DSA _Assignment _

January 1, 2024

[1]:

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

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

def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

# Helper function to create a linked list from a list of values
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

# Get user input for linked list values
values = list(map(int, input("Enter values for the linked list␣
 ↪(space-separated): ").split()))
```

```python
# Create linked list
linked_list = create_linked_list(values)

# Problem 1: Reverse a singly linked list.
reversed_list = reverse_linked_list(linked_list)

# Print the original and reversed linked lists
print("Original Linked List:")
print_linked_list(linked_list)

print("\nReversed Linked List:")
print_linked_list(reversed_list)
```

```
Enter values for the linked list (space-separated):  1 2 3 4 5

Original Linked List:
1 -> None

Reversed Linked List:
5 -> 4 -> 3 -> 2 -> 1 -> None
```

[4]:

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

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

def merge_sorted_lists(list1, list2):
    dummy = ListNode()
    current = dummy
    while list1 and list2:
        if list1.value < list2.value:
            current.next = list1
            list1 = list1.next
        else:
            current.next = list2
            list2 = list2.next
        current = current.next
    current.next = list1 or list2
    return dummy.next

def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")
```

```
# Helper function to create a linked list from a list of values
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

# Get user input for values of the first sorted linked list
list1_values = list(map(int, input("Enter values for the first sorted linked␣
 ↪list (space-separated): ").split()))

# Get user input for values of the second sorted linked list
list2_values = list(map(int, input("Enter values for the second sorted linked␣
 ↪list (space-separated): ").split()))

# Create linked lists
list1 = create_linked_list(list1_values)
list2 = create_linked_list(list2_values)

# Problem 2: Merge two sorted linked lists into one sorted linked list.
merged_list = merge_sorted_lists(list1, list2)

# Print the original and merged linked lists
print("\nFirst Sorted Linked List:")
print_linked_list(list1)

print("\nSecond Sorted Linked List:")
print_linked_list(list2)

print("\nMerged Sorted Linked List:")
print_linked_list(merged_list)
```

```
Enter values for the first sorted linked list (space-separated):  1 3 5
Enter values for the second sorted linked list (space-separated):  2 4 6


First Sorted Linked List:
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> None

Second Sorted Linked List:
2 -> 3 -> 4 -> 5 -> 6 -> None

Merged Sorted Linked List:
```

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

[5]:

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

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

def remove_nth_from_end(head, n):
    dummy = ListNode(0)
    dummy.next = head
    fast = slow = dummy

    # Move the fast pointer n+1 steps ahead
    for _ in range(n + 1):
        fast = fast.next

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

    # Remove the nth node from the end
    slow.next = slow.next.next

    return dummy.next

def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

# Helper function to create a linked list from a list of values
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

# Get user input for linked list values
```

```
values = list(map(int, input("Enter values for the linked list␣
 ↪(space-separated): ").split()))

# Create linked list
linked_list = create_linked_list(values)

# Get user input for the value of n
n = int(input("Enter the value of n: "))

# Problem 3: Remove the nth node from the end of a linked list.
modified_list = remove_nth_from_end(linked_list, n)

# Print the original and modified linked lists
print("\nOriginal Linked List:")
print_linked_list(linked_list)

print("\nLinked List after removing the nth node from the end:")
print_linked_list(modified_list)
```

```
Enter values for the linked list (space-separated):  1 2 3 4 5
Enter the value of n:  2


Original Linked List:
1 -> 2 -> 3 -> 5 -> None

Linked List after removing the nth node from the end:
1 -> 2 -> 3 -> 5 -> None
```

[1]:

```
#Problem 5: Remove duplicates from a sorted linked list.

