DSA

Assignment Questions

Problem 1: Reverse a singly linked list.

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

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

In [1]:
```
# Defining the class Node

class Node:
    def __init__(self, data = None, next = None):
        self.data = data
        self.next = next
    # Method to set the data
    def setdata(self,data):
        self.data = data
    # Method to get the data
    def getdata(self):
        return self.data
    #Method to set the next
    def setnext(self,next):
        self.next = next
    #Method to get the next
    def getnext(self):
        return self.next

# creating the nodes

head = Node(1)
node2 = Node(2)
node3 = Node(3)
node4 = Node(4)
node5 = Node(5)

# linking the nodes/ creating the linked list

head.setnext(node2)
node2.setnext(node3)
node3.setnext(node4)
node4.setnext(node5)

#Function to traverse through the linked list

def traverse(head):
    temp = head
    while(temp):
        print(temp.getdata(),end = "-->")
        temp = temp.getnext()
    print("None")

#Function to reverse the order of linked list

def reverse_linkedlist(head):
```

```
    prev = None
    current = head
    while current:
        next_node = current.getnext()
        current.setnext(prev)
        prev = current
        current = next_node
    return prev

print("Original linked-list:")
traverse(head)
print()
new_head = reverse_linkedlist(head)
print("reversed linked list:")
traverse(new_head)
```

```
Original linked-list:
1-->2-->3-->4-->5-->None

reversed linked list:
5-->4-->3-->2-->1-->None
```

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

In [2]:
```
#creating nodes of  linked list L1 ------
head_a = Node(1)
node_a1 = Node(3)
node_a2 = Node(5)

# linking the nodes of L1--------
head_a.setnext(node_a1)
node_a1.setnext(node_a2)

#creating linked list L2 ------
head_b = Node(2)
node_b1 = Node(4)
node_b2 = Node(6)

# linking the nodes of L2--------
head_b.setnext(node_b1)
node_b1.setnext(node_b2)

traverse(head_a)
traverse(head_b)

#Merging two sorted linked lists into one sorted linked list

def merge_sorted_LL(head_a, head_b):
    temp = Node()
    current = temp
    while(head_a and head_b):
        if head_a.getdata() >= head_b.getdata():
            current.setnext(Node(head_b.getdata()))
            head_b = head_b.getnext()
```

```
        else:
            current.setnext(Node(head_a.getdata()))
            head_a = head_a.getnext()
        current = current.getnext()

    if head_a:
        current.setnext(head_a)
    if head_b:
        current.setnext(head_b)

    return temp.getnext()

new_head = merge_sorted_LL(head_a, head_b)
traverse(new_head)
```

```
1-->3-->5-->None
2-->4-->6-->None
1-->2-->3-->4-->5-->6-->None
```

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

In [3]:
```
# creating the nodes

head = Node(1)
node2 = Node(2)
node3 = Node(3)
node4 = Node(4)
node5 = Node(5)

# linking the nodes/ creating the linked list

head.setnext(node2)
node2.setnext(node3)
node3.setnext(node4)
node4.setnext(node5)

#function to check the length of the linked list
def length(head):
    temp = head
    c = 0
    while temp:
        c += 1
        temp = temp.getnext()
    return c


# Function to delete the nth node from the end of the linked list
def delete_node_end(head,n):
    k = length(head) - n #n-th node from the end is k-th node from the front
    if k >= length(head) or k < 0 or not head:
        print("Enter valid value of n")
        return head
    if k == 0:
        head = head.getnext()
```

```
    else:
        #we jump to k-1`th position
        i = 0
        prev = head
        while(i<k-1):
            prev = prev.getnext()
            i+=1
        #prev will be pointing to the node left of the k`th position
        prev.setnext(prev.getnext().getnext())

    return head



print("Original linked list:")
traverse(head)
print()
n = int(input("Enter the position of the node to be deleted from the end:"))
new_head = delete_node_end(head,n)
print()
print(f"After deletion of the {n}th element from the end:")
traverse(new_head)
```

```
Original linked list:
1-->2-->3-->4-->5-->None

Enter the position of the node to be deleted from the end:2

After deletion of the 2th element from the end:
1-->2-->3-->5-->None
```

