Problem 1: Reverse a singly linked list.

Input: 1 -> 2 -> 3 -> 4 -> 5 Output: 5 -> 4 -> 3 -> 2 -> 1

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

def reverse_linked_list(head):
    prev = None
    current = head

    while current:
        next_node = current.next  # Store next node
        current.next = prev       # Reverse link
        prev = current            # Move prev forward
        current = next_node       # Move current forward

    return prev  # New head of reversed list

# Helper function to print the linked list
def print_list(head):
    while head:
        print(head.val, end=" -> " if head.next else "")
        head = head.next
    print()

# Example usage
head = ListNode(1, ListNode(2, ListNode(3, ListNode(4, ListNode(5)))))
print("Original list:")
print_list(head)

reversed_head = reverse_linked_list(head)
print("Reversed list:")
print_list(reversed_head)
```

```
Original list:
1 -> 2 -> 3 -> 4 -> 5
Reversed list:
5 -> 4 -> 3 -> 2 -> 1
```

Problem 2: Merge two sorted linked lists into one sorted linked list.

Input: List 1: 1 -> 3 -> 5, List 2: 2 -> 4 -> 6 Output: 1 -> 2 -> 3 -> 4 -> 5 -> 6

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

```python
def merge_sorted_lists(l1, l2):
    dummy = ListNode()  # Dummy node to start the merged list
    current = dummy  # Pointer to build the list

    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  # Move the pointer forward

    # Append remaining nodes (if any)
    current.next = l1 if l1 else l2

    return dummy.next  # Return merged list (skip dummy node)

# Helper function to print linked list
def print_list(head):
    while head:
        print(head.val, end=" -> " if head.next else "")
        head = head.next
    print()

# Example usage
l1 = ListNode(1, ListNode(3, ListNode(5)))
l2 = ListNode(2, ListNode(4, ListNode(6)))

print("List 1:")
print_list(l1)

print("List 2:")
print_list(l2)

merged_head = merge_sorted_lists(l1, l2)
print("Merged List:")
print_list(merged_head)
```

```
List 1:
1 -> 3 -> 5
List 2:
2 -> 4 -> 6
Merged List:
1 -> 2 -> 3 -> 4 -> 5 -> 6
```

Problem 3: Remove the nth node from the end of a linked list.

Input: 1 -> 2 -> 3 -> 4 -> 5, n = 2 Output: 1 -> 2 -> 3 -> 5

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

def remove_nth_from_end(head, n):
    dummy = ListNode(0, head)  # Dummy node before head to handle edge cases
    fast = slow = dummy

    # Move fast n steps ahead
    for _ in range(n):
        fast = fast.next

    # Move both pointers until fast reaches the end
    while fast.next:
        fast = fast.next
        slow = slow.next

    # Remove nth node
    slow.next = slow.next.next

    return dummy.next  # Return new head (skip dummy node)

# Helper function to print linked list
def print_list(head):
    while head:
        print(head.val, end=" -> " if head.next else "")
        head = head.next
    print()

# Example usage
head = ListNode(1, ListNode(2, ListNode(3, ListNode(4, ListNode(5)))))
n = 2

print("Original List:")
print_list(head)

new_head = remove_nth_from_end(head, n)
print("List after removing {}th node from end:".format(n))
print_list(new_head)
```

```
Original List:
1 -> 2 -> 3 -> 4 -> 5
List after removing 2th node from end:
1 -> 2 -> 3 -> 5
```

Problem 4: Find the intersection point of two linked lists.

Input: List 1: 1 -> 2 -> 3 -> 4, List 2: 9 -> 8 -> 3 -> 4 Output: Node with value 3

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

def get_intersection_node(headA, headB):
    if not headA or not headB:
        return None

    pointerA, pointerB = headA, headB

    while pointerA != pointerB:
        # Move pointers, switching lists when reaching the end
        pointerA = pointerA.next if pointerA else headB
        pointerB = pointerB.next if pointerB else headA

    return pointerA  # Either the intersection node or None if no intersection

# Helper function to print the intersection node
def print_intersection(node):
    if node:
        print(f"Intersection at node with value: {node.val}")
    else:
        print("No intersection found.")

