dec-24-dsa-1-assignment-14

August 9, 2024

[2]:
```
'''Dec~24_DSA_1_Assignment_14'''
```

[2]:
```
'Dec^24_DSA_1_Assignment_14'
```

[20]:
```
'''Problem 1: Reverse a singly linked list.
Input: 1 -> 2 -> 3 -> 4 -> 5
Output: 5 -> 4 -> 3 -> 2 -> 1'''
#Ceate the linked list first
class Node:
    def __init__(self,data=None,next=None):
        self.data = data
        self.next = next
    def setData(self,data):
        self.data=data
    def getData(self):
        return self.data
    def setNext(self,next):
        self.next=next
    def getNext(self):
        return self.next
#Create a linked list>>Collection of link Nodes
head=Node(1)
node2=Node(2)
node3=Node(3)
node4=Node(4)
node5=Node(5)
#Create the linkage
head.setNext(node2)
node2.setNext(node3)
node3.setNext(node4)
node4.setNext(node5)

#Traverse through the linked list
def traverse(head):
    temp=head
    while(temp):
        print(temp.getData(),end="->")
        temp=temp.getNext()
#Reverse the Linked_List: (1->2->3->4->5->)
def reverse_linked_list(head):
    current = head
    previous = None

    while current is not None:
        next_node = current.next
        current.next = previous
        previous = current
        current = next_node

    head = previous
    return head
print(traverse(head))
#Reverse the linked list
reversed_head = reverse_linked_list(head)
#Traverse through the reversed linked list
traverse(reversed_head)
```

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

[13]:

```
'''
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 Node:
    def __init__(self,data=None,next=None):
        self.data=data
        self.next=next
def merged_lists(head1,head2):
    dummy=Node()
    tail=dummy
    while head1 and head2:
        if head1.data<=head2.data:
            tail.next=head1
            head1=head1.next
        else:
            tail.next=head2
            head2=head2.next
        tail=tail.next
    if head1:
        tail.next=head1
    else:
        tail.next=head2
    return dummy.next
head1=Node(1)
head1.next=Node(3)
head1.next.next=Node(5)

head2=Node(2)
head2.next=Node(4)
head2.next.next=Node(6)
merged_head=merged_lists(head1,head2)
#Traverse through the merged list:
temp=merged_head
while temp:
    print(temp.data,end="->")
    temp=temp.next
print("None")
```

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

[22]:

```
'''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'''
class Node:
    def __init__(self,data=None,next=None):
        self.data=data
        self.next=next
def RemoveNthFromEnd(head,n):
    dummy=Node(0)
    dummy.next=head
    fast=slow=dummy

    for i in range(n):
        fast=fast.next

    while fast.next:
        fast=fast.next
        slow=slow.next

    slow.next=slow.next.next
    return dummy.next
head=Node(1)
node2=Node(2)
node3=Node(3)
node4=Node(4)
node5=Node(5)

head.next=node2
node2.next=node3
```

```
node3.next=node4
node4.next=node5
#Traverse through the LL
def traverse(head):
    temp=head
    while temp:
        print(temp.data,end="->")
        temp=temp.next
print(traverse(head))
traverse(RemoveNthFromEnd(head,2))
```

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

[1]:

```
'''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 Node:
    def __init__(self,data=None,next=None):
        self.data=data
        self.next=next
def FindIntersectingNode(head1,head2):
    if not head1 or not head2:
        return None
    #Find the length
    curr1,curr2=head1,head2
    len1,len2=0,0
    while curr1:
        len1+=1
        curr1=curr1.next
    while curr2:
        len2+=1
        curr2=curr2.next
    #adjusting the head of longerLL
    while len1>len2:
        head1=head1.next
        len1-=1
    while len1<len2:
        head2=head2.next
        len2-=1
    #move both pointer untill they meet at intersection
    while head1!=head2:
        head1=head1.next
        head2=head2.next
    return head1
#If wants to know the link or view
```

```
def traverse(head):
    temp=head
    while temp:
        print(temp.data,end="->")
        temp=temp.next
#creating the collection of Nodes for LL1
head1=Node(1)
node2=Node(2)
node3=Node(3)
node4=Node(4)
#Creating the collection of Nodes for LL2
head2=Node(9)
node_2=Node(8)
#Creating the linkage for LL1
head1.next=node2
node2.next=node3
node3.next=node4
#Creating the linkage for LL2
head2.next=node_2
node_2.next=node3

#Traverse the both LL
print("LL1",traverse(head1))
print("LL2",traverse(head2))
#Driver code
Intersected_node=FindIntersectingNode(head1,head2)
print(f"Node with value: {Intersected_node.data}, is Common or intersecting␣
  ↪Node")
```

```
1->2->3->4->LL1 None
9->8->3->4->LL2 None
Node with value: 3, is Common or intersecting Node
```