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

In [4]:
```
# List-1 is defined here

head11 =Node(1)
node2 =Node(2)
node3 =Node(3)
node4 =Node(4)

# Create Linkage of nodes
head11.setnext(node2)
node2.setnext(node3)
node3.setnext(node4)

# list-2 is define
head21 =Node(9)
node8 =Node(8)
node3 =Node(3)
node4 =Node(4)

# create linkage of nodes
head21.setnext(node8)
node8.setnext(node3)
```

```
node3.setnext(node4)
traverse(head11)
print()
traverse(head21)
print()
def IntersectPoint(head11 ,head21):
    temp1 =head11
    temp2 =head21
    while(temp1 and temp2):
        if(temp1.getdata() == temp2.getdata()):
            return temp1.getdata()
        temp1 =temp1.getnext()
        temp2 =temp2.getnext()

print("Intersection point of the two linked list is :",IntersectPoint(head11 ,head2
```

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

9-->8-->3-->4-->None

Intersection point of the two linked list is : 3
```

Problem 5: Remove duplicates from a sorted linked list.

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

Output: 1 -> 2 -> 3

In [5]:
```
# creating the nodes
head = Node(1)
node2 = Node(2)
node3 = Node(3)
node4 = Node(3)

#creating linkage
head.setnext(node2)
node2.setnext(node3)
node3.setnext(node4)

def remove_duplicates(head):
    temp = head
    while(temp and temp.getnext()):
        if temp.getdata() == temp.getnext().getdata():
            temp.setnext(temp.getnext().getnext())
        else:
            temp = temp.getnext()

print("Original linkedlist:\n")
traverse(head)
remove_duplicates(head)
print("\nAfter removing duplicates:\n")
traverse(head)
```

```
Original linkedlist:

1-->2-->3-->3-->None

After removing duplicates:

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

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)

In [6]:
```
# List 1
head1 = Node(2)
node12 = Node(4)
node13 = Node(3)
#connecting the nodes
head1.setnext(node12)
node12.setnext(node13)
#List 2
head2 = Node(5)
node22 = Node(6)
node23 = Node(4)
#connecting the nodes
head2.setnext(node22)
node22.setnext(node23)



def add_two_numbers(head1, head2):
    l1,l2 = head1,head2
    dummy_head = Node()
    current = dummy_head
    carry = 0

    while l1 or l2 or carry:
        sum_val = carry
        if l1:
            sum_val += l1.getdata()
            l1 = l1.getnext()
        if l2:
            sum_val += l2.getdata()
            l2 = l2.getnext()

        carry, digit = divmod(sum_val, 10)
        current.setnext(Node(digit))
        current = current.getnext()

    return dummy_head.getnext()

print("List1:")
traverse(head1)
```

```
print("\nList2:")
traverse(head2)
k = add_two_numbers(head1,head2)
print("\nAfter adding:")
traverse(k)
```

```
List1:
2-->4-->3-->None

List2:
5-->6-->4-->None

After adding:
7-->0-->8-->None
```

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

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

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

In [7]:
```
# creating the nodes

head = Node(1)
node2 = Node(2)
node3 = Node(3)
node4 = Node(4)

# linking the nodes/ creating the linked list

head.setnext(node2)
node2.setnext(node3)
node3.setnext(node4)

def swap_nodes(head):
    prev = dummy_node = Node()
    while(head and head.getnext()):
        firstnode = head
        secondnode = head.getnext()

        #swapping the nodes
        prev.setnext(secondnode)
        firstnode.setnext(secondnode.getnext())
        secondnode.setnext(firstnode)

        #moving the pointers
        prev = firstnode
        head = firstnode.getnext()
    return dummy_node.getnext()

print("Original list:")
traverse(head)
new_node = swap_nodes(head)
print("After swapping:")
traverse(new_node)
```

```
Original list:
1-->2-->3-->4-->None
After swapping:
2-->1-->4-->3-->None
```

