1, m):
                if matrix[i][0] == 0 or matrix[0][j] == 0:
                    matrix[i][j] = 0

        # first row
        if matrix[0][0] == 0:
```

```
            for j in range(m):
                matrix[0][j] = 0

        # first column
        if col0 == 0:
            for i in range(n):
                matrix[i][0] = 0


if __name__ == "__main__":
    mat = [
        [1, 1, 1],
        [1, 0, 1],
        [1, 1, 1]
    ]
    Solution().setZeroes(mat)
    print("After setZeroes:")
    for row in mat:
        print(row)
```

Complexity

- Time: O(N×M)
- Space: O(1)

Sample Output

```
After setZeroes:
[1, 0, 1]
[0, 0, 0]
[1, 0, 1]
```

Q12 — Valid Anagram

Aim

Check whether two strings are anagrams.

Description

Count frequency from `s`, decrement using `t`. If any mismatch → not anagram.

Full Code

```
class Solution:
    def isAnagram(self, s, t):
        if len(s) != len(t):
            return False
        freq = {}
        for ch in s:
            freq[ch] = freq.get(ch, 0) + 1
        for ch in t:
            if ch not in freq or freq[ch] == 0:
                return False
            freq[ch] -= 1
        return True


if __name__ == "__main__":
    print(Solution().isAnagram("anagram", "nagaram"))  # True
    print(Solution().isAnagram("rat", "car"))          # False
```

Complexity

- Time: O(N)
- Space: O(1) if alphabet bounded (else O(N))

Sample Output

```
True
```

```
False
```

Q13 — Longest Consecutive Sequence

Aim

Return length of the longest consecutive integer sequence in an array.

Description

Use a set for O(1) lookups; only start expanding when number is sequence start (`x-1` not in set).

Full Code

```
class Solution:
    def longestConsecutive(self, nums):
        if not nums:
            return 0
        mySet = set(nums)
        longest = 0
        for x in mySet:
            if x - 1 not in mySet:
                length = 1
                y = x
                while y + 1 in mySet:
                    y += 1
                    length += 1
                if length > longest:
                    longest = length
        return longest


if __name__ == "__main__":
    print(Solution().longestConsecutive([100, 4, 200, 1, 3, 2]))  # 4
    print(Solution().longestConsecutive([]))                      # 0
```

Complexity

- Time: O(N) average

- Space: O(N)

Sample Output

```
4
0
```

Q14 — Longest Common Prefix

Aim

Find the longest common prefix among a list of strings.

Description

Take the shortest string and compare characters across all strings, shrinking prefix on mismatch.

Full Code

```
class Solution:
    def longestCommonPrefix(self, strs):
        if not strs:
            return ""
        # pick the shortest string as initial candidate
        word = min(strs, key=len)
        for s in strs:
            i = 0
            # compare until mismatch or end of candidate
            while i < len(word) and i < len(s) and word[i] == s[i]:
                i += 1
            word = word[:i]
            if not word:
                return ""
        return word


if __name__ == "__main__":
    print(Solution().longestCommonPrefix(["flower", "flow",
"flight"]))  # "fl"
    print(Solution().longestCommonPrefix(["dog", "racecar", "car"]))
# ""
    print(Solution().longestCommonPrefix([]))
# ""
```

Complexity

- Time: O(N × M) where M = length of shortest string
- Space: O(1)

Sample Output

```
fl

```

(The second line is an empty string printed as a blank line.)

Q15 — GCD of Min and Max

Aim

Compute GCD of the smallest and largest numbers in an array.

Description

Find `min` and `max`, then use Euclidean algorithm for GCD.

Full Code

```
class Solution:
    def findGCD(self, nums):
        a = min(nums)
        b = max(nums)
        # Euclidean algorithm
        while b != 0:
            a, b = b, a % b
        return a


if __name__ == "__main__":
    print(Solution().findGCD([2, 5, 6, 9, 10]))  # 2
    print(Solution().findGCD([3, 3]))            # 3
```

Complexity

- Time: O(log(min, max)) per GCD steps
- Space: O(1)

Sample Output

```
2
3
```

Q16 — Find Intersection Values (Counts)

Aim

Count how many elements from `nums1` appear in `nums2` (and vice versa).

Description

Make sets for constant-time membership checks; iterate each list counting occurrences in the other.

Full Code

```
class Solution:
    def findIntersectionValues(self, nums1, nums2):
        set1 = set(nums1)
        set2 = set(nums2)
        c1 = sum(1 for x in nums1 if x in set2)
        c2 = sum(1 for x in nums2 if x in set1)
        return [c1, c2]



if __name__ == "__main__":
    print(Solution().findIntersectionValues([4,3,2,3,1],
[2,2,5,2,3,6]))  # [3,2]
    print(Solution().findIntersectionValues([], [1,2]))
# [0,0]
```

Complexity

- Time: O(N + M)
- Space: O(N + M)

Sample Output

```
[3, 2]
[0, 0]
```