[13]:

```
'''Problem 5: Remove duplicates from a sorted linked list.
Input: 1 -> 1 -> 2 -> 3 -> 3
Output: 1 -> 2 -> 3'''
class Node:
    def __init__(self,data=None,next=None):
        self.data=data
        self.next=next
def RemoveDup(head):
    if not head:
        return head
    #initialise the current
    current=head
    #Check if there is any other element in the LL
    while current.next:
        if current.data==current.next.data:
            current.next=current.next.next
        else:
            current=current.next

    return head
#Traverse the linked list
def traverse(head):
    temp=head
    while temp:
        print(temp.data,end="->")
        temp=temp.next

#Create the collection of node or LL
head=Node(1)
node2=Node(1)
node3=Node(2)
node4=Node(3)
node5=Node(3)
#Creating the linkage:
head.next=node2
node2.next=node3
node3.next=node4
node4.next=node5

#Traverse the LL
print(traverse(head))
#remove the duplicate
removed_head=RemoveDup(head)
traverse(removed_head)
```

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

[20]:

```
'''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)'''
class Node:
    def __init__(self, data=None, next=None):
        self.data = data
        self.next = next

def addTwoNumbers(head1, head2):
    dummy = Node()
    current = dummy
    carry = 0
    while head1 or head2 or carry:
        sum = carry

        if head1:
            sum += head1.data
            head1 = head1.next

        if head2:
            sum += head2.data
            head2 = head2.next

        carry = sum // 10
        current.next = Node(sum % 10)
        current = current.next

    return dummy.next

# Creating the first linked list
head1 = Node(2)
node2 = Node(4)
node3 = Node(3)
head1.next = node2
node2.next = node3

# Creating the second linked list
head2 = Node(5)
node4 = Node(6)
node5 = Node(4)
head2.next = node4
node4.next = node5

# Calling the addTwoNumbers function
result = addTwoNumbers(head1, head2)

# Printing the sum as a linked list
current = result
while current:
    print(current.data, end=" -> ")
    current = current.next
print("None")
```

```
7 -> 0 -> 8 -> None
```

[6]:

```
'''
Problem 7: Swap nodes in pairs in a linked list.
Input: 1 -> 2 -> 3 -> 4
```

```python
Output: 2 -> 1 -> 4 -> 3
'''
class Node:
    def __init__(self, data=0, next=None):
        self.data = data
        self.next = next

def SwapPairs(head):
    dummy=Node(0)
    dummy.next=head
    current=dummy

    #check is there atleast two element after dummy
    while current.next and current.next.next:
        first=current.next
        second=current.next.next

        #swappin the pairs
        first.next=second.next
        second.next=first
        current.next=second

        current=current.next.next
    return dummy.next
def travers(head):
    temp=head
    while temp:
        print(temp.data,end="->")
        temp=temp.next

#Create the collection of nodes
head=Node(1)
node2=Node(2)
node3=Node(3)
node4=Node(4)
#Create linkage
head.next=node2
node2.next=node3
node3.next=node4
print(travers(head))
Swaped_head=SwapPairs(head)
print(travers(Swaped_head))
```

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

[11]:

```
'''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 Node:
    def __init__(self, data=0, next=None):
        self.data = data
        self.next = next

def reverseKGroup(head, k):
    if not head or k == 1:
        return head

    dummy = Node(0)
    dummy.next = head
    prev = dummy
    curr = head
    count = 0

    while curr:
        count += 1
        if count % k == 0:
            prev = reverse(prev, curr.next)
            curr = prev.next
        else:
            curr = curr.next

    return dummy.next

def reverse(prev, next):
    last = prev.next
    curr = last.next

    while curr != next:
        last.next = curr.next
        curr.next = prev.next
        prev.next = curr
        curr = last.next

    return last
def travers(head):
    temp=head
    while temp:
        print(temp.data,end="->")
        temp=temp.next

#Collection of nodes:
head = Node(1)
```

```
node2= Node(2)
node3 = Node(3)
node4 = Node(4)
node5 = Node(5)
#Linkage
head.next=node2
node2.next=node3
node3.next=node4
node4.next=node5
#Traverse the Original LL
print(travers(head))

#Driver code
k = 3
reversed_head = reverseKGroup(head, k)
#Printing the reversed linked list
print(travers(reversed_head))
```