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

In [8]:
```
def reverse_k_group(head, k):
    if not head or k == 1:
        return head

    dummy = Node(0)
    dummy.setnext(head)
    prev_group_end = dummy

    while prev_group_end:
        group_start = prev_group_end.getnext()
        group_end = group_start
        for _ in range(k - 1):
            group_end = group_end.getnext()
            if not group_end:
                return dummy.getnext()

        next_group_start = group_end.getnext()
        group_end.setnext(None)

        # Reverse the current group
        prev = None
        current = group_start
        while current:
            next_node = current.getnext()
            current.setnext(prev)
            prev = current
            current = next_node

        # Connect the reversed group back to the main list
        prev_group_end.setnext(group_end)
        group_start.setnext(next_group_start)

        # Update prev_group_end for the next iteration
        prev_group_end = group_start

    return dummy.getnext()

# creating the nodes

head = Node(1)
node2 = Node(2)
node3 = Node(3)
node4 = Node(4)
node5 = Node(5)

# linking the nodes/ creating the linked list
```

```
head.setnext(node2)
node2.setnext(node3)
node3.setnext(node4)
node4.setnext(node5)

print("Original Linked list:")
traverse(head)
new_head = reverse_k_group(head,3)
print("Output:")
traverse(new_head)
```

```
Original Linked list:
1-->2-->3-->4-->5-->None
Output:
3-->2-->1-->4-->5-->None
```

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

Input: 1 -> 2 -> 2 -> 1

Output: True

In [9]:

```
def is_palindrome(head):
    if not head or not head.getnext():
        return True

    # Find the middle of the linked list
    slow = head
    fast = head
    while fast.getnext() and fast.getnext().getnext():
        slow = slow.getnext()
        fast = fast.getnext().getnext()

    # Reverse the second half of the linked list
    second_half_head = reverse_linked_list(slow.getnext())

    # Compare the first half with the reversed second half
    first_half = head
    second_half = second_half_head
    while second_half:
        if first_half.getdata() != second_half.getdata():
            return False
        first_half = first_half.getnext()
        second_half = second_half.getnext()

    return True

def reverse_linked_list(head):
    prev = None
    current = head
    while current:
        next_node = current.getnext()
        current.setnext(prev)
        prev = current
        current = next_node
    return prev

# Creating a linked list
```

```
head = Node(1)
node2 = Node(2)
node3 = Node(2)
node4 = Node(1)

# Linking the nodes
head.setnext(node2)
node2.setnext(node3)
node3.setnext(node4)

# Traversing and printing the linked list
print("Linked list:")
traverse(head)

# Checking if the linked list is a palindrome
print("Is palindrome?", is_palindrome(head))
```

```
Linked list:
1-->2-->2-->1-->None
Is 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

In [10]:

```
def rotate_right(head, k):
    if not head or k == 0:
        return head

    # Find the length of the linked list
    length = 1
    tail = head
    while tail.getnext():
        tail = tail.getnext()
        length += 1

    # Calculate the actual number of rotations needed
    k %= length
    if k == 0:
        return head

    # Traverse to the (length - k) - 1 node
    prev = head
    for _ in range(length - k - 1):
        prev = prev.getnext()

    # Set the new head and break the list
    new_head = prev.getnext()
    prev.setnext(None)

    # Connect the tail to the original head
    tail.setnext(head)

    return new_head

# creating the nodes
```

```
head = Node(1)
node2 = Node(2)
node3 = Node(3)
node4 = Node(4)
node5 = Node(5)

