Practice Exercises:

Question 1 (Rotate a Singly Linked List)

Write a function rotateList(head, k) that rotates a **singly linked list** to the right by k positions. The rotation should move the last k elements to the front of the list.

For example, given the following singly linked list:

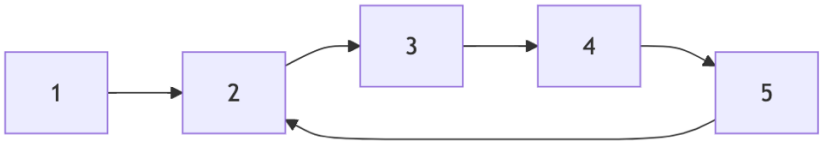
If k = 2, after rotation, the list becomes:

- Provide the pseudocode for rotateList(head, k).

Explain how you handle cases where k is larger than the length of the list.

Question 2 (Cycle Detection in a Singly Linked List)

Given a **singly linked list**, write a function `detectCycle(head)` that returns true if there is a cycle in the list and false otherwise. A cycle occurs when a node's next pointer points to a previous node in the list instead of null.



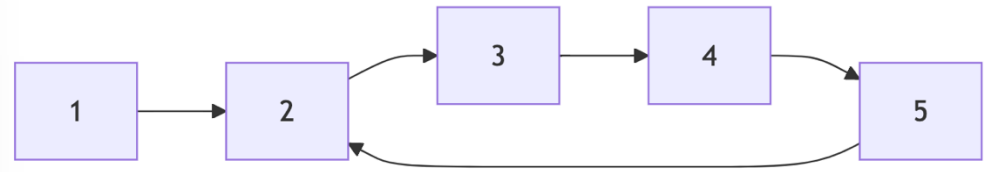
Consider the following linked list with a cycle:

- Describe how you would modify a standard traversal algorithm to detect the cycle.
- Provide the pseudocode for detectCycle(head).

Question 3 (Detect and Remove a Cycle in a Singly Linked List)

Write a function removeCycle(head) that detects and removes a cycle from a **singly linked list**. Assume that if there is a cycle, the last node in the cycle points back to one of the previous nodes in the list.

For example, consider the following linked list with a cycle:



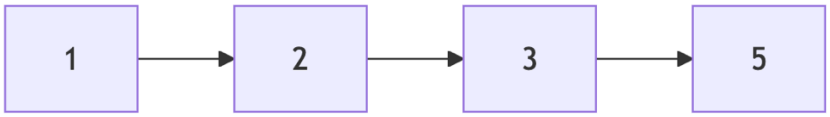
The function should modify the list to remove the cycle:

- Provide the pseudocode for removeCycle(head).
- Describe how to modify the last node's next pointer to break the cycle.

Question 4 (insert a Node at a Specific Position in a Singly Linked List)

Write a function insertAtPosition(head, newNode, position) that inserts a node at a specific position in a **singly linked list**. If the position is invalid (e.g., larger than the length of the list), the function should return false. Otherwise, it should return true.

For example, given the following singly linked list:



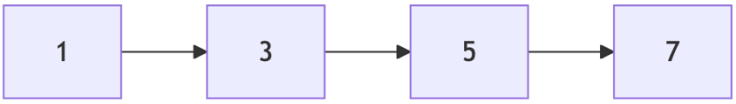
If you insert a node with the value 4 at position 4, the list becomes:

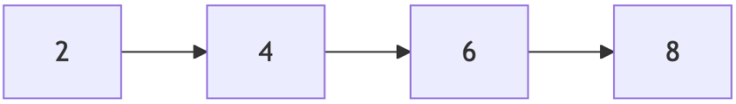
- Provide the pseudocode for insertAtPosition(head, newNode, position).
- Explain the edge cases, such as inserting at the list's front or end.

Question 5 (Merge Two Sorted Singly Linked Lists)

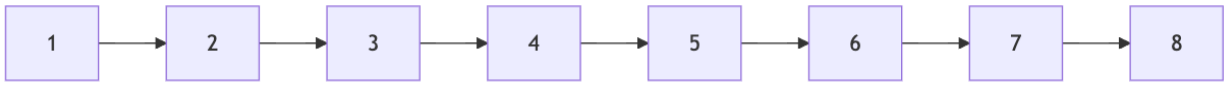
Given two **sorted singly linked lists**, write a function mergeSortedLists(list1, list2) that merges them into one sorted singly linked list.

