Name: M. Charan

Reg no:192324115

Course code: CSA0676

Course name: DAA

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 the number 12345 is 54321

2. **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 = 100
perfect_numbers = find_perfect_numbers(limit)
print("Perfect numbers up to", limit, "are:", perfect_numbers)
```

output:

perfect numbers upto 100 are 6,28

3. **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:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1

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

4. **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)
```

5. **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):
    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)"
```

6. **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))
```

7. **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 intersect(nums1, nums2):
    count1, count2 = Counter(nums1), Counter(nums2)
    return list((count1 & count2).elements())

nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
print(intersect(nums1, nums2))
```

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)
```

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

   Program:

```
nums = [1, 2, 3, 4, 5, 6]
half_length = len(nums) // 2
half_odd = [num for num in nums if num % 2 != 0][:half_length]
half_even = [num for num in nums if num % 2 == 0][:half_length]
result = half_odd + half_even
print(result)
```

10. **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):
    nums.sort(key=lambda x: (x % 2, x % 2 == 0))
    return nums
```