# linking the nodes/ creating the linked list

head.setnext(node2)
node2.setnext(node3)
node3.setnext(node4)
node4.setnext(node5)

print("Original Linked list:")
traverse(head)
new_head = rotate_right(head,2)
print("Output:")
traverse(new_head)
```

```
Original Linked list:
1-->2-->3-->4-->5-->None
Output:
4-->5-->1-->2-->3-->None
```

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

In [11]:
```
class Node:
    def __init__(self, val=None, 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

    current = head
    while current:
        if current.child:
            next_node = current.next
            child_tail = flatten(current.child)
            current.next = current.child
            current.child.prev = current
            current.child = None
            if next_node:
                child_tail.next = next_node
                next_node.prev = child_tail
            current = child_tail
        else:
```

```
            current = current.next

    return head

def print_list(head):
    current = head
    while current:
        print(current.val, end=" <-> ")
        current = current.next
    print("None")

# Create the input multilevel doubly linked list
# Level 1
head = Node(1)
node2 = Node(2)
node3 = Node(3)
node7 = Node(7)
node8 = Node(8)
node11 = Node(11)
node12 = Node(12)
# Level 2
node4 = Node(4)
node5 = Node(5)
node9 = Node(9)
node10 = Node(10)
# Level 3
node6 = Node(6)
node13 = Node(13)

# Connect the nodes
head.next = node2
node2.prev = head
node2.next = node3
node3.prev = node2
node3.next = node7
node7.prev = node3
node7.next = node8
node8.prev = node7
node8.next = node11
node11.prev = node8
node11.next = node12
node12.prev = node11

node3.child = node4
node4.next = node5
node5.prev = node4
node5.next = node9
node9.prev = node5
node9.next = node10
node10.prev = node9

node9.child = node6
node6.next = node13
node13.prev = node6

# Flatten the list
head = flatten(head)

# Print the flattened list
print("Flattened list:")
print_list(head)
```

```
Flattened list:
1 <-> 2 <-> 3 <-> 4 <-> 7 <-> 8 <-> 11 <-> 12 <-> None
```

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

In [12]:
```
# Defining the class Node
class Node:
    def __init__(self, data=None, next=None):
        self.data = data
        self.next = next

    # Method to set the data
    def setdata(self, data):
        self.data = data

    # Method to get the data
    def getdata(self):
        return self.data

    # Method to set the next
    def setnext(self, next):
        self.next = next

    # Method to get the next
    def getnext(self):
        return self.next

# Function to traverse through the linked list
def traverse(head):
    temp = head
    while temp:
        print(temp.getdata(), end=" -> ")
        temp = temp.getnext()
    print("None")

def rearrange_linked_list(head):
    if not head or not head.getnext():
        return head

    # Separate the linked list into two lists: odd-positioned and even-positioned n
    odd_head = Node()
    even_head = Node()
    odd_tail = odd_head
    even_tail = even_head
    is_odd = True
    current = head
    while current:
        if is_odd:
            odd_tail.setnext(current)
            odd_tail = odd_tail.getnext()
        else:
            even_tail.setnext(current)
            even_tail = even_tail.getnext()
        is_odd = not is_odd
```

```
        current = current.getnext()

    # Connect the odd-positioned nodes list with the even-positioned nodes list
    odd_tail.setnext(even_head.getnext())
    even_tail.setnext(None)

    return odd_head.getnext()

# Creating the input linked list
head = Node(1)
node2 = Node(2)
node3 = Node(3)
node4 = Node(4)
node5 = Node(5)

# Linking the nodes
head.setnext(node2)
node2.setnext(node3)
node3.setnext(node4)
node4.setnext(node5)

# Print the original linked list
print("Original linked list:")
traverse(head)

# Rearrange the linked list
head = rearrange_linked_list(head)

# Print the rearranged linked list
print("Rearranged linked list:")
traverse(head)
```

```
Original linked list:
1 -> 2 -> 3 -> 4 -> 5 -> None
Rearranged linked list:
1 -> 3 -> 5 -> 2 -> 4 -> None
```

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)

In [13]:

```
# Defining the class Node
class Node:
    def __init__(self, data=None, next=None):
        self.data = data
        self.next = next

    # Method to set the data
    def setdata(self, data):
        self.data = data

    # Method to get the data
    def getdata(self):
        return self.data

    # Method to set the next
```

```
    def setnext(self, next):
        self.next = next

    # Method to get the next
    def getnext(self):
        return self.next

# Function to traverse through the linked list
def traverse(head):
    temp = head
    while temp:
        print(temp.getdata(), end=" -> ")
        temp = temp.getnext()
    print("None")

def add_one(head):
    # Initialize carry to 1
    carry = 1
    dummy = Node(0)
    dummy.setnext(head)
    current = head
    last_non_nine = dummy

    # Traverse the linked list from right to left
    while current:
        if current.getdata() != 9:
            last_non_nine = current
        current = current.getnext()

    # Add one to the last non-nine node
    last_non_nine.setdata(last_non_nine.getdata() + 1)

    # Update the nodes after the last non-nine node
    current = last_non_nine.getnext()
    while current:
        current.setdata(0)
        current = current.getnext()

    # If the dummy node's value is 1, return dummy.next, otherwise return dummy
    if dummy.getdata() == 1:
        return dummy.getnext()
    else:
        return dummy

# Creating the input linked list
head = Node(1)
node2 = Node(2)
node3 = Node(3)

# Linking the nodes
head.setnext(node2)
node2.setnext(node3)

# Print the original linked list
print("Original linked list:")
traverse(head)

# Add one to the linked list
head = add_one(head)

# Print the modified linked list
print("Modified linked list:")
traverse(head)
```

```
Original linked list:
1 -> 2 -> 3 -> None
Modified linked list:
0 -> 1 -> 2 -> 4 -> None
```

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

In [14]:
```python
def search_insert_position(nums, target):
    # Base case: If the array is empty, return 0
    if not nums:
        return 0

    left, right = 0, len(nums) - 1

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

        # If target is found, return the index
        if nums[mid] == target:
            return mid

        # If target is smaller than the value at mid, search in the left half
        elif nums[mid] > target:
            right = mid - 1

        # If target is larger than the value at mid, search in the right half
        else:
            left = mid + 1

    # If target is not found, left will be the index where it would be inserted
    return left

# Test the function
nums = [1, 3, 5, 6]
target = 5
print("Input:", nums, "Target:", target)
print("Output:", search_insert_position(nums, target))
```

```
Input: [1, 3, 5, 6] Target: 5
Output: 2
```

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

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

Output: 0

In [15]:
```python
def find_min_in_rotated_array(nums):
    left, right = 0, len(nums) - 1

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

        # If the middle element is greater than the last element, search in the rig
        if nums[mid] > nums[right]:
            left = mid + 1
        # If the middle element is smaller than or equal to the last element, searc
        else:
            right = mid

    # At the end of the loop, left will point to the minimum element
    return nums[left]

# Test the function
nums = [4, 5, 6, 7, 0, 1, 2]
print("Input:", nums)
print("Output:", find_min_in_rotated_array(nums))
```

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

In [16]:
```python
def search_rotated_sorted_array(nums, target):
    left, right = 0, len(nums) - 1

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

        if nums[mid] == target:
            return mid

        # Check if left half is sorted
        if nums[left] <= nums[mid]:
            # Check if target lies in the left half
            if nums[left] <= target < nums[mid]:
                right = mid - 1
            else:
                left = mid + 1
        # Right half is sorted
        else:
            # Check if target lies in the right half
            if nums[mid] < target <= nums[right]:
                left = mid + 1
            else:
                right = mid - 1

    return -1
```

```
# Test the function
nums = [4, 5, 6, 7, 0, 1, 2]
target = 0
print("Input:", nums, "Target:", target)
print("Output:", search_rotated_sorted_array(nums, target))
```

