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

**Program:**

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}")

Output:

The reverse of 12345 is: 54321

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

**Program:**

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**)**

**Output**:

Perfect numbers up to 10000 are: [6, 28, 496, 8128]

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

**Program:**

def linear\_search(arr, target):

for i in range(len(arr)):

if arr[i] == target:

return i

return -1

def binary\_search(arr, target):

low = 0

high = len(arr) - 1

while low <= high:

mid = (low + high) // 2

if arr[mid] < target:

low = mid + 1

elif arr[mid] > target:

high = mid - 1

else:

return mid

return -1

def factorial(n):

if n == 0:

return 1

return n \* factorial(n - 1)

def fibonacci(n):

if n <= 1:

return n

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

arr = [2, 3, 5, 7, 11, 13, 17, 19, 23]

target = 13

print("Linear Search Result:", linear\_search(arr, target))

arr.sort()

print("Binary Search Result:", binary\_search(arr, target))

n = 5

print("Factorial of", n, "is", factorial(n))

n = 6

print("Fibonacci Sequence at", n, "is", fibonacci(n))

**Output:**

Linear Search Result: 5

Binary Search Result: 5

Factorial of 5 is 120

Fibonacci Sequence at 6 is 8

1. **Write C programs demonstrating the mathematical analysis of non-recursive and recursive algorithms.**

**Program:**

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

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}")

**Output:**

Non-Recursive Algorithm Result for n=5: 15

Recursive Algorithm Result for n=5: 15

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.**

**Program:**

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

if isinstance(a, int) and isinstance(b, int) and isinstance(k, int) and isinstance(p, int):

if a > 0 and b > 1 and k >= 0 and p >= 0:

if a == b \*\* k:

return f"T(n) = Θ(n^{p} \* log^{k} n)"

elif a < b \*\* k:

return f"T(n) = Θ(n^{k})"

elif a > b \*\* k:

return f"T(n) = Θ(n^{math.log(a, b)})"

import math

a = 3

b = 2

k = 0

p = 1

print(master\_theorem(a, b, k, p))

**Output:**

T(n) = Θ(n^1.5849625007211563)

def substitution\_method(T, n):

if n == 1:

return 1

else:

return 3 \* T(n / 2) + n

def T(n):

if n == 1:

return 1

else:

return 3 \* T(n / 2) + n

n = 8

print(f"T({n}) = {T(n)}")

**Output**:

T(8) = 65.0

def iteration\_method(a, b, k, p, n):

while n >= 1:

if n == 1:

return 1

else:

return a \* iteration\_method(a, b, k, p, n / b) + n

a = 3

b = 2

k = 0

p = 1

n = 8

print(f"T({n}) = {iteration\_method(a, b, k, p, 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 order.**

**Program:**

def intersection(nums1, nums2):

set1 = set(nums1)

set2 = set(nums2)

return list(set1.intersection(set2))

nums1 = [1, 2, 2, 1]

nums2 = [2, 2]

print(intersection(nums1, nums2))

**Output:**

[2]

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.**

**Program:**

def intersection(nums1, nums2):

count1 = {}

for num in nums1:

count1[num] = count1.get(num, 0) + 1

intersection = []

for num in nums2:

if num in count1 and count1[num] > 0:

intersection.append(num)

count1[num] -= 1

return intersection

nums1 = [1, 2, 2, 1]

nums2 = [2, 2]

print(intersection(nums1, nums2))

**Output:**

**[2, 2]**

**8. 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.**

**Program:**

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)

**Output:**

**[5, 6, 7, 11, 12, 13]**

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

**Program:**

def generate\_array(n):

half = n // 2

odd\_numbers = [2 \* i + 1 for i in range(half)]

even\_numbers = [2 \* i for i in range(half)]

return odd\_numbers + even\_numbers

n = 6

result = generate\_array(n)

print(result)

**Output:**

**[1, 3, 5, 0, 2, 4]**

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.**

**Program:**

def sort\_array(nums):

odd\_nums = [x for x in nums if x % 2 == 1]

even\_nums = [x for x in nums if x % 2 == 0]

result = []

for odd, even in zip(odd\_nums, even\_nums):

result.extend([odd, even])

if len(odd\_nums) > len(even\_nums):

result.extend(odd\_nums[len(even\_nums):])

elif len(even\_nums) > len(odd\_nums):

result.extend(even\_nums[len(odd\_nums):])

return result

nums = [4, 2, 5, 7]

print(sort\_array(nums))

**Output:**

**[5, 4, 7, 2]**