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

def remove_nth_from_end(head, n):
    dummy = ListNode(0)
    dummy.next = head
    fast = slow = dummy

    # Move the fast pointer n+1 steps ahead
    for _ in range(n + 1):
        fast = fast.next

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

    # Remove the nth node from the end
    slow.next = slow.next.next

    return dummy.next

def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

# Helper function to create a linked list from a list of values
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

# Get user input for linked list values
values = list(map(int, input("Enter values for the linked list␣
 ↪(space-separated): ").split()))

# Create linked list
linked_list = create_linked_list(values)

# Get user input for the value of n
n = int(input("Enter the value of n: "))

# Problem 3: Remove the nth node from the end of a linked list.
modified_list = remove_nth_from_end(linked_list, n)

# Print the original and modified linked lists
print("\nOriginal Linked List:")
print_linked_list(linked_list)

print("\nLinked List after removing the nth node from the end:")
print_linked_list(modified_list)

#Problem 5: Remove duplicates from a sorted linked list.
```

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

def remove_duplicates(head):
    current = head
    while current and current.next:
        if current.value == current.next.value:
            current.next = current.next.next
        else:
            current = current.next
    return head

def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

# Helper function to create a linked list from a list of values
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

# Get user input for sorted linked list values
values = list(map(int, input("Enter values for the sorted linked list (space-separated): ").split()))

# Create sorted linked list
sorted_linked_list = create_linked_list(values)

# Problem 5: Remove duplicates from a sorted linked list.
unique_list = remove_duplicates(sorted_linked_list)

# Print the original and modified linked lists
print("\nSorted Linked List with Duplicates:")
print_linked_list(sorted_linked_list)

print("\nSorted Linked List after Removing Duplicates:")
```

```
print_linked_list(unique_list)
```

```
Enter values for the linked list (space-separated):  1 1 2 3 3
Enter the value of n:  4


Original Linked List:
1 -> 2 -> 3 -> 3 -> None

Linked List after removing the nth node from the end:
1 -> 2 -> 3 -> 3 -> None

Enter values for the sorted linked list (space-separated):  1 2 3 3


Sorted Linked List with Duplicates:
1 -> 2 -> 3 -> None

Sorted Linked List after Removing Duplicates:
1 -> 2 -> 3 -> None
```

[2]:

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

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

def add_two_numbers(list1, list2):
    dummy = ListNode()
    current = dummy
    carry = 0

    while list1 or list2 or carry:
        sum_val = (list1.value if list1 else 0) + (list2.value if list2 else 0)␣
 ↪+ carry
        carry, digit = divmod(sum_val, 10)
        current.next = ListNode(digit)
        current = current.next

        if list1:
            list1 = list1.next
        if list2:
            list2 = list2.next

    return dummy.next
```

```python
def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

# Helper function to create a linked list from a list of values
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

# Get user input for values of the first number linked list
num1_values = list(map(int, input("Enter values for the first number linked list (space-separated): ").split()))

# Get user input for values of the second number linked list
num2_values = list(map(int, input("Enter values for the second number linked list (space-separated): ").split()))

# Create linked lists for the two numbers
num1 = create_linked_list(num1_values)
num2 = create_linked_list(num2_values)

# Problem 6: Add two numbers represented by linked lists.
sum_list = add_two_numbers(num1, num2)

# Print the two numbers and their sum
print("\nFirst Number Linked List:")
print_linked_list(num1)

print("\nSecond Number Linked List:")
print_linked_list(num2)

print("\nSum of Two Numbers Linked List:")
print_linked_list(sum_list)
```

```
Enter values for the first number linked list (space-separated):  2 4 3
Enter values for the second number linked list (space-separated):  5 6 4


First Number Linked List:
```

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

Second Number Linked List:
5 -> 6 -> 4 -> None

Sum of Two Numbers Linked List:
7 -> 0 -> 8 -> None
```

[3]:

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

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

def swap_pairs(head):
    dummy = ListNode()
    dummy.next = head
    current = dummy

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

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

        current = current.next.next

    return dummy.next

def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

# Helper function to create a linked list from a list of values
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

# Get user input for linked list values
values = list(map(int, input("Enter values for the linked list␣
 ↪(space-separated): ").split()))