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

[12]:

```
'''Problem 9: Determine if a linked list is a palindrome.
Input: 1 -> 2 -> 2 -> 1
Output: True'''
class Node:
    def __init__(self, data=0, next=None):
        self.data = data
        self.next = next

def isPalindrome(head):
    # Store the values of each node in a list
    values = []
    current = head
    while current:
        values.append(current.data)
        current = current.next

    # Use two pointers to compare values
    left = 0
    right = len(values) - 1
    while left < right:
        if values[left] != values[right]:
            return False
        left += 1
        right -= 1

    return True
```

```
def travers(head):
    temp=head
    while temp:
        print(temp.data,end="->")
        temp=temp.next
#Collection of nodes:
head = Node(1)
node2= Node(2)
node3 = Node(2)
node4 = Node(1)
#Linkage
head.next=node2
node2.next=node3
node3.next=node4
print(travers(head))
isPalindrome(head)
```

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

[12]:

```
True
```

[21]:

```
'''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 Node:
    def __init__(self, data=None, next=None):
        self.data = data
        self.next = next

def rotateRight(head, k):
    if not head or not head.next or k == 0:
        return head

    # Find the length of the linked list
    length = 1
    tail = head
    while tail.next:
        tail = tail.next
        length += 1

    # Calculate the actual rotation index
    rotation_index = k % length
    if rotation_index == 0:
        return head

    # Find the node before the new tail
    new_tail_index = length - rotation_index - 1
    new_tail = head
    for _ in range(new_tail_index):
        new_tail = new_tail.next

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

    return new_head
def travers(head):
    temp=head
    while temp:
        print(temp.data,end="->")
        temp=temp.next
#Nodes and linkage together
head = Node(1)
head.next = Node(2)
head.next.next = Node(3)
head.next.next.next = Node(4)
head.next.next.next.next = Node(5)
#Check the input LL
print(travers(head))
#Driver code:
k = 2
rotated_head = rotateRight(head, k)
print(travers(rotated_head))
```

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

[19]:

```
'''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, data):
        self.data = data
        self.prev = None
        self.next = None
        self.child = None

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

    curr = head
    while curr:
        if curr.child:
            next_node = curr.next
            child = curr.child
            curr.next = child
            child.prev = curr
            curr.child = None

            while child.next:
                child = child.next
            child.next = next_node

            if next_node:
                next_node.prev = child
        curr = curr.next

    return head
# Print the flattened list
def traverse(head):
    temp = head
    while temp:
        print(temp.data, end="->")
        temp = temp.next

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

# Connect the nodes
head.next = node2
node2.prev = head
node2.next = node3
node3.prev = node2
node3.next = node4
node4.prev = node3
node4.next = node5
```

```
node5.prev = node4
node5.next = node6
node6.prev = node5
node6.next = node7
node7.prev = node6
node7.next = node8
node8.prev = node7
node8.next = node9
node9.prev = node8
node9.next = node10
node10.prev = node9
node10.next = node11
node11.prev = node10
node11.next = node12
node12.prev = node11
node12.next = node13
node13.prev = node12
# Flatten the list
flatten_head = flatten_list(head)
traverse(flatten_head)
```

```
1->2->3->4->5->6->7->8->9->10->11->12->13->
```

[36]:

```
'''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 Node:
    def __init__(self, data):
        self.data = data
        self.next = None

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

    odd_head = head
    even_head = head.next

    odd_ptr = odd_head
    even_ptr = even_head

    while even_ptr and even_ptr.next:
        odd_ptr.next = even_ptr.next
        odd_ptr = odd_ptr.next

        even_ptr.next = odd_ptr.next
        even_ptr = even_ptr.next

    odd_ptr.next = even_head

    return odd_head
def traverse(head):
    temp = head
    while temp:
        print(temp.data, end="->")
        temp = temp.next

# Create the nodes
head = Node(1)
node2 = Node(2)
node3 = Node(3)
node4 = Node(4)
node5 = Node(5)

# Connect the nodes
head.next = node2
node2.next = node3
node3.next = node4
node4.next = node5
print(traverse(head))
# Rearrange the list
rearranged_head = rearrange_list(head)
# Print the rearranged list
traverse(rearranged_head)
```