# Example usage
# Creating an intersection manually
common = ListNode(3, ListNode(4))  # Common part

list1 = ListNode(1, ListNode(2, common))  # 1 -> 2 -> 3 -> 4
list2 = ListNode(9, ListNode(8, common))  # 9 -> 8 -> 3 -> 4

print_intersection(get_intersection_node(list1, list2))
```

```
Intersection at node with value: 3
```

Problem 5: Remove duplicates from a sorted linked list.

Input: 1 -> 1 -> 2 -> 3 -> 3 Output: 1 -> 2 -> 3

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

def remove_duplicates(head):
    current = head

    while current and current.next:
        if current.val == current.next.val:
            current.next = current.next.next  # Skip the duplicate node
        else:
            current = current.next  # Move to the next node if no duplicate

    return head  # Return the modified list

# Helper function to print linked list
def print_list(head):
    while head:
        print(head.val, end=" -> " if head.next else "")
        head = head.next
    print()
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def remove_duplicates(head):
    current = head

    while current and current.next:
        if current.val == current.next.val:
            current.next = current.next.next  # Skip the duplicate node
        else:
            current = current.next  # Move to the next node if no duplicate

    return head  # Return the modified list

# Helper function to print linked list
def print_list(head):
    while head:
        print(head.val, end=" -> " if head.next else "")
        head = head.next
    print()

# Example usage
head = ListNode(1, ListNode(1, ListNode(2, ListNode(3, ListNode(3)))))
print("Original List:")
print_list(head)

new_head = remove_duplicates(head)
print("List after removing duplicates:")
print_list(new_head)

new_head = remove_duplicates(head)
print("List after removing duplicates:")
print_list(new_head)
```

```
Original List:
1 -> 1 -> 2 -> 3 -> 3
List after removing duplicates:
1 -> 2 -> 3
List after removing duplicates:
1 -> 2 -> 3
```

Problem 6: Add two numbers represented by linked lists (where each node contains a single digit).

Input: List 1: 2 -> 4 -> 3, List 2: 5 -> 6 -> 4 (represents 342 + 465) Output: 7 -> 0 -> 8 (represents 807)

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

def add_two_numbers(l1, l2):
    dummy = ListNode()  # Dummy node to simplify list operations
    current = dummy
    carry = 0  # Carry for addition

    while l1 or l2 or carry:
        val1 = l1.val if l1 else 0
        val2 = l2.val if l2 else 0
        total = val1 + val2 + carry  # Sum of digits and carry

        carry = total // 10  # Carry for next iteration
        current.next = ListNode(total % 10)  # Store last digit in new node
        current = current.next  # Move to next node

        # Move forward in lists if not at the end
        l1 = l1.next if l1 else None
        l2 = l2.next if l2 else None

    return dummy.next  # Return the new list, skipping the dummy node

# Helper function to print linked list
def print_list(head):
    while head:
        print(head.val, end=" -> " if head.next else "")
        head = head.next
    print()

# Example usage
l1 = ListNode(2, ListNode(4, ListNode(3)))  # Represents 342
l2 = ListNode(5, ListNode(6, ListNode(4)))  # Represents 465

print("List 1:")
print_list(l1)

print("List 2:")
print_list(l2)

result_head = add_two_numbers(l1, l2)
print("Sum List:")
print_list(result_head)
```