# Create linked list
linked_list = create_linked_list(values)

# Problem 7: Swap nodes in pairs in a linked list.
swapped_list = swap_pairs(linked_list)

# Print the original and swapped linked lists
print("\nOriginal Linked List:")
print_linked_list(linked_list)

print("\nLinked List after Swapping Nodes in Pairs:")
print_linked_list(swapped_list)
```

```
Enter values for the linked list (space-separated):  1 2 3 4


Original Linked List:
1 -> 4 -> 3 -> None

Linked List after Swapping Nodes in Pairs:
2 -> 1 -> 4 -> 3 -> None
```

[10]:

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

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

def reverse_k_groups(head, k):
    def reverse_group(start, end):
        prev, current = None, start
        while current != end:
            next_node = current.next
            current.next = prev
            prev = current
            current = next_node
        return prev

    dummy = ListNode()
    dummy.next = head
    current = dummy

    while True:
        start = current.next
        end = current
        for _ in range(k):
            end = end.next
            if not end:
                return dummy.next
        next_group = end.next
        reversed_group = reverse_group(start, end)
        current.next = reversed_group
        start.next = next_group
        current = start

def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

# Helper function to create a linked list from a list of values
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

# Get user input for linked list values
values = list(map(int, input("Enter values for the linked list␣
 ↪(space-separated): ").split()))

# Create linked list
linked_list = create_linked_list(values)

# Get user input for the value of k
k = int(input("Enter the value of k: "))

# Problem 8: Reverse nodes in a linked list in groups of k.
reversed_list = reverse_k_groups(linked_list, k)
```

```
# Print the original and reversed linked lists
print("\nOriginal Linked List:")
print_linked_list(linked_list)

print(f"\nLinked List after Reversing Nodes in Groups of {k}:")
print_linked_list(reversed_list)
```

```
Enter values for the linked list (space-separated):  1 2 3 4 5
Enter the value of k:  3


Original Linked List:
1 -> 4 -> 5 -> None

Linked List after Reversing Nodes in Groups of 3:
2 -> 1 -> 4 -> 5 -> None
```

[11]:

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

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

def is_palindrome(head):
    def reverse_list(start):
        prev, current = None, start
        while current:
            next_node = current.next
            current.next = prev
            prev = current
            current = next_node
        return prev

    slow = fast = head

    # Move slow and fast pointers to find the middle of the list
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next

    # Reverse the second half of the list
    reversed_second_half = reverse_list(slow)

    # Compare the first half with the reversed second half
    while reversed_second_half:
        if head.value != reversed_second_half.value:
            return False
        head = head.next
        reversed_second_half = reversed_second_half.next

    return True

def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

# Helper function to create a linked list from a list of values
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

# Get user input for linked list values
values = list(map(int, input("Enter values for the linked list␣
 ↪(space-separated): ").split()))

# Create linked list
linked_list = create_linked_list(values)

# Problem 9: Determine if a linked list is a palindrome.
result = is_palindrome(linked_list)

# Print the original linked list and the result
print("\nOriginal Linked List:")
print_linked_list(linked_list)

print("\nIs the Linked List a Palindrome?", result)
```

```
Enter values for the linked list (space-separated):  1 2 2 1


Original Linked List:
1 -> 2 -> 2 -> None

Is the Linked List a Palindrome? True
```

[13]:

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

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

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.next:
        length += 1
        tail = tail.next

    # Calculate the effective rotation
    k = k % length

    if k == 0:
        return head

    # Move the tail to the new end after rotation
    new_tail_position = length - k
    new_tail = head
    for _ in range(new_tail_position - 1):
        new_tail = new_tail.next

    # Perform the rotation
    new_head = new_tail.next
    new_tail.next = None
    tail.next = head

    return new_head

def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

# Helper function to create a linked list from a list of values
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

# Get user input for linked list values
values = list(map(int, input("Enter values for the linked list␣
 ↪(space-separated): ").split()))

# Create linked list
linked_list = create_linked_list(values)

# Get user input for the value of k
k = int(input("Enter the value of k: "))

# Problem 10: Rotate a linked list to the right by k places.
rotated_list = rotate_right(linked_list, k)

# Print the original and rotated linked lists
print("\nOriginal Linked List:")
print_linked_list(linked_list)

print(f"\nLinked List after Rotating to the Right by {k} places:")
print_linked_list(rotated_list)
```

```
Enter values for the linked list (space-separated):  1 2 3 4 5
Enter the value of k:  2


Original Linked List:
1 -> 2 -> 3 -> None

Linked List after Rotating to the Right by 2 places:
4 -> 5 -> 1 -> 2 -> 3 -> None
```

[16]:

```
#Problem 11: Flatten a multilevel doubly linked list.

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

def flatten_multilevel_dll(head):
    current = head

    while current:
        if current.child:
            next_node = current.next
            current.next = current.child
            current.child.prev = current
            current.child = None

            child_end = current.next
            while child_end.next:
                child_end = child_end.next

            if next_node:
                child_end.next = next_node
                next_node.prev = child_end

        current = current.next

    return head

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

# Helper function to create a multilevel doubly linked list
def create_multilevel_dll(values):
    if not values:
        return None

    head = Node(values[0])
    current = head
    for value in values[1:]:
        current.next = Node(value, prev=current)
        current = current.next

    return head

# Get user input for multilevel doubly linked list values
values = list(map(int, input("Enter values for the multilevel doubly linked list (space-separated): ").split()))

# Create multilevel doubly linked list
multilevel_dll = create_multilevel_dll(values)
```

```python
# Problem 11: Flatten a multilevel doubly linked list.
flattened_dll = flatten_multilevel_dll(multilevel_dll)

# Print the original and flattened multilevel doubly linked lists
print("\nOriginal Multilevel Doubly Linked List:")
print_multilevel_dll(multilevel_dll)

print("\nFlattened Multilevel Doubly Linked List:")
print_multilevel_dll(flattened_dll)
```

```
Enter values for the multilevel doubly linked list (space-separated):  1 2 3 7 8
11 12 4 5 9 10 6 13


Original Multilevel Doubly Linked List:
1 <-> 2 <-> 3 <-> 7 <-> 8 <-> 11 <-> 12 <-> 4 <-> 5 <-> 9 <-> 10 <-> 6 <-> 13
<-> None

Flattened Multilevel Doubly Linked List:
1 <-> 2 <-> 3 <-> 7 <-> 8 <-> 11 <-> 12 <-> 4 <-> 5 <-> 9 <-> 10 <-> 6 <-> 13
<-> None
```

[17]:

```python
#Problem 12: Rearrange a linked list such that all even positioned nodes are␣
 ↪placed at the end.

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

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

    odd_head = head
    even_head = head.next
    odd_current = odd_head
    even_current = even_head

    while even_current and even_current.next:
        odd_current.next = even_current.next
        odd_current = odd_current.next
        even_current.next = odd_current.next
        even_current = even_current.next

    odd_current.next = even_head
    return odd_head

def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

# Helper function to create a linked list from a list of values
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

# Get user input for linked list values
values = list(map(int, input("Enter values for the linked list (space-separated): ").split()))

# Create linked list
linked_list = create_linked_list(values)

# Problem 12: Rearrange a linked list such that all even positioned nodes are placed at the end.
rearranged_list = rearrange_linked_list(linked_list)

# Print the original and rearranged linked lists
print("\nOriginal Linked List:")
print_linked_list(linked_list)

print("\nRearranged Linked List:")
print_linked_list(rearranged_list)
```

```
Enter values for the linked list (space-separated):  1 2 3 4 5


Original Linked List:
1 -> 3 -> 5 -> 2 -> 4 -> None

Rearranged Linked List:
1 -> 3 -> 5 -> 2 -> 4 -> None
```

[18]:

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

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

def add_one_to_linked_list(head):
    dummy = ListNode(0)
    dummy.next = head
    current = dummy

    # Find the rightmost non-nine digit
    last_non_nine = None
    while current:
        if current.value != 9:
            last_non_nine = current
        current = current.next

    # Increment the rightmost non-nine digit (if exists) and set the following digits to zero
    if last_non_nine:
        last_non_nine.value += 1
        current = last_non_nine.next
        while current:
            current.value = 0
            current = current.next
    else:
        # If no non-nine digit found, insert a new digit at the beginning
        new_digit = ListNode(1)
        new_digit.next = dummy.next
        dummy.next = new_digit

    return dummy.next

def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

# Helper function to create a linked list from a list of values
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

# Get user input for linked list values
values = list(map(int, input("Enter values for the linked list␣
 ↪(space-separated): ").split()))

# Create linked list
linked_list = create_linked_list(values)

# Problem 13: Given a non-negative number represented as a linked list, add one␣
 ↪to it.
result_list = add_one_to_linked_list(linked_list)

# Print the original linked list and the result
print("\nOriginal Linked List:")
print_linked_list(linked_list)

print("\nLinked List after Adding One:")
print_linked_list(result_list)
```

```
Enter values for the linked list (space-separated):  1 2 3


Original Linked List:
1 -> 2 -> 4 -> None

Linked List after Adding One:
1 -> 2 -> 4 -> None
```

[19]:

```
 #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.

def search_insert_position(nums, target):
    if not nums:
        return 0

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

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

```python
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return left

# Get user input for the sorted array
nums = list(map(int, input("Enter a sorted array (space-separated): ").split()))

# Get user input for the target value
target = int(input("Enter the target value: "))

# Problem 14: Return the index if the target is found, otherwise return the␣
 ↪index where it would be inserted.
result = search_insert_position(nums, target)

print(f"\nIndex of Target ({target}): {result}")
```

```
Enter a sorted array (space-separated):  1 3 5 6
Enter the target value:  5


Index of Target (5): 2
```

[20]:

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

def find_minimum_in_rotated_sorted_array(nums):
    if not nums:
        return None

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

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

        if nums[mid] > nums[right]:
            left = mid + 1
        else:
            right = mid

    return nums[left]

# Get user input for the rotated sorted array
nums = list(map(int, input("Enter a rotated sorted array (space-separated): ").
 ↪split()))
```

```python
# Problem 15: Find the minimum element in a rotated sorted array.
minimum_element = find_minimum_in_rotated_sorted_array(nums)

print(f"\nMinimum Element in Rotated Sorted Array: {minimum_element}")
```

```
Enter a rotated sorted array (space-separated):  4 5 6 7 0 1 2


Minimum Element in Rotated Sorted Array: 0
```

[1]:

```python
#Problem 17: Find the peak element in an array. A peak element is greater than its neighbors.
def find_peak_element(nums):
    if not nums:
        return None  # No peak element in an empty array

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

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

        if nums[mid] > nums[mid + 1]:
            # The peak element is on the left side, including mid
            right = mid
        else:
            # The peak element is on the right side, excluding mid
            left = mid + 1

    return nums[left]

# Example usage:
nums = [1, 2, 3, 1]
peak_element = find_peak_element(nums)
print(f"The peak element is: {peak_element}")
```

```
The peak element is: 3
```

[11]:

```python
##Problem 18: Given a m x n matrix where each row and column is sorted in ascending order, count the number
#of negative numbers.
def count_negatives(matrix):
    if not matrix or not matrix[0]:
        return 0  # Empty matrix or empty row

    rows, cols = len(matrix), len(matrix[0])
    count = 0
    row, col = rows - 1, 0  # Start from the bottom-left corner
    while row >= 0 and col < cols:
        if matrix[row][col] < 0:
            # All elements to the right of matrix[row][col] will also be␣
 ↪negative
            count += (cols - col)
            row -= 1
        else:
            col += 1

    return count

# Example usage:
matrix = [
    [-3, -2, -1, 1],
    [-2,  2,  3,  4],
    [ 4,  5,  7,  8]
]

negatives_count = count_negatives(matrix)
print(f"The number of negative numbers is: {negatives_count}")
```