```
Input: [4, 5, 6, 7, 0, 1, 2] Target: 0
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)

In [17]:
```
def find_peak_element(nums):
    left, right = 0, len(nums) - 1

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

        # Check if mid is a peak element
        if nums[mid] > nums[mid + 1]:
            right = mid
        else:
            left = mid + 1

    return left

# Test the function
nums = [1, 2, 3, 1]
print("Input:", nums)
print("Output:", find_peak_element(nums),"(index of peak element)")
```

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

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

In [18]:
```python
def count_negatives(grid):
    m, n = len(grid), len(grid[0])
    count = 0
    row, col = 0, n - 1  # Start from the top-right corner

    while row < m and col >= 0:
        if grid[row][col] < 0:
            # All elements to the left of grid[row][col] will also be negative
            count += (m - row)
            col -= 1  # Move left
        else:
            row += 1  # Move down

    return count

# Test the function
grid = [
    [4, 3, 2, -1],
    [3, 2, 1, -1],
    [1, 1, -1, -2],
    [-1, -1, -2, -3]
]
print("Input:")
for row in grid:
    print(row)
print("Output:", count_negatives(grid))
```

```
Input:
[4, 3, 2, -1]
[3, 2, 1, -1]
[1, 1, -1, -2]
[-1, -1, -2, -3]
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

In [19]:
```python
def search_matrix(matrix, target):
    if not matrix or not matrix[0]:
        return False

    m, n = len(matrix), len(matrix[0])
    left, right = 0, m * n - 1

    while left <= right:
        mid = left + (right - left) // 2
        mid_val = matrix[mid // n][mid % n]
```

```
        if mid_val == target:
            return True
        elif mid_val < target:
            left = mid + 1
        else:
            right = mid - 1

    return False

# Test the function
matrix = [
    [1, 3, 5, 7],
    [10, 11, 16, 20],
    [23, 30, 34, 60]
]
target = 3
print("Input:")
for row in matrix:
    print(row)
print("Target:", target)
print("Output:", search_matrix(matrix, target))
```

```
Input:
[1, 3, 5, 7]
[10, 11, 16, 20]
[23, 30, 34, 60]
Target: 3
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

In [20]:

```
def find_median_sorted_arrays(nums1, nums2):
    merged = []
    i, j = 0, 0

    while i < len(nums1) and j < len(nums2):
        if nums1[i] < nums2[j]:
            merged.append(nums1[i])
            i += 1
        else:
            merged.append(nums2[j])
            j += 1

    # Add remaining elements from nums1 (if any)
    while i < len(nums1):
        merged.append(nums1[i])
        i += 1
```

```
    # Add remaining elements from nums2 (if any)
    while j < len(nums2):
        merged.append(nums2[j])
        j += 1

    n = len(merged)
    if n % 2 == 0:
        return (merged[n // 2 - 1] + merged[n // 2]) / 2
    else:
        return merged[n // 2]

# Test the function
nums1 = [1, 3]
nums2 = [2]
print("Input:")
print("nums1 =", nums1)
print("nums2 =", nums2)
print("Median:", find_median_sorted_arrays(nums1, nums2))
```

```
Input:
nums1 = [1, 3]
nums2 = [2]
Median: 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

In [21]:

```
def next_greatest_letter(letters, target):
    left, right = 0, len(letters) - 1

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

        # If mid is less than or equal to the target, search in the right half
        if letters[mid] <= target:
            left = mid + 1
        # If mid is greater than the target, update the result and search in the le
        else:
            result = letters[mid]
            right = mid - 1