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

[48]:

```
'''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 Node:
    def __init__(self, data=0, next=None):
        self.data=data
        self.next=next

def add_One(head):
    # Step 1: Reverse the linked list
    def reverseLinkedList(node):
        prev = None
        while node:
            temp = node.next
            node.next = prev
            prev = node
            node = temp
        return prev

    reversed_head = reverseLinkedList(head)

    #Step 2: Traverse the reversed linked list and add one
    current = reversed_head
    carry = 1
    while current and carry:
        current.data += carry
        carry = current.data // 10
        current.data %= 10
        if carry and not current.next:
            current.next =Node(0)
        current = current.next

    # Step 3: Reverse the linked list again
    result = reverseLinkedList(reversed_head)

    return result
def traverse(head):
    temp=head
    while temp:
        print(temp.data, end=" -> ")
        temp=temp.next

# Example usage:
# Input: 1 -> 2 -> 3
# Output: 1 -> 2 -> 4
head = Node(1, Node(2, Node(3)))
print(traverse(head))

result = add_One(head)
# Print the result
print(traverse(result))
```

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

[6]:

```
'''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 SearchInsert(arr,target):
    left=0
    right=len(arr)-1
    while left<=right:
        mid=left+(right-left)//2
        if arr[mid]==target:
            return mid
        elif arr[mid]<target:
            left+=1
        else:
            right-=1
    return left
arr=[1,3,5,6]
target=5
result=SearchInsert(arr,target)
print(result)
```

```
2
```

[15]:

```
'''Problem 15: Find the minimum element in a rotated sorted array.
Input: [4, 5, 6, 7, 0, 1, 2]
Output: 0'''
def findMin(arr):
    left=0
    right=len(arr)-1

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

        if arr[mid]>arr[right]:
            left=mid+1
        else:
            right=mid
    return arr[left]
arr = [4, 5, 6, 7, 0, 1, 2]
result=findMin(arr)
print("Output:",result)
```

```
Output: 0
```

[22]:

```
'''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(arr, target):
    left = 0
    right = len(arr)-1

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

        if arr[mid]==target:
            return mid

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

    return -1

arr = [4, 5, 6, 7, 0, 1, 2]
target = 0
result = search(arr, target)
print("Output:",result)
```

```
Output: 4
```

[20]:

```
'''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 peak_index(arr):
    left=0
    right=len(arr)-1

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

        if arr[mid]<arr[mid+1]:
            left=mid
        else:
            right=mid+1
    return left
nums = [1, 2, 3, 1]
peak_index = find_peak(nums)
print("Output:",peak_index,"(index of peak element)")
```

```
Output: 2 (index of peak element)
```

[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 CountNegative(grid):
    count=0
    for row in grid:
        for num in row:
            if num<0:
                count+=1
    return count
grid = [[4, 3, 2, -1], [3, 2, 1, -1], [1, 1, -1, -2], [-1, -1, -2, -3]]
Output=CountNegative(grid)
print("Count of Negative numbers:",Output)
```

```
Count of Negative numbers: 8
```

[18]:

```
'''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 FindTarget(matrix,target):
    for rows in matrix:
        for i in rows:
            if i==target:
                return True
    return False
matrix=[[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 60]]
target = 7
result=FindTarget(matrix,target)
print("Output:",result)
```

```
Output: True
```

[20]:

```
'''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'''
#Using Python Library
import numpy as np
num1=[1,3]
num2=[2]
combined_nums=num1+num2
np.median(combined_nums)
```

[20]: 2.0

[25]:
```
#Mathematical definition
num1,num2=[1,3],[2]
joint_arr=sorted(num1+num2)
m=len(joint_arr)//2
if len(joint_arr)%2==0:
    median=float((joint_arr[m+1]+joint_arr[m])/2)
else:
    median=float(joint_arr[m])
print("Median:",median)
```

