**DAY 2 LAB PROGRAMS**

1. **Write a program to find the reverse of a given number using recursive.**

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

if n == 0:

return rev

else:

return reverse\_number(n // 10, rev \* 10 + n % 10)

number = 12345

reversed\_number = reverse\_number(number)

print(f"The reverse of {number} is: {reversed\_number}")

1. **Write a program to find the perfect number.**

def is\_perfect\_number(num):

return num == sum([i for i in range(1, num) if num % i == 0])

print(is\_perfect\_number(28)) # Output: True

print(is\_perfect\_number(15)) # Output: False

1. **Write program that demonstrates the usage of these notations by analyzing the time complexity of some example algorithms.**

def analyze\_time\_complexity(algorithm):

# Analyze the time complexity of the algorithm

pass

# Example algorithm

def example\_algorithm(n):

for i in range(n):

print(i)

# Analyzing time complexity of the example algorithm

analyze\_time\_complexity(example\_algorithm)

1. **Write programs that demonstrate the mathematical analysis of non-recursive and recursive algorithms.**

# Non-Recursive Algorithm Analysis

def non\_recursive\_algorithm\_analysis(n):

for i in range(n):

print(i)

# Recursive Algorithm Analysis

def recursive\_algorithm\_analysis(n):

if n > 0:

print(n)

recursive\_algorithm\_analysis(n-1)

# Test the Algorithms

n = 5

non\_recursive\_algorithm\_analysis(n)

recursive\_algorithm\_analysis(n)

1. **. Write C programs for solving recurrence relations using the Master Theorem, Substitution Method, and Iteration Method will demonstrate how to calculate the time complexity of an example recurrence relation using the specified technique.**

def master\_theorem(a, b, k):

return f"T(n) = O(n^{k})"

def substitution\_method():

return f"T(n) = O(log(n))"

def iteration\_method():

return f"T(n) = O(n)"

1. **Given two integer arrays nums1 and nums2, return an array of their Intersection. Each element in the result must be unique and you may return the result in any or**

def intersection(nums1, nums2):

set1 = set(nums1)

set2 = set(nums2)

return list(set1.intersection(set2))

1. **Given two integer arrays nums1 and nums2, return an array of their intersection. Each element in the result must appear as many times as it shows in both arrays and you may return the result in any order.**

**from collections import Counter**

def intersect(nums1, nums2):

count1, count2 = Counter(nums1), Counter(nums2)

return list((count1 & count2).elements())

1. **Given an array of integers nums, sort the array in ascending order and return it.You must solve the problem without using any built-in functions in O(nlog(n)) time complexity and with the smallest space complexity possible.**

def merge\_sort(arr):

if len(arr) <= 1:

return arr

mid = len(arr) // 2

left = merge\_sort(arr[:mid])

right = merge\_sort(arr[mid:])

return merge(left, right)

def merge(left, right):

result = []

i = j = 0

while i < len(left) and j < len(right):

if left[i] < right[j]:

result.append(left[i])

i += 1

else:

result.append(right[j])

j += 1

result.extend(left[i:])

result.extend(right[j:])

return result

nums = [12, 11, 13, 5, 6, 7]

sorted\_nums = merge\_sort(nums)

print(sorted\_nums)

1. **Given an array of integers nums, half of the integers in nums are odd, and the other half are even.**

nums = [1, 2, 3, 4, 5, 6]

half\_odd\_even = [x for x in nums if x % 2 == 0] + [x for x in nums if x % 2 != 0]

print(half\_odd\_even)

1. **Sort the array so that whenever nums[i] is odd, i is odd, and whenever nums[i] is even, i is even. Return any answer array that satisfies this condition.**

def sort\_array\_by\_parity(nums):

nums.sort(key=lambda x: (x % 2, x % 2 == 0))

return nums