```
The number of negative numbers is: 0
```

[2]:

```
#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.

def search_matrix(matrix, target):
    if not matrix or not matrix[0]:
        return False  # Empty matrix or empty row

    rows, cols = len(matrix), len(matrix[0])
    row, col = 0, cols - 1  # Start from the top-right corner

    while row < rows and col >= 0:
        if matrix[row][col] == target:
            return True
        elif matrix[row][col] < target:
            row += 1  # Move down in the current column
        else:
            col -= 1  # Move left in the current row

    return False

# Input prompt for the matrix
```

```python
def input_matrix():
    matrix = []
    rows = int(input("Enter the number of rows: "))
    cols = int(input("Enter the number of columns: "))

    print("Enter the matrix elements row-wise:")
    for i in range(rows):
        row = list(map(int, input().split()))
        matrix.append(row)

    return matrix

# Input target value
target = int(input("Enter the target value: "))

# Get the matrix from the user
matrix = input_matrix()

# Check if the target value is present in the matrix
result = search_matrix(matrix, target)

# Display the result
if result:
    print(f"The target value {target} is present in the matrix.")
else:
    print(f"The target value {target} is not present in the matrix.")
```

```
Enter the target value:  3
Enter the number of rows:  3
Enter the number of columns:  4
```

```
Enter the matrix elements row-wise:
```

```
 1 3 5 7
 10 11 16 20
 13 30 34 60
```

```
The target value 3 is present in the matrix.
```

[3]:

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

def find_median_sorted_arrays(nums1, nums2):
    # Merge the sorted arrays
    merged_array = merge_sorted_arrays(nums1, nums2)

    # Calculate the median based on the length of the merged array
    median_index = len(merged_array) // 2
    if len(merged_array) % 2 == 0:
        # If the length is even, take the average of the two middle elements
        median = (merged_array[median_index - 1] + merged_array[median_index]) /
 ↪ 2
    else:
        # If the length is odd, take the middle element
        median = merged_array[median_index]

    return median

def merge_sorted_arrays(nums1, nums2):
    merged_array = []
    i, j = 0, 0

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

    # Append the remaining elements from both arrays
    merged_array.extend(nums1[i:])
    merged_array.extend(nums2[j:])

    return merged_array

# Example usage:
nums1 = [1, 3]
nums2 = [2]

median = find_median_sorted_arrays(nums1, nums2)
print(f"The median of the combined sorted arrays is: {median}")
```

```
The median of the combined sorted arrays is: 2
```

[4]:

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

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

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

        if letters[mid] <= target:
            left = mid + 1
        else:
            right = mid - 1

    # If the right pointer is out of bounds, return the first element
    return letters[left] if left <= len(letters) - 1 else letters[0]

# Example usage:
sorted_letters = ['a', 'c', 'f', 'h']
target_letter = 'f'

result = next_greatest_letter(sorted_letters, target_letter)
print(f"The smallest letter greater than {target_letter} is: {result}")
```

```
The smallest letter greater than f is: h
```

[5]:

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

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

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

        if letters[mid] <= target:
            left = mid + 1
        else:
            right = mid - 1

    # If the right pointer is out of bounds, return the first element
    return letters[left] if left <= len(letters) - 1 else letters[0]

# Input prompt for the sorted character array
def input_sorted_letters():
    letters = input("Enter the sorted character array (each character separated␣
 ↪by space): ").split()
    return sorted(letters)

# Input prompt for the target letter
target_letter = input("Enter the target letter: ")

# Get the sorted letters from the user
sorted_letters = input_sorted_letters()

# Find the smallest letter greater than the target
```

```
result = next_greatest_letter(sorted_letters, target_letter)
print(f"The smallest letter greater than {target_letter} is: {result}")
```

```
Enter the target letter:  a
Enter the sorted character array (each character separated by space):  c f j
```

```
The smallest letter greater than a is: c
```