For example, given the following two linked lists:



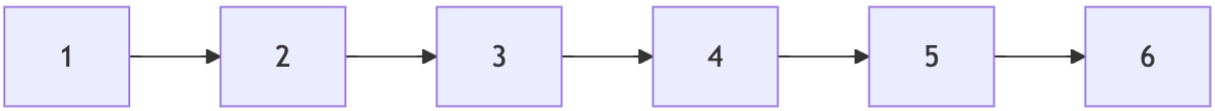


After merging, the result should be:



- Provide the pseudocode for mergeSortedLists(list1, list2).
- Describe how you handle cases where one list is exhausted before the other.

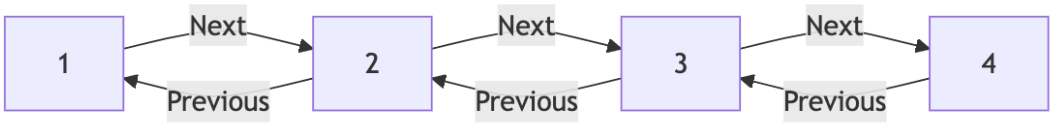
Question 6 (Find the Middle Node in a Singly Linked List)
Write a function findMiddleNode(head) that returns the middle node of a **singly linked list**. If the list has an even number of nodes, return the second of the two middle nodes.
For example, given the following singly linked list:



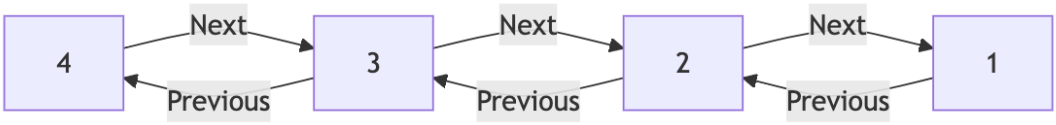
The function should return the node with value 4 (the second middle node).

- Provide the pseudocode for findMiddleNode(head).
- Explain how you would handle cases where the list is empty or has only one node.

Question 7 (Reverse a Doubly Linked List)
Given a **doubly linked list**, write a function `reverseDLL(front)` that reverses the list so that the last node becomes the first node, and the first node becomes the last. After reversal, each node's next pointer should point to the previous node, and the previous pointer should point to the next node.
Consider the following doubly linked list:



After the reversal, it should become:

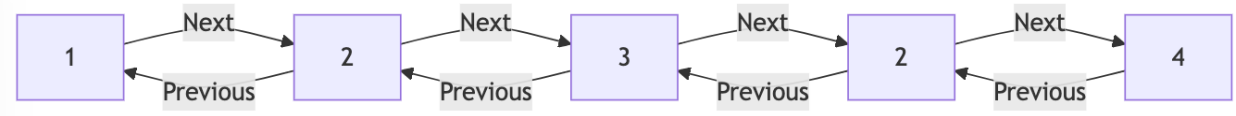


Provide the pseudocode for `reverseDLL(front)`.
Explain how the reversal affects both the next and the previous pointers.

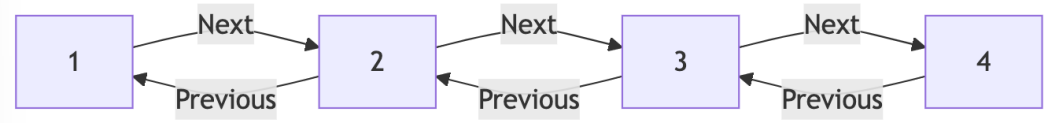
Question 8 (Remove Duplicates from a Doubly Linked List)

Write a function removeDuplicatesDLL(front) that removes all duplicate nodes from a **doubly linked list**. Assume the list is not sorted.

For example, given the following doubly linked list:



After removing duplicates, it should become:



- Provide the pseudocode for removeDuplicatesDLL(front).
- Discuss how the doubly linked list structure (with next and previous pointers) simplifies the removal of duplicates.

Appendix A – Solutions

Cycle Detection in a Singly Linked List

Solution:

- Maintain two-pointers, slow and fast. Move slow one step at a time and fast two steps at a time.
- If slow and fast meet, a cycle is detected. If fast reaches null, there is no cycle.