    # If all letters are less than or equal to the target, return the first letter
    return letters[left % len(letters)]

# Test the function
letters = ['c', 'f', 'j']
target = 'a'
print("Input:")
print("letters =", letters)
print("target =", target)
print("Output:", next_greatest_letter(letters, target))
```

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

In [23]:
```python
def sortColors(nums):
    # Initialize pointers for the boundaries of each color
    red, white, blue = 0, 0, len(nums) - 1

    # Iterate through the array
    while white <= blue:
        if nums[white] == 0:
            # If the current element is 0, swap it with the element at the red poir
            nums[red], nums[white] = nums[white], nums[red]
            # Move both red and white pointers to the right
            red += 1
            white += 1
        elif nums[white] == 1:
            # If the current element is 1, just move the white pointer to the right
            white += 1
        else:
            # If the current element is 2, swap it with the element at the blue poi
            nums[white], nums[blue] = nums[blue], nums[white]
            # Move the blue pointer to the left
            blue -= 1



nums = [2, 0, 2, 1, 1, 0]
sortColors(nums)
print(nums)
```

```
[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

In [25]:
```python
# first we sort the array in descending order
def bubblesort(arr):
    for i in range(len(arr)-1, 0, -1):
```

```
        for j in range(i):
            if (arr[j] < arr[j+1]):
                arr[j],arr[j+1] = arr[j+1],arr[j]
    return arr

def find_k_largest(arr,k):
    bubblesort(arr)
    return arr[k-1]

print(find_k_largest([3, 2, 1, 5, 6, 4], k = 2))
```

```
5
```

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]

In [28]:
```
def altSort(nums):

    # Iterate through the array starting from the second element
    for i in range(1, len(nums)):
        if (i % 2 == 0 and nums[i] > nums[i - 1]) or (i % 2 != 0 and nums[i] < nums
            nums[i], nums[i - 1] = nums[i - 1], nums[i]

# Test the function with the given input
nums = [3, 5, 2, 1, 6, 4]
altSort(nums)
print(nums)
```

```
[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

In [30]:
```
def sum_arr(arr):
    sumele = 0
    for _ in arr:
        sumele += _
    return sumele
print(sum_arr([1,2,3,4,5]))
```

```
15
```

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

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

Output: 9

In [31]:
```
def find_max_ele(arr):
    max_ele = arr[0]
    for i in arr:
        if i > max_ele:
            max_ele = i
    return max_ele
print(find_max_ele([3,7,2,9,4,1]))
```

```
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

In [36]:
```
def find_index(arr,target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1 #when target is not in array
print(find_index([5,3,8,2,7,4],8))
```

```
2
```

Problem 28: Calculate the factorial of a given number.

Input: 5

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

In [38]:
```
def fact(num):
    if num == 0 or n == 1:
        return 1
    else:
        return num*fact(num-1)
print(fact(5))
```

```
120
```

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

Input: 7

Output: True

In [45]:
```
def isprime(num):
    if num < 2:
        return False
    if num == 2:
        return True
    if num % 2 == 0:
        return False
    for i in range(3, int(num**0.5) + 1, 2):
        if num % i == 0:
            return False
    return True
print(isprime(7))
```

```
True
```

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

Input: 8

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

In [50]:
```
def fibonacci(n):
    fib_series = [0, 1]
    for i in range(2, n):
        fib_series.append(fib_series[-1] + fib_series[-2])
    return fib_series

# Test the function with the given input
print(fibonacci(8))
```

```
[0, 1, 1, 2, 3, 5, 8, 13]
```

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)

In [52]:
```
def power(base,exponent):
    if exponent == 0:
        return 1
    elif exponent == 1:
        return base
    else:
        return base * power(base,exponent-1)

print(power(3,4))
```

```
81
```

Problem 32: Reverse a given string.

Input: "hello"

Output: "olleh"

In [55]:
```
def rev_str(string):
    rev = ""
    for char in string:
        rev = char + rev
    return rev

print("Reversed string:",rev_str("hello"))
```

```
Reversed string: olleh
```

In [ ]: