Question 1

```
# Version A
def myFunc(head):
    prev = None
    while head:
        next_node = head.next
        head.next = prev
        prev = head
        head = next_node
    return prev
```

```
# Version B
def myFunc(head):
    if not head or not head.next:
        return head
    rest = reverseList(head.next)
    head.next.next = head
    head.next = None
    return rest
```

> - Which version would you deploy to production and why?
> - What’s the stack behavior in the recursive version?
> - How would each version behave on a list with 1M elements?

Question 2

```
# Version A
def myFunc(n):
    if n < 2:
        return n
    return fib_A(n-1) + fib_A(n-2)
```

```
# Version B
def myFunc(n, memo=None):
    if memo is None:
        memo = {}
    if n < 2:
        return n
    if n in memo:
        return memo[n]
    memo[n] = fib_B(n-1, memo) + fib_B(n-2, memo)
    return memo[n]
```

```
# Version C
def myFunc(n):
    if n < 2:
        return n
    a, b = 0, 1
    for _ in range(2, n+1):
        a, b = b, a + b
    return b
```

> **Follow-up Questions:**
>
> - How do these compare for large values of `n`?
> - What are the stack implications of recursion?

Question 3

Version A

```
def myFunc(s, t):
    return sorted(s) == sorted(t)
```

- Explain Logic: What is happening here
- Time complexity and Space complexity

Version B

```
from collections import Counter

def myFunc(s, t):
    return Counter(s) == Counter(t)
```

- Explain Logic: What is happening here
- Time complexity and Space complexity

Version C

```python
def myFunc(s, t):
    if len(s) != len(t):
        return False
    count = [0] * 26
    for c1, c2 in zip(s, t):
        count[ord(c1) - ord('a')] += 1
        count[ord(c2) - ord('a')] -= 1
    return all(x == 0 for x in count)
```

- Explain Logic: What is happening here
- Time complexity and Space complexity

Which of these is best for production use-case?

Question 4

Version A

```python
def moveZeroes(nums):
    nums.sort(key=lambda x: x == 0)
```

- Explain Logic: What is happening here
- Time complexity and Space complexity

Version B

```python
def moveZeroes(nums):
    non_zero = [x for x in nums if x != 0]
    nums[:] = non_zero + [0] * (len(nums) - len(non_zero))
```

- Explain Logic: What is happening here
- Time complexity and Space complexity

Version C

```python
def moveZeroes(nums):
    insert_pos = 0
    for i in range(len(nums)):
        if nums[i] != 0:
            nums[insert_pos] = nums[i]
            insert_pos += 1
    for i in range(insert_pos, len(nums)):
        nums[i] = 0
```

- Explain Logic: What is happening here
- Time complexity and Space complexity

Which of these is best for production use-case?

Question 5

Version A

```
def myFunc(l1, l2):
    if not l1 or not l2:
        return l1 or l2
    if l1.val < l2.val:
        l1.next = merge(l1.next, l2)
        return l1
    else:
        l2.next = merge(l1, l2.next)
        return l2
```

- Explain Logic: What is happening here
- Time complexity and Space complexity

Version B

```
def myFunc(l1, l2):
    dummy = ListNode()
    current = dummy
    while l1 and l2:
        if l1.val < l2.val:
            current.next = l1
            l1 = l1.next
        else:
            current.next = l2
            l2 = l2.next
        current = current.next
    current.next = l1 or l2
    return dummy.next
```

- Explain Logic: What is happening here
- Time complexity and Space complexity

Which of these is best for production use-case?