[6]:

```
#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.

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

    while mid <= high:
        if nums[mid] == 0:
            nums[low], nums[mid] = nums[mid], nums[low]
            low += 1
            mid += 1
        elif nums[mid] == 1:
            mid += 1
        else:
            nums[mid], nums[high] = nums[high], nums[mid]
            high -= 1

# Input prompt for the array of colors
def input_colors():
    colors = input("Enter the array of colors (each color separated by space, use 0 for red, 1 for white, and 2 for blue): ").split()
    return [int(color) for color in colors]

# Get the array of colors from the user
colors_array = input_colors()

# Sort the array of colors in-place
sort_colors(colors_array)

# Display the sorted array
print("The sorted array of colors is:", colors_array)
```

```
Enter the array of colors (each color separated by space, use 0 for red, 1 for
white, and 2 for blue):  2 0 2 1 1 0
```

```
The sorted array of colors is: [0, 0, 1, 1, 2, 2]
```

[7]:

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

```
def find_kth_largest(nums, k):
    nums.sort(reverse=True)
    return nums[k - 1]

# Input prompt for the array
def input_array():
    array = input("Enter the unsorted array (each element separated by space):␣
 ↪").split()
    return [int(element) for element in array]

# Input prompt for k
k = int(input("Enter the value of k: "))

# Get the unsorted array from the user
unsorted_array = input_array()

# Find the kth largest element
kth_largest = find_kth_largest(unsorted_array, k)

# Display the result
print(f"The {k}th largest element in the array is: {kth_largest}")
```

```
Enter the value of k:  2
Enter the unsorted array (each element separated by space):  3 2 1 5 6 4

The 2th largest element in the array is: 5
```

[8]:

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

def wiggle_sort(nums):
    n = len(nums)

    for i in range(1, n, 2):
        if i < n - 1 and nums[i] < nums[i + 1]:
            nums[i], nums[i + 1] = nums[i + 1], nums[i]

    for i in range(0, n, 2):
        if i < n - 1 and nums[i] > nums[i + 1]:
            nums[i], nums[i + 1] = nums[i + 1], nums[i]

# Input prompt for the array
def input_array():
    array = input("Enter the unsorted array (each element separated by space):␣
 ↪").split()
    return [int(element) for element in array]
```

```python
# Get the unsorted array from the user
unsorted_array = input_array()

# Reorder the array in-place
wiggle_sort(unsorted_array)

# Display the result
print("The reordered array is:", unsorted_array)
```

```
Enter the unsorted array (each element separated by space):  3 5 2 1 6 4

The reordered array is: [3, 5, 2, 6, 1, 4]
```

[9]:

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

def calculate_sum_of_elements(nums):
    return sum(nums)

# Input prompt for the array of integers
def input_array():
    array = input("Enter the array of integers (each element separated by␣
 ↪space): ").split()
    return [int(element) for element in array]

# Get the array of integers from the user
integer_array = input_array()

# Calculate the sum of elements
sum_of_elements = calculate_sum_of_elements(integer_array)

# Display the result
print(f"The sum of elements in the array is: {sum_of_elements}")
```

```
Enter the array of integers (each element separated by space):  1 2 3 4 5

The sum of elements in the array is: 15
```

[10]:

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

def find_max_element(nums):
    if not nums:
        return None  # Return None for an empty array

    max_element = nums[0]

    for num in nums[1:]:
        if num > max_element:
            max_element = num

    return max_element

# Input prompt for the array of integers
def input_array():
    array = input("Enter the array of integers (each element separated by␣
 ↪space): ").split()
    return [int(element) for element in array]

# Get the array of integers from the user
integer_array = input_array()

# Find the maximum element
max_element = find_max_element(integer_array)

# Display the result
if max_element is not None:
    print(f"The maximum element in the array is: {max_element}")
else:
    print("The array is empty.")
```

```
Enter the array of integers (each element separated by space):  3 7 2 9 4 1

The maximum element in the array is: 9
```

[11]:

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

def find_max_element(nums):
    if not nums:
        return None  # Return None for an empty array

    max_element = nums[0]

    for num in nums[1:]:
        if num > max_element:
            max_element = num

    return max_element

# Input prompt for the array of integers
def input_array():
    array = input("Enter the array of integers (each element separated by␣
 ↪space): ").split()
    return [int(element) for element in array]
```

```
# Get the array of integers from the user
integer_array = input_array()

# Find the maximum element
max_element = find_max_element(integer_array)

# Display the result
if max_element is not None:
    print(f"The maximum element in the array is: {max_element}")
else:
    print("The array is empty.")
```

```
Enter the array of integers (each element separated by space):  5 3 8 2 7 4

The maximum element in the array is: 8
```

[12]:

```
#Problem 28 Calculate the factorial of a given number.

def calculate_factorial(n):
    if n < 0:
        return None  # Factorial is undefined for negative numbers
    elif n == 0 or n == 1:
        return 1  # Factorial of 0 and 1 is 1

    factorial_result = 1
    for i in range(2, n + 1):
        factorial_result *= i

    return factorial_result

# Input prompt for the number
number = int(input("Enter a non-negative integer to calculate its factorial: "))

# Calculate the factorial
result = calculate_factorial(number)

# Display the result
if result is not None:
    print(f"The factorial of {number} is: {result}")
else:
    print("Factorial is undefined for negative numbers.")
```

```
Enter a non-negative integer to calculate its factorial:  5

The factorial of 5 is: 120
```

[13]:

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

```python
import math

def is_prime(number):
    if number <= 1:
        return False  # Numbers less than or equal to 1 are not prime

    # Check for divisors up to the square root of the number
    for i in range(2, int(math.sqrt(number)) + 1):
        if number % i == 0:
            return False  # Number is not prime if it has a divisor

    return True  # Number is prime if no divisors are found

# Input prompt for the number
number = int(input("Enter a positive integer to check if it's prime: "))

# Check if the number is prime
if is_prime(number):
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
```

```
Enter a positive integer to check if it's prime:  7
```

```
7 is a prime number.
```

[19]:

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

def generate_fibonacci_series(n):
    fibonacci_series = [0, 1]

    while len(fibonacci_series) < n:
        next_number = fibonacci_series[-1] + fibonacci_series[-2]
        fibonacci_series.append(next_number)

    return fibonacci_series

# Input prompt for the value of n
n = int(input("Enter the value of n for generating Fibonacci series: "))

# Generate the Fibonacci series up to the nth Fibonacci number
fibonacci_series = generate_fibonacci_series(n)

# Display the result
print(f"Fibonacci series up to the {n}th Fibonacci number:", fibonacci_series)
```

```
Enter the value of n for generating Fibonacci series:  8
```

```
Fibonacci series up to the 8th Fibonacci number: [0, 1, 1, 2, 3, 5, 8, 13]
```

[20]:

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

def power(base, exponent):
    if exponent == 0:
        return 1
    else:
        return base * power(base, exponent - 1)

# Input prompts for the base and exponent
base = float(input("Enter the base: "))
exponent = int(input("Enter the exponent (a non-negative integer): "))

# Check if the exponent is non-negative
if exponent < 0:
    print("Exponent should be a non-negative integer.")
else:
    # Calculate the power using recursion
    result = power(base, exponent)
    print(f"{base} to the power of {exponent} is: {result}")
```

```
Enter the base:  3
Enter the exponent (a non-negative integer):  4
```

```
3.0 to the power of 4 is: 81.0
```

[21]:

```
#Problem 32: Reverse a given string.

def reverse_string(input_string):
    return input_string[::-1]

# Input prompt for the string
input_string = input("Enter a string to reverse: ")

# Reverse the string
reversed_string = reverse_string(input_string)

# Display the result
print("Original String:", input_string)
print("Reversed String:", reversed_string)
```

```
Enter a string to reverse:  hello
```

```
Original String: hello
Reversed String: olleh
```

[ ]:

```

```