Pseudocode:

```
function detectCycle(head):
  slow = head
  fast = head

  while fast ≠ null and fast.next ≠ null do:
    slow = slow.next
    fast = fast.next.next

    if slow == fast:
      return true  # Cycle detected

  return false  # No cycle
```

Reverse a Doubly Linked List

Solution:

- Traverse the list from the front and for each node, swap its next and prev pointers.
- At the end, swap the front and tail pointers.

Pseudocode:

```
function reverseDLL(front):
  current = front
  previous = null

  while current ≠ null do:
    temp = current.next
    current.next = current.prev
    current.prev = temp
    previous = current
    current = temp
```

```
  return previous  # New front (was tail before reversal)
```

Insert a Node at a Specific Position in a Singly Linked List

Solution:

- If the position is 1, insert the node at the front.
- Otherwise, traverse the list until the predecessor node of the target position and insert the new node there.

Pseudocode:

pseudo

Copy code

```
function insertAtPosition(head, newNode, position):
  if position == 1:
    newNode.next = head
    return newNode  # New head

  current = head
  for i = 1 to position - 1 do:
    if current == null:
      return false  # Invalid position

    current = current.next

  newNode.next = current.next
  current.next = newNode

  return true
```

Merge Two Sorted Singly Linked Lists

Solution:

- Use a two-pointer technique to traverse both lists, adding the smaller node to the merged list at each step.
- Continue until both lists are fully merged.

Pseudocode:

pseudo

Copy code

```
function mergeSortedLists(list1, list2):
  if list1 == null:
    return list2
```

```
    if list2 == null:
        return list1

    if list1.data < list2.data:
        result = list1
        result.next = mergeSortedLists(list1.next, list2)
    else:
        result = list2
        result.next = mergeSortedLists(list1, list2.next)

    return result
```

Find the Middle Node in a Singly Linked List

Solution:

- Use two pointers: slow and fast. Move slow one step at a time and fast two steps. When fast reaches the end, slow will be at the middle.

Pseudocode:

pseudo

Copy code

```
function findMiddleNode(head):
    slow = head
    fast = head

    while fast ≠ null and fast.next ≠ null do:
        slow = slow.next
        fast = fast.next.next

    return slow  # Middle node
```

Remove Duplicates from a Doubly Linked List

Solution:

- Use a nested loop to compare each node with all subsequent nodes. If a duplicate is found, remove it by adjusting the next and previous pointers of the adjacent nodes.

Pseudocode:

pseudo

Copy code

```
function removeDuplicatesDLL(front):
    current = front
```

```
  while current ≠ null do:
    runner = current.next

    while runner ≠ null do:
      if runner.data == current.data:
        # Remove the runner node
        if runner.next ≠ null:
          runner.next.prev = runner.prev
        if runner.prev ≠ null:
          runner.prev.next = runner.next
      runner = runner.next

    current = current.next
```

---

7. Rotate a Singly Linked List

Solution:

- First, find the length of the list.
- Move the last k nodes to the front by finding the node that will become the new head.

Pseudocode:

```
function rotateList(head, k):
  if head == null or k == 0:
    return head

  # Step 1: Find the length of the list
  length = 1
  current = head
  while current.next ≠ null do:
    current = current.next
    length = length + 1

  # Step 2: Adjust k (if k > length)
  k = k % length
  if k == 0:
    return head  # No rotation needed

  # Step 3: Find the new head and tail
  newTail = head
```

```
for i = 1 to length - k - 1 do:
    newTail = newTail.next

newHead = newTail.next
newTail.next = null
current.next = head  # Make the last node point to the original head

return newHead
```

Detect and Remove a Cycle in a Singly Linked List

Solution:

- First, use Floyd's Cycle-Finding Algorithm to detect the cycle.
- Then, find the node where the cycle starts and remove it by setting the next pointer of the last node in the cycle to null.

Pseudocode:

pseudo

Copy code

```
function removeCycle(head):
    slow = head
    fast = head

    # Step 1: Detect cycle
    while fast ≠ null and fast.next ≠ null do:
        slow = slow.next
        fast = fast.next.next

        if slow == fast:
            break  # Cycle detected

    if fast == null or fast.next == null:
        return  # No cycle

    # Step 2: Find the start of the cycle
    slow = head
    while slow ≠ fast do:
        slow = slow.next
        fast = fast.next
```

```
# Step 3: Remove the cycle
prev = null
while fast.next ≠ slow do:
   fast = fast.next
fast.next = null  # Break the cycle
```