```
Median: 2.0
```

[21]:
```
"""
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 findSmallest_Letter(letters,target):
    for letter in letters:
            if letter>target:
                return letter
    return letters[0]
letters=['c', 'f', 'j']
target="a"
Output=findSmallest_Letter(letters,target)
print("Output:",Output)
```

```
Output: c
```

[22]:
```
'''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 = 0
    mid = 0
    high = 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

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

```
[0, 0, 1, 1, 2, 2]
```

[26]:

```
'''Problem 23: Find the kth largest element in an unsorted array.
Input: nums = [3, 2, 1, 5, 6, 4], k = 2
Output: 5
'''
def find_kth_largest(nums, k):
    nums.sort(reverse=True)
    return nums[k-1]

# Example usage
nums = [3, 2, 1, 5, 6, 4]
k = 2
result = find_kth_largest(nums, k)
print(result)
```

```
5
```

[27]:

```
'''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 reorder_array(nums):
    for i in range(1, len(nums), 2):
        if nums[i] < nums[i-1]:
            nums[i], nums[i-1] = nums[i-1], nums[i]
        if i+1 < len(nums) and nums[i] < nums[i+1]:
            nums[i], nums[i+1] = nums[i+1], nums[i]
    return nums

# Example usage
nums = [3, 5, 2, 1, 6, 4]
result = reorder_array(nums)
print(result)
```

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

[36]:

```
'''Problem 25: Given an array of integers, calculate the sum of all its␣
 ↪elements.
Input: [1, 2, 3, 4, 5]
Output: 15
'''
def find_Sum(arr):
    if len(arr)==0:
        return 0
    else:
        return arr[0]+find_Sum(arr[1:])
arr=[1, 2, 3, 4, 5]
print("Sum of the array:",find_Sum(arr))
```

```
Sum of the array: 15
```

[44]:

```
'''Problem 26: Find the maximum element in an array of integers.
Input: [3, 7, 2, 9, 4, 1]
Output: 9'''
def MaxFind(arr):
    max_element=arr[0]
    for num in arr:
        if num>max_element:
            max_element=num
    return max_element
arr=[3, 7, 2, 9, 4, 1]
print("Maximum element:",MaxFind(arr))
```

```
Maximum element: 9
```

[52]:

```
'''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 Search(arr,target):
    for num in range(len(arr)):
        if target == arr[num]:
            print("Target is at index:",num)
            return
    return -1
target=8
arr=[5, 3, 8, 2, 7, 4]
Search(arr,target)
```

```
Target is at index: 2
```

[54]:

```python
'''Problem 28 Calculate the factorial of a given number.
Input: 5
Output: 120 (as 5! = 5 * 4 * 3 * 2 * 1 = 120)
'''
def fact(n):
    if n<=1:
        return n
    else:
        return n*fact(n-1)
n=int(input("Enter the number:"))
print("Factorial of a given number is:",fact(n))
```

```
Enter the number: 5
```

```
Factorial of a given number is: 120
```

[55]:

```python
'''Problem 29: Check if a given number is a prime number.
Input: 7
Output: True
'''
def check_primes(n):
    if n<=1:
        return False
    for i in range(2,int(n**0.5)+1):
        if n%i==0:
            return False
    return True
n=int(input("Enter the value for n:"))
check_primes(n)
```

```
Enter the value for n: 7
```

[55]:

```
True
```

[61]:

```python
'''Problem 30: Generate the Fibonacci series up to a given number n.
Input: 8
Output: [0, 1, 1, 2, 3, 5, 8, 13]'''
def generate_fibonacci(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

# Example usage
n = 8
fibonacci_result = generate_fibonacci(n)
print(fibonacci_result)
```

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

[65]:

```
'''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 powerfind(n,p):
    if n!=0 and p==0:
        return 1
    else:
        return n*powerfind(n,p-1)
print("Output:",powerfind(3,4))
```

```
Output: 81
```

[81]:

```
'''Problem 32: Reverse a given string.
Input: "hello"
Output: "olleh"
'''
def reverse(string):
    L=list(string) #As strings are immutable, convert to list
    left=0
    right=len(L)-1
    while left<=right:
        L[left],L[right]=L[right],L[left]
        left+=1
        right-=1
    reversed_str = "".join(L)
    return reversed_str
string="hello" #Input
print("Output:",reverse(string)) #Output
```

```
Output: olleh
```

[ ]: