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 `1`s 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 Matrix

Aim

Rotate a square matrix by 90 degrees clockwise in-place.

Description

We first **transpose the matrix** (swap `matrix[i][j]` with `matrix[j][i]` for all `i < j`) and then **reverse each row**. This gives the rotated matrix without using extra space.

Full Code

```
from typing import List

class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        for i in range(0, len(matrix) - 1):
            for j in range(i + 1, len(matrix)):
                matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

        for i in range(0, len(matrix)):
            matrix[i].reverse()

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

Time Complexity

- **O(n^2)** (transpose + reverse each row)

Space Complexity

- **O(1)** (in-place)

Sample Output

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

Q10 → Spiral Order

Aim

Return all elements of a matrix in spiral order.

Description

We use four boundaries (`top`, `bottom`, `left`, `right`) and iterate layer by layer in spiral order until all elements are traversed.

Full Code

```
from typing import List

class Solution:
    def spiralOrder(self, matrix: List[List[int]])
-> List[int]:
        n, m = len(matrix), len(matrix[0])
        top, bottom, left, right = 0, n - 1, 0, m -
1
        ans = []

        while left <= right and top <= bottom:
            for i in range(left, right + 1):
                ans.append(matrix[top][i])
            top += 1

            for i in range(top, bottom + 1):
                ans.append(matrix[i][right])
            right -= 1

            if top <= bottom:
                for i in range(right, left - 1,
-1):
                    ans.append(matrix[bottom][i])
                bottom -= 1

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

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

Time Complexity

- **O(n·m)** (each element visited once)

Space Complexity

- **O(1)** (excluding output list)

Sample Output

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

Q11 → Set Matrix Zeroes

Aim

Modify a matrix in-place such that if any element is `0`, its entire row and column become `0`.

Description

We use the **first row and first column as markers** to indicate which rows/columns need to be zeroed, and a flag (`col0`) to track the first column separately.

Full Code

```
from typing import List

class Solution:
    def setZeroes(self, matrix: List[List[int]]) ->
None:
        n, m = len(matrix), len(matrix[0])
        col0 = 1

        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

        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

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

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

if __name__ == "__main__":
    matrix = [[1,1,1],[1,0,1],[1,1,1]]
    Solution().setZeroes(matrix)
    print(matrix)
```

Time Complexity

- **O(n·m)**

Space Complexity

- **O(1)**

Sample Output

```
[[1,0,1],[0,0,0],[1,0,1]]
```

Q12 → Valid Anagram

Aim

Check if two strings are anagrams.

Description

We count the frequency of characters in `s` and then decrement counts using `t`. If counts match, they are anagrams.

Full Code

```
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        if len(s) != len(t):
            return False

        freq = {}
        for i in s:
            freq[i] = freq.get(i, 0) + 1

        for i in t:
            if i not in freq or freq[i] == 0:
                return False
            freq[i] -= 1

        return True

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

Time Complexity

- **O(n)**

Space Complexity

- **O(1)** (since only lowercase letters)

Sample Output

```
True
False
```

Q13 → Longest Consecutive Sequence

Aim

Find the length of the longest consecutive sequence in an array.

Description

We use a set for O(1) lookups. For each element, if it’s the start of a sequence, extend forward to count length.

Full Code

```
from typing import List


class Solution:
    def longestConsecutive(self, nums: List[int])
-> int:
        if not nums:
            return 0
        longest = 1
        mySet = set(nums)

        for i in mySet:
            if (i - 1) in mySet:
                continue
            cnt, x = 1, i
            while (x + 1) in mySet:
                x += 1
                cnt += 1
            longest = max(cnt, longest)

        return longest

if __name__ == "__main__":

print(Solution().longestConsecutive([100,4,200,1,3,
2]))
```

Time Complexity

- **O(n)** (on average with hashing)

Space Complexity

- **O(n)**

Sample Output

```
4
```

Q14 → Longest Common Prefix

Aim

Find the longest common prefix among an array of strings.

Description

We first select the shortest string, then compare prefixes with each other string and shrink as necessary.

Full Code

```python
from typing import List

class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        mini = 1e9
        word = ""

        for i in strs:
            if mini > len(i):
                mini = len(i)
                word = i

        for n in strs:
            for i in range(len(word)):
                if i == 0 and word[0] != n[0]:
                    return ""
                if word[i] != n[i]:
                    word = word[:i]
                    break

        return word

if __name__ == "__main__":

print(Solution().longestCommonPrefix(["flower","flow","flight"]))

print(Solution().longestCommonPrefix(["dog","racecar","car"]))
```

Time Complexity

- **O(n·m)** (n = no. of strings, m = min length string)

Space Complexity

- **O(1)**

Sample Output

```
fl
""
```

Q15 → GCD of Min & Max

Aim

Find the GCD of the minimum and maximum element of an array.

Description

We use the **Euclidean algorithm** for GCD on the smallest and largest numbers in the list.

Full Code

```
from typing import List

class Solution:
    def findGCD(self, nums: List[int]) -> int:
        a = min(nums)
        b = max(nums)

        def myGCD(a, b):
            while b != 0:
                a, b = b, a % b
            return a

        return myGCD(a, b)

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

Time Complexity

- **O(log(min(a, b)))**

Space Complexity

- **O(1)**

Sample Output

```
2
```

Q16 → Find Intersection Values

Aim

Find how many elements of one array appear in another and return counts for both arrays.

Description

We convert arrays to sets for O(1) membership check and count occurrences.

Full Code

```
from typing import List


class Solution:
    def findIntersectionValues(self, nums1: List[int], nums2: List[int]) -> List[int]:
        set1, set2 = set(nums1), set(nums2)
        c1 = sum(1 for i in nums1 if i in set2)
        c2 = sum(1 for i in nums2 if i in set1)
        return [c1, c2]

if __name__ == "__main__":

print(Solution().findIntersectionValues([4,3,2,3,1],[2,2,5,2,3,6]))
```

Time Complexity

- **O(n + m)**

Space Complexity

- **O(n + m)**

Sample Output

```
[3, 2]
```