```
List 1:
2 -> 4 -> 3
List 2:
5 -> 6 -> 4
Sum List:
7 -> 0 -> 8
```

Problem 7: Swap nodes in pairs in a linked list.

Input: 1 -> 2 -> 3 -> 4 Output: 2 -> 1 -> 4 -> 3

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

def swap_pairs(head):
    dummy = ListNode(0, head)  # Dummy node to handle edge cases
    prev = dummy

    while prev.next and prev.next.next:
        first = prev.next
        second = first.next

        # Swap nodes
        first.next = second.next
        second.next = first
        prev.next = second

        # Move prev forward by two nodes
        prev = first

    return dummy.next  # Return new head (skip dummy)

# Helper function to print linked list
def print_list(head):
    while head:
        print(head.val, end=" -> " if head.next else "")
        head = head.next
    print()

# Example usage
head = ListNode(1, ListNode(2, ListNode(3, ListNode(4))))
print("Original List:")
print_list(head)

swapped_head = swap_pairs(head)
print("List after swapping pairs:")
print_list(swapped_head)
```

```
Original List:
1 -> 2 -> 3 -> 4
List after swapping pairs:
2 -> 1 -> 4 -> 3
```

Problem 8: Reverse nodes in a linked list in groups of k.

Input: 1 -> 2 -> 3 -> 4 -> 5, k = 3 Output: 3 -> 2 -> 1 -> 4 -> 5

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

def reverse_k_group(head, k):
    dummy = ListNode(0, head)  # Dummy node before head
    prev_group = dummy

    while True:
        kth = prev_group
        count = 0

        # Find the kth node from prev_group
        for _ in range(k):
            kth = kth.next
            if not kth:
                return dummy.next  # If fewer than k nodes remain, return result

        group_next = kth.next  # Node after the k-group

        # Reverse k nodes
        prev, curr = kth.next, prev_group.next
        for _ in range(k):
            next_node = curr.next
            curr.next = prev
            prev = curr
            curr = next_node

        # Connect reversed group with the previous part
        temp = prev_group.next  # First node of the group before reversal
        prev_group.next = kth  # Link prev_group to the new head of this group
        prev_group = temp  # Move prev_group forward for the next group

    return dummy.next  # Return new head (skip dummy)

# Helper function to print linked list
def print_list(head):
    while head:
        print(head.val, end=" -> " if head.next else "")
        head = head.next
    print()

# Example usage
head = ListNode(1, ListNode(2, ListNode(3, ListNode(4, ListNode(5)))))
k = 3

print("Original List:")
print_list(head)

reversed_head = reverse_k_group(head, k)
print("List after reversing in groups of {}:".format(k))
print_list(reversed_head)
```

```
Original List:
1 -> 2 -> 3 -> 4 -> 5
List after reversing in groups of 3:
3 -> 2 -> 1 -> 4 -> 5
```

Problem 9: Determine if a linked list is a palindrome.

Input: 1 -> 2 -> 2 -> 1 Output: True

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

def is_palindrome(head):
    if not head or not head.next:
        return True  # Single node or empty list is always a palindrome

    # Step 1: Find the middle (slow/fast pointer)
    slow, fast = head, head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next

    # Step 2: Reverse the second half of the list
    prev, curr = None, slow
    while curr:
        next_node = curr.next
        curr.next = prev
        prev = curr
        curr = next_node

    # Step 3: Compare both halves
    first_half, second_half = head, prev
    while second_half:  # Compare until second_half ends
        if first_half.val != second_half.val:
            return False
        first_half = first_half.next
        second_half = second_half.next

    return True  # If all nodes matched, it's a palindrome

# Helper function to create a linked list from a list
def create_list(values):
    head = ListNode(values[0])
    current = head
    for val in values[1:]:
        current.next = ListNode(val)
        current = current.next
    return head

# Example usage
head = create_list([1, 2, 2, 1])
```

```
print("Is the linked list a palindrome?", is_palindrome(head))  # Output: True
```

```
Is the linked list a palindrome? True
```

Problem 10: Rotate a linked list to the right by k places.

