DAA RECURSION ASSIGNMENT 🌞

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

CODE:

```
def sum_of_digits_recursive(number):
    if number < 10:
        return number
    else:
        last_digit = number % 10
        remaining_digits = number // 10
        return last_digit + sum_of_digits_recursive(remaining_digits)

# Example usage:
input_number = 12345
result = sum_of_digits_recursive(input_number)
print(f"Sum of digits of {input_number} is {result}")
```

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

CODE:

```
def product_of_digits_recursive(number):
    if number < 10:
        return number
```

```
    else:
        last_digit = number % 10
        remaining_digits = number // 10
        return last_digit * product_of_digits_recursive(remaining_digits)

# Example usage:
input_number = 12345
result = product_of_digits_recursive(input_number)
print(f"Product of digits of {input_number} is {result}")
```

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

CODE:

```
def product_recursive(x, y):
    if x == 0 or y == 0:
        return 0
    else:
        return x + product_recursive(x, y - 1)

# Example usage:
x = 5
y = 3
result = product_recursive(x, y)
print(f"Product of {x} and {y} is {result}")
```

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

CODE:

```
def power_recursive(a, b):
    if b == 0:
        return 1
    else:
        return a * power_recursive(a, b - 1)

# Example usage:
a = 2
b = 3
result = power_recursive(a, b)
print(f"{a} raised to the power of {b} is {result}")
```

5)Convert decimal to binary using recursion

CODE:

```
def decimal_to_binary_recursive(number):
    if number == 0:
        return ""
    else:
        remainder = number % 2
        quotient = number // 2
        return decimal_to_binary_recursive(quotient) + str(remainder)

# Example usage:
decimal_number = 10
binary_result = decimal_to_binary_recursive(decimal_number)
print(f"Binary representation of {decimal_number} is {binary_result}")
```

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

CODE:

```
def print_reverse_countdown(N):
    if N == 0:
        return
    else:
        print(N)
        print_reverse_countdown(N - 1)

# Example usage:
N = 5
print_reverse_countdown(N)
```

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

CODE:

```
def print_countdown(N):
    if N == 0:
        return
    else:
        print(N)
        print_countdown(N - 1)

# Example usage:
N = 5
print_countdown(N)
```

8)Reverse a number using recursion

CODE:

```
def reverse_number_recursive(number):
```

```
    if number < 10:
        return number
    else:
        last_digit = number % 10
        remaining_digits = number // 10
        return int(str(last_digit) + str(reverse_number_recursive(remaining_digits)))

# Example usage:
input_number = 12345
reversed_result = reverse_number_recursive(input_number)
print(f"Reversed number of {input_number} is {reversed_result}")
```

9)Find length of string using recursion.

CODE:

```
def string_length_recursive(s):
    if not s:
        return 0
    else:
        return 1 + string_length_recursive(s[1:])

# Example usage:
input_string = "Hello, World!"
length = string_length_recursive(input_string)
print(f"Length of '{input_string}' is {length}")
```

10)Reverse a string using recursion.

CODE:

```
def reverse_string_recursive(s):
    if not s:
        return ""
    else:
        return s[-1] + reverse_string_recursive(s[:-1])

# Example usage:
input_string = "Hello, World!"
reversed_result = reverse_string_recursive(input_string)
print(f"Reversed string of '{input_string}' is '{reversed_result}'")
```

11)Check is a String is Palindrome using recursion

CODE:

```
def is_palindrome_recursive(s):
    if len(s) <= 1:
        return True
    elif s[0] == s[-1]:
        return is_palindrome_recursive(s[1:-1])
    else:
        return False

# Example usage:
input_string = "racecar"
result = is_palindrome_recursive(input_string)
print(f"'{input_string}' is a palindrome: {result}")
```

12)Compute sum of first N natural numbers using recursion

CODE:

```
def sum_of_natural_numbers_recursive(N):
    if N == 0:
        return 0
    else:
        return N + sum_of_natural_numbers_recursive(N - 1)

# Example usage:
N = 5
result = sum_of_natural_numbers_recursive(N)
print(f"Sum of the first {N} natural numbers is {result}")
```

13) Implement pow(x,n), which calculates x raised to the power n (i.e., xn)

CODE:

```
def my_pow(x: float, n: int) -> float:
    # Base case: If n is 0, return 1
    if n == 0:
        return 1.0

    # If n is negative, compute the reciprocal of x and negate n
    if n < 0:
        x = 1 / x
        n = -n

    # Initialize the result
    result = 1.0
```

```
    # Binary exponentiation: Divide n by 2 until it becomes 0
    while n > 0:
        # If n is odd, multiply result by x
        if n % 2 == 1:
            result *= x
        # Square x and halve n
        x *= x
        n //= 2

    return result

# Example usage
x1, n1 = 2.00000, 10
x2, n2 = 2.10000, 3
x3, n3 = 2.00000, -2

print(f"Output for x = {x1}, n = {n1}: {my_pow(x1, n1):.5f}")
print(f"Output for x = {x2}, n = {n2}: {my_pow(x2, n2):.5f}")
print(f"Output for x = {x3}, n = {n3}: {my_pow(x3, n3):.5f}")
```

14) Given an integer n, return true if it is a power of two. Otherwise, return false.
An integer n is a power of two, if there exists an integer x such that n ==2x.

CODE:

```
 def is_power_of_two(n: int) -> bool:
    # Base case: If n is non-positive, it cannot be a power of two
    if n <= 0:
        return False
```

```
    # Keep dividing n by 2 until it becomes 1
    while n > 1:
        if n % 2 != 0:
            return False
        n //= 2

    return True

# Example usage
print(is_power_of_two(1))
print(is_power_of_two(16))
print(is_power_of_two(3))
```

15)Given an integer n, return true if it is a power of three. Otherwise, return false.
An integer n is a power of three, if there exists an integer x such that n == 3x.

CODE:

```
def is_power_of_three(n: int) -> bool:
    # Base case: If n is non-positive, it cannot be a power of three
    if n <= 0:
        return False

    # Keep dividing n by 3 until it becomes 1
    while n > 1:
        if n % 3 != 0:
            return False
        n //= 3

    return True
```

```
# Example usage
print(is_power_of_three(27))
print(is_power_of_three(0))
print(is_power_of_three(-1))
```

16) Given an integer n, return true if it is a power of four. Otherwise, return false.
An integer n is a power of four, if there exists an integer x such that n == 4x.

CODE:

```
def is_power_of_four(n: int) -> bool:
    # Base case: If n is non-positive, it cannot be a power of four
    if n <= 0:
        return False

    # Keep dividing n by 4 until it becomes 1
    while n > 1:
        if n % 4 != 0:
            return False
        n //= 4

    return True

# Example usage
print(is_power_of_four(16))
print(is_power_of_four(5))
print(is_power_of_four(1))
```

17)The Fibonacci numbers, commonly denoted F(n) form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
Given n, calculate F(n).

CODE:

```
def fibonacci(n: int) -> int:
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

# Example usage
n1 = 2
n2 = 3
n3 = 4

print(f"F({n1}) = {fibonacci(n1)}")
print(f"F({n2}) = {fibonacci(n2)}")
print(f"F({n3}) = {fibonacci(n3)}")
```

18)Find the Nth number of the series. The nth number of geek-onacci series is a sum of the last three numbers (summation of N-1th, N-2th, and N-3th geek-onacci numbers)

Input:

1. The first line of the input contains a single integer T denoting the number of test cases. The description of T test cases follows.
2. The first line of each test case contains four space-separated integers A, B, C, and N.

CODE:

```
def geek_onacci(A: int, B: int, C: int, N: int) -> int:
    if N == 1:
        return A
    elif N == 2:
        return B
    elif N == 3:
        return C

    # Initialize the geek-onacci series
    geek_series = [A, B, C]

    # Calculate the Nth geek-onacci number
    for i in range(3, N):
        next_number = geek_series[-1] + geek_series[-2] + geek_series[-3]
        geek_series.append(next_number)

    return geek_series[-1]

# Example usage
T = int(input())  # Number of test cases
for _ in range(T):
    A, B, C, N = map(int, input().split())
    print(geek_onacci(A, B, C, N))
```

19)f(0) = a;
f(1) = b;
f(n) = f(n-1) ^ f(n-2);  when n>1, where ^ denotes the bitwise xor operation.
You are given three integers a,b and n , calculate f(n).

Input

The input contains one or more independent test cases.

The first line of input contains a single integer T (1≤T≤103), the number of test cases.

Each of the T following lines contains three space-separated integers a, b, and n (0≤a,b,n≤109) respectively.

