**Assignment: 3**

1. Compute the sum of digits of a number using recursion

class RecursionProblems:

@staticmethod

def sum\_of\_digits(n):

if n == 0:

return 0

else:

return n % 10 + RecursionProblems.sum\_of\_digits(n // 10)

2. Compute the product of digits of a number using recursion

class RecursionProblems:

@staticmethod

def product\_of\_digits(n):

if n == 0:

return 0

elif n % 10 == n:

return n

else:

return (n % 10) \* RecursionProblems.product\_of\_digits(n // 10)

3. Given two numbers x and y, find the product using recursion

class RecursionProblems:

@staticmethod

def recursive\_multiply(x, y):

if y == 0:

return 0

elif y > 0:

return x + RecursionProblems.recursive\_multiply(x, y - 1)

else:

return -RecursionProblems.recursive\_multiply(x, -y)

4. Find the value of 'a' raised to the power 'b' using recursion

class RecursionProblems:

@staticmethod

def power(a, b):

if b == 0:

return 1

else:

return a \* RecursionProblems.power(a, b - 1)

5. Convert decimal to binary using recursion

class RecursionProblems:

@staticmethod

def decimal\_to\_binary(n):

if n == 0:

return ''

else:

return RecursionProblems.decimal\_to\_binary(n // 2) + str(n % 2)

# Example usage

number = 10

binary\_representation = RecursionProblems.decimal\_to\_binary(number)

binary\_representation = binary\_representation if binary\_representation else '0'

print(f"Binary representation of {number} is: {binary\_representation}")

6. For any positive value of N, print 1 to N without using for or while loops via recursion (Reverse Countdown)

class RecursionProblems:

@staticmethod

def print\_1\_to\_n(n):

if n > 0:

RecursionProblems.print\_1\_to\_n(n - 1)

print(n, end=' ')

7. For any positive value of N, print N to 1 without using for or while loops via recursion (Countdown)

class RecursionProblems:

@staticmethod

def print\_n\_to\_1(n):

if n > 0:

print(n, end=' ')

RecursionProblems.print\_n\_to\_1(n - 1)

8. Reverse a number using recursion

class RecursionProblems:

@staticmethod

def reverse\_number(n, result=0):

if n == 0:

return result

else:

return RecursionProblems.reverse\_number(n // 10, result \* 10 + n % 10)

9. Find the length of a string using recursion

class RecursionProblems:

@staticmethod

def length\_of\_string(s):

if s == '':

return 0

else:

return 1 + RecursionProblems.length\_of\_string(s[1:])

10. Reverse a string using recursion

class RecursionProblems:

@staticmethod

def reverse\_string(s):

if len(s) == 0:

return ''

else:

return s[-1] + RecursionProblems.reverse\_string(s[:-1])

11. Check if a string is a palindrome using recursion

class RecursionProblems:

@staticmethod

def is\_palindrome(s):

if len(s) <= 1:

return True

if s[0] != s[-1]:

return False

return RecursionProblems.is\_palindrome(s[1:-1])

12. Compute sum of first N natural numbers using recursion

class RecursionProblems:

@staticmethod

def sum\_natural\_numbers(n):

if n == 0:

return 0

else:

return n + RecursionProblems.sum\_natural\_numbers(n - 1)

13. Pow(x, n) using recursion (LeetCode Problem)

class RecursionProblems:

@staticmethod

def my\_pow(x, n):

if n == 0:

return 1

elif n < 0:

return 1 / RecursionProblems.my\_pow(x, -n)

elif n % 2:

return x \* RecursionProblems.my\_pow(x, n - 1)

else:

half = RecursionProblems.my\_pow(x, n // 2)

return half \* half

14. Power of Two (LeetCode Problem)

class RecursionProblems:

@staticmethod

def is\_power\_of\_two(n):

if n <= 0:

return False

if n == 1:

return True

if n % 2 != 0:

return False

return RecursionProblems.is\_power\_of\_two(n // 2)

15. Power of Three (LeetCode Problem)

class RecursionProblems:

@staticmethod

def is\_power\_of\_three(n):

if n <= 0:

return False

if n == 1:

return True

if n % 3 != 0:

return False

return RecursionProblems.is\_power\_of\_three(n // 3)

16. Power of Four (LeetCode Problem)

class RecursionProblems:

@staticmethod

def is\_power\_of\_four(n):

if n <= 0:

return False

if n == 1:

return True

if n % 4 != 0:

return False

return RecursionProblems.is\_power\_of\_four(n // 4)

17. Find the nth number in a Fibonacci Series using Recursion (LeetCode Problem)

class RecursionProblems:

@staticmethod

def fibonacci(n):

if n <= 1:

return n

else:

return RecursionProblems.fibonacci(n - 1) + RecursionProblems.fibonacci(n - 2)

18. Geekonacci Number (GeeksforGeeks Problem)

class RecursionProblems:

@staticmethod

def geekonacci(a, b, c, n):

if n == 1:

return a

elif n == 2:

return b

elif n == 3:

return c

else:

return RecursionProblems.geekonacci(a, b, c, n - 1) + RecursionProblems.geekonacci(a, b, c, n - 2) + RecursionProblems.geekonacci(a, b, c, n - 3)

19. Special Fibonacci (CodeChef Problem)

class RecursionProblems:

@staticmethod

def special\_fibonacci(a, b, n):

if n == 1:

return a

elif n == 2:

return b

else:

return RecursionProblems.special\_fibonacci(a, b, n - 1) ^ RecursionProblems.special\_fibonacci(a, b, n - 2)

20. Count Number of Zeros in a Number using recursion

class RecursionProblems:

@staticmethod

def count\_zeros(n):

if n == 0:

return 1

if n < 10:

return 0

if n % 10 == 0:

return 1 + RecursionProblems.count\_zeros(n // 10)

else:

return RecursionProblems.count\_zeros(n // 10)

21. Number of Steps to Reduce a Number to Zero (LeetCode Problem) using recursion

class RecursionProblems:

@staticmethod

def number\_of\_steps(num):

if num == 0:

return 0

if num % 2 == 0:

return 1 + RecursionProblems.number\_of\_steps(num // 2)

else:

return 1 + RecursionProblems.number\_of\_steps(num - 1)

22. Find sum of Array elements using recursion

class RecursionProblems:

@staticmethod

def sum\_array(arr, n):

if n == 0:

return 0

else:

return arr[n - 1] + RecursionProblems.sum\_array(arr, n - 1)

23. Find mean of Array elements using recursion

class RecursionProblems:

@staticmethod

def mean\_array(arr, n):

if n == 0:

return 0

else:

total\_sum = RecursionProblems.sum\_array(arr, n)

return total\_sum / n

24. Find maximum and minimum of Array elements using recursion

class RecursionProblems:

@staticmethod

def find\_max(arr, n):

if n == 1:

return arr[0]

else:

return max(arr[n - 1], RecursionProblems.find\_max(arr, n - 1))

@staticmethod

def find\_min(arr, n):

if n == 1:

return arr[0]

else:

return min(arr[n - 1], RecursionProblems.find\_min(arr, n - 1))

25. Compute the factorial of a number using recursion

class RecursionProblems:

@staticmethod

def factorial(n):

if n == 0:

return 1

else:

return n \* RecursionProblems.factorial(n - 1)

26. Sum Triangle (GeeksforGeeks Problem)

class RecursionProblems:

@staticmethod

def sum\_triangle(arr):

if len(arr) < 1:

return

temp = []

for i in range(0, len(arr) - 1):

temp.append(arr[i] + arr[i + 1])

RecursionProblems.sum\_triangle(temp)

print(arr)

27. Perform binary search using recursion (LeetCode Problem)

class RecursionProblems:

@staticmethod

def binary\_search(arr, low, high, target):

if high >= low:

mid = (high + low) // 2

if arr[mid] == target:

return mid

elif arr[mid] > target:

return RecursionProblems.binary\_search(arr, low, mid - 1, target)

else:

return RecursionProblems.binary\_search(arr, mid + 1, high, target)

else:

return -1

28. Perform Bubble Sort using Recursion

class RecursionProblems:

@staticmethod

def bubble\_sort(arr, n=None):

if n is None:

n = len(arr)

if n == 1:

return

for i in range(n - 1):

if arr[i] > arr[i + 1]:

arr[i], arr[i + 1] = arr[i + 1], arr[i]

RecursionProblems.bubble\_sort(arr, n - 1)

29. Perform Insertion Sort using Recursion

class RecursionProblems:

@staticmethod

def insertion\_sort(arr, n=None):

if n is None:

n = len(arr)

if n <= 1:

return

RecursionProblems.insertion\_sort(arr, n - 1)

last = arr[n - 1]

j = n - 2

while j >= 0 and arr[j] > last:

arr[j + 1] = arr[j]

j = j - 1

arr[j + 1] = last

30. Reverse a Linked List using recursion

class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

class RecursionProblems:

@staticmethod

def reverse\_linked\_list(head):

if head is None or head.next is None:

return head

rest = RecursionProblems.reverse\_linked\_list(head.next)

head.next.next = head

head.next = None

return rest

# Helper function to print linked list

def print\_linked\_list(node):

while node:

print(node.val, end=" ")

node = node.next

print()

31. Merge two sorted Linked Lists using recursion

class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

class RecursionProblems:

@staticmethod

def merge\_two\_sorted\_lists(l1, l2):

if not l1:

return l2

if not l2:

return l1

if l1.val < l2.val:

l1.next = RecursionProblems.merge\_two\_sorted\_lists(l1.next, l2)

return l1

else:

l2.next = RecursionProblems.merge\_two\_sorted\_lists(l1, l2.next)

return l2

# Helper function to print linked list

def print\_linked\_list(node):

while node:

print(node.val, end=" ")

node = node.next

print()

32. Print all leaf nodes of a Binary Search Tree using recursion

class TreeNode:

def \_\_init\_\_(self, val=0, left=None, right=None):

self.val = val

self.left = left

self.right = right

class RecursionProblems:

@staticmethod

def print\_leaf\_nodes(root):

if root is None:

return

if root.left is None and root.right is None:

print(root.val, end=" ")

RecursionProblems.print\_leaf\_nodes(root.left)

RecursionProblems.print\_leaf\_nodes(root.right)

33. Solve the Towers of Hanoi problem

class RecursionProblems:

@staticmethod

def towers\_of\_hanoi(n, source, target, auxiliary):

if n == 1:

print(f"Move disk 1 from {source} to {target}")

return

RecursionProblems.towers\_of\_hanoi(n - 1, source, auxiliary, target)

print(f"Move disk {n} from {source} to {target}")

RecursionProblems.towers\_of\_hanoi(n - 1, auxiliary, target, source)

34. Compute the GCD of two numbers using the Euclidean Algorithm

class RecursionProblems:

@staticmethod

def gcd(a, b):

if b == 0:

return a

else:

return RecursionProblems.gcd(b, a % b)