Input: 1 -> 2 -> 3 -> 4 -> 5, k = 2 Output: 4 -> 5 -> 1 -> 2 -> 3

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

def rotate_right(head, k):
    if not head or not head.next or k == 0:
        return head  # No rotation needed

    # Step 1: Find length of linked list
    length = 1  # Start from 1 because we already have head
    tail = head
    while tail.next:
        tail = tail.next
        length += 1

    # Step 2: Calculate effective rotation
    k = k % length
    if k == 0:
        return head  # No change needed

    # Step 3: Make the list circular
    tail.next = head

    # Step 4: Find the new tail (n - k steps from the head)
    new_tail = head
    for _ in range(length - k - 1):  # Move (length - k - 1) steps
        new_tail = new_tail.next

    # Step 5: Break the circular link and set new head
    new_head = new_tail.next
    new_tail.next = None  # Break cycle

    return new_head  # Return new head

# Helper function to create a linked list from a list
def create_list(values):
    head = ListNode(values[0])
    current = head
    for val in values[1:]:
        current.next = ListNode(val)
        current = current.next
    return head

# Helper function to print linked list
```

```
def print_list(head):
    while head:
        print(head.val, end=" -> " if head.next else "")
        head = head.next
    print()

# Example usage
head = create_list([1, 2, 3, 4, 5])
k = 2

print("Original List:")
print_list(head)

rotated_head = rotate_right(head, k)
print("List after rotating by {}:".format(k))
print_list(rotated_head)
```

```
Original List:
1 -> 2 -> 3 -> 4 -> 5
List after rotating by 2:
4 -> 5 -> 1 -> 2 -> 3
```

Problem 11: Flatten a multilevel doubly linked list.

Input: 1 <-> 2 <-> 3 <-> 7 <-> 8 <-> 11 -> 12, 4 <-> 5 -> 9 -> 10, 6 -> 13 Output: 1 <-> 2 <-> 3 <-> 4 <-> 5 <-> 6 <-> 7 <-> 8 <-> 9 <-> 10 <-> 11 <-> 12 <-> 13

```
class Node:
    def __init__(self, val=0, prev=None, next=None, child=None):
        self.val = val
        self.prev = prev
        self.next = next
        self.child = child

def flatten(head):
    if not head:
        return None

    stack = []
    current = head

    while current:
        if current.child:
            # If there's a next node, push it onto the stack
            if current.next:
                stack.append(current.next)

            # Connect child as the next node
            current.next = current.child
            current.next.prev = current
            current.child = None  # Remove child reference
        # If no next node and stack is not empty, pop a node from stack
        if not current.next and stack:
            current.next = stack.pop()
            current.next.prev = current

        current = current.next  # Move forward

    return head

# Helper function to print the flattened list
def print_list(head):
    while head:
        print(head.val, end=" <-> " if head.next else "")
        head = head.next
    print()

# Example usage: Creating a multilevel linked list
head = Node(1)
node2 = Node(2, head)
node3 = Node(3, node2)
node4 = Node(4, node3)
node5 = Node(5, node4)
node6 = Node(6, node5)

head.next = node2
node2.next = node3
node3.next = node4
node4.next = node5
node5.next = node6

node7 = Node(7)
node3.child = node7
node8 = Node(8, node7)
node7.next = node8
node9 = Node(9)
node10 = Node(10, node9)
node9.next = node10
node5.child = node9

print("Flattened List:")
flattened_head = flatten(head)
print_list(flattened_head)
```

```
Flattened List:
1 <-> 2 <-> 3 <-> 7 <-> 8 <-> 4 <-> 5 <-> 9 <-> 10 <-> 6
```

Problem 12: Rearrange a linked list such that all even positioned nodes are placed at the end.

Input: 1 -> 2 -> 3 -> 4 -> 5 Output: 1 -> 3 -> 5 -> 2 -> 4

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

def rearrange_even_nodes(head):
    if not head or not head.next:
        return head  # No rearrangement needed for empty or single-node list

    odd = head  # First node (odd-positioned)
    even = head.next  # Second node (even-positioned)
    even_head = even  # Keep track of even list start

    while even and even.next:
        odd.next = even.next  # Connect odd nodes
        odd = odd.next
        even.next = odd.next  # Connect even nodes
        even = even.next

    odd.next = even_head  # Append even list after odd list

    return head

# Helper function to print linked list
def print_list(head):
    while head:
        print(head.val, end=" -> " if head.next else "")
        head = head.next
    print()

# Example usage
head = ListNode(1, ListNode(2, ListNode(3, ListNode(4, ListNode(5)))))

print("Original List:")
print_list(head)

rearranged_head = rearrange_even_nodes(head)
print("Rearranged List:")
print_list(rearranged_head)
```

```
Original List:
1 -> 2 -> 3 -> 4 -> 5
Rearranged List:
1 -> 3 -> 5 -> 2 -> 4
```