CODE:

```
def calculate_fn(a: int, b: int, n: int) -> int:
    if n == 0:
        return a
    elif n == 1:
        return b

    # Initialize the first two values
    fn_minus_2, fn_minus_1 = a, b

    # Calculate f(n) iteratively
    for i in range(2, n + 1):
        fn = fn_minus_1 ^ fn_minus_2
        fn_minus_2, fn_minus_1 = fn_minus_1, fn

    return fn
```

```
# Input
T = int(input())  # Number of test cases
for _ in range(T):
    a, b, n = map(int, input().split())
    print(calculate_fn(a, b, n))
```

20) Count Number of Zeros in a Number using Recursion.

CODE:

```
def count_zeros_recursive(num: int) -> int:
    # Base case: If the number is 0, it contains one zero
    if num == 0:
        return 1

    # Recursive case: Check the last digit
    last_digit = num % 10
    if last_digit == 0:
        return 1 + count_zeros_recursive(num // 10)
    else:
        return count_zeros_recursive(num // 10)

# Example usage
number = 102040
zero_count = count_zeros_recursive(number)
print(f"Number of zeros in {number}: {zero_count}")
```

21)Given an integer num, return the number of steps to reduce it to zero.

In one step, if the current number is even, you have to divide it by 2, otherwise, you have to subtract 1 from it.

CODE:

```
def reduce_to_zero(num: int) -> int:
    steps = 0
    while num > 0:
        if num % 2 == 0:
            num //= 2
        else:
            num -= 1
        steps += 1
    return steps

# Example usage
print(reduce_to_zero(14))
print(reduce_to_zero(8))
print(reduce_to_zero(123))
```

22) Find sum of an array elements using recursion

CODE:

```
def array_sum_recursive(arr, n):
    if n == 0:
        return 0
    else:
        return arr[n - 1] + array_sum_recursive(arr, n - 1)
```

```
# Example usage:
my_array = [1, 2, 3, 4, 5]
array_size = len(my_array)
result = array_sum_recursive(my_array, array_size)
print(f"The sum of array elements is: {result}")
```

23)Find the mean of Array elements using recursion.

CODE:

```
def array_mean_recursive(arr, n):
    if n == 0:
        return 0
    else:
        return arr[n - 1] + array_mean_recursive(arr, n - 1)

def calculate_mean(arr):
    array_size = len(arr)
    if array_size == 0:
        return 0
    else:
        return array_mean_recursive(arr, array_size) / array_size

# Example usage:
my_array = [1, 2, 3, 4, 5]
mean_result = calculate_mean(my_array)
print(f"The mean of array elements is: {mean_result:.2f}")
```

24) Find maximum and minimum of array elements using recursion.

CODE:

```
def array_max_recursive(arr, n):
    if n == 1:
        return arr[0]
    else:
        return max(arr[n - 1], array_max_recursive(arr, n - 1))

def array_min_recursive(arr, n):
    if n == 1:
        return arr[0]
    else:
        return min(arr[n - 1], array_min_recursive(arr, n - 1))

# Example usage:
my_array = [5, 2, 9, 1, 7]
array_size = len(my_array)

max_result = array_max_recursive(my_array, array_size)
min_result = array_min_recursive(my_array, array_size)

print(f"The maximum element is: {max_result}")
print(f"The minimum element is: {min_result}")
```

25)Compute the factorial of a number using recursion.

CODE:

```
def factorial_recursive(n):
```

```
    if n == 0:
        return 1
    else:
        return n * factorial_recursive(n - 1)

# Example usage:
number = 5
factorial_result = factorial_recursive(number)
print(f"The factorial of {number} is: {factorial_result}")
```

27) Perform binary search using recursion.

Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.

You must write an algorithm with O(log n) runtime complexity.

CODE:

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

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

```
    return -1

# Example usage:
nums = [-1, 0, 3, 5, 9, 12]
target1 = 9
target2 = 2

result1 = search(nums, target1)
result2 = search(nums, target2)

print(f"Index of {target1} in nums: {result1}")
print(f"Index of {target2} in nums: {result2}")
```

28)Perform bubble sort using recursion.

CODE:

```
def bubble_sort_recursive(arr, n):
    if n == 1:
        return  # Base case: array of size 1 is already sorted

    # Perform one pass of bubble sort
    for i in range(n - 1):
        if arr[i] > arr[i + 1]:
            arr[i], arr[i + 1] = arr[i + 1], arr[i]  # Swap elements

    # Recur with the reduced array size
    bubble_sort_recursive(arr, n - 1)
```

```
# Example usage:
my_array = [64, 34, 25, 12, 22, 11, 90]
array_size = len(my_array)

bubble_sort_recursive(my_array, array_size)

print("Sorted array using bubble sort:")
print(my_array)
```

29)Perform insertion sort using Recursion.

CODE:

```
def insertion_sort_recursive(arr, n):
    if n <= 1:
        return  # Base case: array of size 1 or less is already sorted

    # Recur with the reduced array size
    insertion_sort_recursive(arr, n - 1)

    # Insert the last element into the correct position
    key = arr[n - 1]
    j = n - 2
    while j >= 0 and arr[j] > key:
        arr[j + 1] = arr[j]
        j -= 1
    arr[j + 1] = key

# Example usage:
```

```
my_array = [64, 34, 25, 12, 22, 11, 90]
array_size = len(my_array)

insertion_sort_recursive(my_array, array_size)

print("Sorted array using insertion sort:")
print(my_array)
```

30)Reverse a linked list using recursion.

CODE:

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

def reverse_linked_list(head):
    if not head or not head.next:
        return head  # Base case: empty list or single node

    # Recursively reverse the rest of the list
    new_head = reverse_linked_list(head.next)

    # Update the next pointer of the current node
    head.next.next = head
    head.next = None

    return new_head

# Example usage:
# Create a linked list: 1 -> 2 -> 3 -> 4 -> 5
```

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

print("Original linked list:")
current = head
while current:
    print(current.val, end=" -> ")
    current = current.next

reversed_head = reverse_linked_list(head)

print("\nReversed linked list:")
current = reversed_head
while current:
    print(current.val, end=" -> ")
    current = current.next
```

31)Merge two sorted Linked list using recursion.

CODE:

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

```
        self.next = next

def merge_sorted_lists(head1, head2):
    if not head1:
        return head2
    if not head2:
        return head1

    if head1.val < head2.val:
        head1.next = merge_sorted_lists(head1.next, head2)
        return head1
    else:
        head2.next = merge_sorted_lists(head1, head2.next)
        return head2

# Example usage:
# Create two sorted linked lists: 1 -> 3 -> 5 and 2 -> 4 -> 6
node5 = ListNode(5)
node3 = ListNode(3, node5)
head1 = ListNode(1, node3)

node6 = ListNode(6)
node4 = ListNode(4, node6)
head2 = ListNode(2, node4)

print("List 1:")
current = head1
while current:
    print(current.val, end=" -> ")
    current = current.next

print("\nList 2:")
```

```
current = head2
while current:
    print(current.val, end=" -> ")
    current = current.next

merged_head = merge_sorted_lists(head1, head2)

print("\nMerged sorted list:")
current = merged_head
while current:
    print(current.val, end=" -> ")
    current = current.next
```

32)Print all leaf nodes of a binary search tree using recursion.

CODE:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def print_leaf_nodes(root):
    if not root:
        return

    # Recur on the left subtree
    print_leaf_nodes(root.left)
```

```
    # Check if it's a leaf node (both left and right children are None)
    if not root.left and not root.right:
        print(root.val)

    # Recur on the right subtree
    print_leaf_nodes(root.right)

# Example usage:
# Create a sample BST: 4 -> 2 -> 6 -> 1 -> 3 -> 5 -> 7
root = TreeNode(4)
root.left = TreeNode(2)
root.right = TreeNode(6)
root.left.left = TreeNode(1)
root.left.right = TreeNode(3)
root.right.left = TreeNode(5)
root.right.right = TreeNode(7)

print("Leaf nodes of the BST:")
print_leaf_nodes(root)
```

33)Solve the tower of hanoi problem

CODE:

```
def towers_of_hanoi(n, source, target, auxiliary):
    if n == 1:
        print(f"Move disk 1 from {source} to {target}")
        return
    else:
```

```
        # Move n-1 disks from source to auxiliary
        towers_of_hanoi(n - 1, source, auxiliary, target)
        # Move the largest disk from source to target
        print(f"Move disk {n} from {source} to {target}")
        # Move n-1 disks from auxiliary to target
        towers_of_hanoi(n - 1, auxiliary, target, source)

# Example usage:
num_disks = 3
towers_of_hanoi(num_disks, 'A', 'C', 'B')
```

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

CODE:

```
def gcd_euclidean(a, b):
    while b:
        a, b = b, a % b
    return a

# Example usage:
num1 = 48
num2 = 18
result = gcd_euclidean(num1, num2)
print(f"The GCD of {num1} and {num2} is: {result}")
```