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):

sum\_divisors = 0

for i in range(1, num):

if num % i == 0:

sum\_divisors += i

return sum\_divisors == num

def find\_perfect\_numbers(limit):

perfect\_numbers = []

for i in range(1, limit + 1):

if is\_perfect\_number(i):

perfect\_numbers.append(i)

return perfect\_numbers

limit = 10000

perfect\_numbers = find\_perfect\_numbers(limit)

print("Perfect numbers up to", limit, "are:", perfect\_numbers)

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

# Python program that demonstrates the usage of notations by analyzing the time complexity of example algorithms

def example\_algorithm(n):

print("This is a constant time complexity algorithm")

for i in range(n):

print("This is a linear time complexity algorithm")

for i in range(n):

for j in range(n):

print("This is a quadratic time complexity algorithm")

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

def non\_recursive\_algorithm(n):

result = 0

for i in range(1, n+1):

result += i

return result

def recursive\_algorithm(n):

if n == 0:

return 0

else:

return n + recursive\_algorithm(n-1)

n = 5

non\_recursive\_result = non\_recursive\_algorithm(n)

recursive\_result = recursive\_algorithm(n)

print(f"Non-Recursive Algorithm Result for n={n}: {non\_recursive\_result}")

print(f"Recursive Algorithm Result for n={n}: {recursive\_result}")

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