Problem 13: Given a non-negative number represented as a linked list, add one to it.

Input: 1 -> 2 -> 3 (represents the number 123) Output: 1 -> 2 -> 4 (represents the number 124)

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

# Function to reverse a linked list
def reverse_list(head):
    prev = None
    while head:
        next_node = head.next
        head.next = prev
        prev = head
        head = next_node
    return prev

# Function to add one to the number represented by linked list
def add_one(head):
    # Step 1: Reverse the list
    head = reverse_list(head)

    # Step 2: Add one
    carry = 1
    current = head

    while current:
        new_val = current.val + carry
        current.val = new_val % 10
        carry = new_val // 10  # Carry forward if needed

        if carry == 0:
            break  # No more carry, stop early

        if not current.next and carry:  # If at last node and carry still exists
            current.next = ListNode(1)  # Add new node with value 1
            carry = 0  # Stop carrying forward

        current = current.next  # Move forward

    # Step 3: Reverse back to original order
    return reverse_list(head)

# Helper function to print linked list
def print_list(head):
    while head:
        print(head.val, end=" -> " if head.next else "")
        head = head.next
    print()

# Helper function to create a linked list from a list
def create_list(values):
    head = ListNode(values[0])
    current = head
    for val in values[1:]:
        current.next = ListNode(val)
        current = current.next
    return head

# Example usage
head = create_list([1, 2, 3])

print("Original List:")
print_list(head)
```

```
updated_head = add_one(head)
print("List after adding one:")
print_list(updated_head)
```

```
Original List:
1 -> 2 -> 3
List after adding one:
1 -> 2 -> 4
```

Problem 14: Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be inserted.

Input: nums = [1, 3, 5, 6], target = 5 Output: 2

```
def search_insert(nums, target):
    left, right = 0, len(nums) - 1

    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid  # Target found
        elif nums[mid] < target:
            left = mid + 1  # Move right
        else:
            right = mid - 1  # Move left

    return left  # Insertion position

# Example usage
nums = [1, 3, 5, 6]
target = 5
print("Output:", search_insert(nums, target))  # Output: 2
```

```
Output: 2
```

Problem 15: Find the minimum element in a rotated sorted array.

Input: [4, 5, 6, 7, 0, 1, 2] Output: 0

```
def find_min(nums):
    left, right = 0, len(nums) - 1

    while left < right:
        mid = (left + right) // 2

        # If mid element is greater than rightmost, min is in the right half
        if nums[mid] > nums[right]:
            left = mid + 1
        else:
            right = mid  # Min is in the left half or is mid itself

    return nums[left]  # Minimum element

# Example usage
nums = [4, 5, 6, 7, 0, 1, 2]
print("Output:", find_min(nums))  # Output: 0
```

```
Output: 0
```

Problem 16: Search for a target value in a rotated sorted array.

Input: nums = [4, 5, 6, 7, 0, 1, 2], target = 0 Output: 4

```
def search(nums, target):
    left, right = 0, len(nums) - 1

    while left <= right:
        mid = (left + right) // 2

        if nums[mid] == target:
            return mid  # Target found

        # Determine which half is sorted
        if nums[left] <= nums[mid]:  # Left half is sorted
            if nums[left] <= target < nums[mid]:  # Target in left half
                right = mid - 1
            else:  # Target in right half
                left = mid + 1
        else:  # Right half is sorted
            if nums[mid] < target <= nums[right]:  # Target in right half
                left = mid + 1
            else:  # Target in left half
                right = mid - 1

    return -1  # Target not found

# Example usage
nums = [4, 5, 6, 7, 0, 1, 2]
target = 0
print("Output:", search(nums, target))  # Output: 4
```

```
Output: 4
```

Problem 17: Find the peak element in an array. A peak element is greater than its neighbors.

Input: nums = [1, 2, 3, 1] Output: 2 (index of peak element)

```
def find_peak_element(nums):
    left, right = 0, len(nums) - 1
    while left < right:
        mid = (left + right) // 2

        if nums[mid] > nums[mid + 1]:  # Peak is in the left half
            right = mid
        else:  # Peak is in the right half
            left = mid + 1

    return left  # or return right (both point to peak)

# Example usage
nums = [1, 2, 3, 1]
print("Output:", find_peak_element(nums))  # Output: 2
```

```
Output: 2
```

Problem 18: Given a m x n matrix where each row and column is sorted in ascending order, count the number of negative numbers.

Input: grid = [[4, 3, 2, -1], [3, 2, 1, -1], [1, 1, -1, -2], [-1, -1, -2, -3]] Output: 8

```
def count_negatives(grid):
    m, n = len(grid), len(grid[0])
    row, col = m - 1, 0  # Start at bottom-left
    count = 0

    while row >= 0 and col < n:
        if grid[row][col] < 0:
            count += (n - col)  # All elements in this row to the right are negative
            row -= 1  # Move up
        else:
            col += 1  # Move right

    return count

# Example usage
grid = [
    [4, 3, 2, -1],
    [3, 2, 1, -1],
    [1, 1, -1, -2],
    [-1, -1, -2, -3]
]
print("Output:", count_negatives(grid))  # Output: 8
```

```
Output: 8
```

Problem 19: Given a 2D matrix sorted in ascending order in each row, and the first integer of each row is greater than the last integer of the previous row, determine if a target value is present in the matrix.

Input: matrix = [[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 60]], target = 3 Output: True

```
def search_matrix(matrix, target):
    if not matrix or not matrix[0]:  # Edge case: empty matrix
        return False

    m, n = len(matrix), len(matrix[0])
    left, right = 0, m * n - 1  # Treat as 1D array

    while left <= right:
        mid = (left + right) // 2
        row, col = mid // n, mid % n  # Convert mid index to 2D index

        if matrix[row][col] == target:
            return True
        elif matrix[row][col] < target:
            left = mid + 1  # Search right half
        else:
            right = mid - 1  # Search left half

    return False  # Target not found

# Example usage
matrix = [
    [1, 3, 5, 7],
    [10, 11, 16, 20],
    [23, 30, 34, 60]
]
target = 3
print("Output:", search_matrix(matrix, target))  # Output: True
```

```
Output: True
```

Problem 20: Find Median in Two Sorted Arrays Problem: Given two sorted arrays, find the median of the combined sorted array.

Input: nums1 = [1, 3], nums2 = [2] Output: 2.0

```
def findMedianSortedArrays(nums1, nums2):
    if len(nums1) > len(nums2):  # Ensure nums1 is the smaller array
        nums1, nums2 = nums2, nums1

    x, y = len(nums1), len(nums2)
    low, high = 0, x

    while low <= high:
        midX = (low + high) // 2
        midY = (x + y + 1) // 2 - midX

        maxLeftX = float('-inf') if midX == 0 else nums1[midX - 1]
        minRightX = float('inf') if midX == x else nums1[midX]
        maxLeftY = float('-inf') if midY == 0 else nums2[midY - 1]
        minRightY = float('inf') if midY == y else nums2[midY]

        if maxLeftX <= minRightY and maxLeftY <= minRightX:
            # Odd total length
            if (x + y) % 2 == 1:
                return max(maxLeftX, maxLeftY)
            # Even total length
            return (max(maxLeftX, maxLeftY) + min(minRightX, minRightY)) / 2
        elif maxLeftX > minRightY:
            high = midX - 1  # Move left
        else:
            low = midX + 1  # Move right

# Example usage
nums1 = [1, 3]
nums2 = [2]
print("Output:", findMedianSortedArrays(nums1, nums2))  # Output: 2.0
```

```
Output: 2
```

Problem 21: Given a sorted character array and a target letter, find the smallest letter in the array that is greater than the target.

Input: letters = ['c', 'f', 'j'], target = a Output: 'c'

```
def nextGreatestLetter(letters, target):
    low, high = 0, len(letters) - 1

    while low <= high:
        mid = (low + high) // 2
        if letters[mid] > target:
            high = mid - 1  # Move left
        else:
            low = mid + 1  # Move right

    return letters[low % len(letters)]  # Wrap around

# Example usage
letters = ['c', 'f', 'j']
target = 'a'
print("Output:", nextGreatestLetter(letters, target))  # Output: 'c'
```

```
Output: c
```

Problem 22: Given an array with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

Input: nums = [2, 0, 2, 1, 1, 0] Output: [0, 0, 1, 1, 2, 2]

```
def sortColors(nums):
    low, mid, high = 0, 0, len(nums) - 1

    while mid <= high:
        if nums[mid] == 0:  # Swap 0s to front
            nums[low], nums[mid] = nums[mid], nums[low]
            low += 1
            mid += 1
        elif nums[mid] == 1:  # Leave 1s in place
            mid += 1
        else:  # Swap 2s to end
            nums[mid], nums[high] = nums[high], nums[mid]
            high -= 1  # Move high only (mid stays to re-evaluate swapped value)

# Example usage
nums = [2, 0, 2, 1, 1, 0]
sortColors(nums)
print("Output:", nums)  # Output: [0, 0, 1, 1, 2, 2]
```

```
Output: [0, 0, 1, 1, 2, 2]
```

Problem 23: Find the kth largest element in an unsorted array.

Input: nums = [3, 2, 1, 5, 6, 4], k = 2 Output: 5

```
import random

def quickselect(nums, left, right, k_smallest):
    if left == right:
        return nums[left]

    pivot_index = random.randint(left, right)
    pivot_index = partition(nums, left, right, pivot_index)

    if pivot_index == k_smallest:
        return nums[pivot_index]
    elif pivot_index > k_smallest:
        return quickselect(nums, left, pivot_index - 1, k_smallest)
    else:
        return quickselect(nums, pivot_index + 1, right, k_smallest)

def partition(nums, left, right, pivot_index):
    pivot = nums[pivot_index]
    nums[pivot_index], nums[right] = nums[right], nums[pivot_index]  # Move pivot to end
    store_index = left

    for i in range(left, right):
        if nums[i] > pivot:  # Descending order
            nums[i], nums[store_index] = nums[store_index], nums[i]
            store_index += 1
    nums[store_index], nums[right] = nums[right], nums[store_index]  # Move pivot to fina
    return store_index

def findKthLargest(nums, k):
    return quickselect(nums, 0, len(nums) - 1, len(nums) - k)

# Example usage
nums = [3, 2, 1, 5, 6, 4]
k = 2
print("Output:", findKthLargest(nums, k))  # Output: 5
```

```
Output: 2
```

Problem 24: Given an unsorted array, reorder it in-place such that nums[0] <= nums[1] >= nums[2] <= nums[3]...

Input: nums = [3, 5, 2, 1, 6, 4] Output: [3, 5, 1, 6, 2, 4]

```
def wiggleSort(nums):
    for i in range(len(nums) - 1):
        if (i % 2 == 0 and nums[i] > nums[i + 1]) or (i % 2 == 1 and nums[i] < nums[i + 1
            nums[i], nums[i + 1] = nums[i + 1], nums[i]  # Swap elements

# Example usage
nums = [3, 5, 2, 1, 6, 4]
wiggleSort(nums)
print("Output:", nums)  # Output: [3, 5, 1, 6, 2, 4]
```

```
Output: [3, 5, 1, 6, 2, 4]
```

Problem 25: Given an array of integers, calculate the sum of all its elements.

Input: [1, 2, 3, 4, 5] Output: 15

```
def arraySum(nums):
    total = 0
    for num in nums:
        total += num
    return total

# Example usage
nums = [1, 2, 3, 4, 5]
print("Output:", arraySum(nums))  # Output: 15
```

```
Output: 15
```

Problem 26: Find the maximum element in an array of integers.

Input: [3, 7, 2, 9, 4, 1] Output: 9

```
def findMax(nums):
    max_val = nums[0]  # Assume first element is max
    for num in nums:
        if num > max_val:
            max_val = num
    return max_val

# Example usage
nums = [3, 7, 2, 9, 4, 1]
print("Output:", findMax(nums))  # Output: 9
```

```
Output: 9
```

Problem 27: Implement linear search to find the index of a target element in an array.

Input: [5, 3, 8, 2, 7, 4], target = 8 Output: 2

```
def linearSearch(nums, target):
    for i in range(len(nums)):  # Traverse the array
        if nums[i] == target:   # Check if current element is the target
            return i
    return -1  # Target not found

# Example usage
nums = [5, 3, 8, 2, 7, 4]
target = 8
print("Output:", linearSearch(nums, target))  # Output: 2
```

```
Output: 2
```

Problem 28 Calculate the factorial of a given number.

Input: 5 Output: 120 (as 5! = 5 * 4 * 3 * 2 * 1 = 120)

```
def factorial(n):
    if n == 0 or n == 1:
        return 1  # Base case
    return n * factorial(n - 1)  # Recursive step

# Example usage
num = 5
print("Output:", factorial(num))  # Output: 120
```

```
Output: 120
```

Problem 29: Check if a given number is a prime number.

Input: 7 Output: True

```
import math

def isPrime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True  # 2 and 3 are prime
    if n % 2 == 0 or n % 3 == 0:
        return False

    # Check factors up to sqrt(n), skipping even numbers
    for i in range(5, int(math.sqrt(n)) + 1, 2):
        if n % i == 0:
            return False
    return True

# Example usage
num = 7
print("Output:", isPrime(num))  # Output: True
```

```
Output: True
```

Problem 30: Generate the Fibonacci series up to a given number n.

Input: 8 Output: [0, 1, 1, 2, 3, 5, 8, 13]

```
def fibonacci_recursive(n, a=0, b=1, result=None):
    if result is None:
        result = [a, b]
    next_val = a + b
    if next_val > n:
        return result
    result.append(next_val)
    return fibonacci_recursive(n, b, next_val, result)

# Example usage
num = 8
print("Output:", fibonacci_recursive(num))  # Output: [0, 1, 1, 2, 3, 5, 8, 13]
```

```
Output: [0, 1, 1, 2, 3, 5, 8]
```

Problem 31: Calculate the power of a number using recursion.

Input: base = 3, exponent = 4 Output: 81 (as 3^4 = 3 * 3 * 3 * 3 = 81)

```
def power(base, exponent):
    if exponent == 0:
        return 1  # Base case: b^0 = 1
    if exponent % 2 == 0:
        half = power(base, exponent // 2)
        return half * half  # b^e = (b^(e/2))^2
    else:
        return base * power(base, exponent - 1)  # b^e = b * b^(e-1)

# Example usage
base = 3
exponent = 4
print("Output:", power(base, exponent))  # Output: 81
```

```
Output: 81
```

Problem 32: Reverse a given string.

Input: "hello" Output: "olleh"

```
def reverse_recursive(s):
    if len(s) <= 1:
        return s  # Base case
    return s[-1] + reverse_recursive(s[:-1])  # Reverse recursively

# Example usage
input_str = "hello"
print("Output:", reverse_recursive(input_str))  # Output: "olleh"
```

```
Output: olleh
```