1. WRITE A PROGRAM TO FIND THE REVERSE OF A GIVEN NUMBER USING RECURSIVE.

```
def reverse_number(n, reverse=0):
    if n == 0:
        return reverse
    else:
        return reverse_number(n // 10, reverse * 10 + n % 10)

number = 12345
reversed_number = reverse_number(number)
print(f"The reverse of {number} is {reversed_number}")
```

2.WRITE A PROGRAM TO FIND THE PERFECT NUMBER

```
def is_perfect_number(n):
    if n < 1:
        return False

    sum_of_divisors = 0
    for i in range(1, n // 2 + 1):
        if n % i == 0:
            sum_of_divisors += i

    return sum_of_divisors == n

number = 28
if is_perfect_number(number):
    print(f"{number} is a perfect number.")
else:
    print(f"{number} is not a perfect number.")
```

3.DEMONSTRATES THE USAGE OF THESE NOTATIONS BY ANALYZING THE TIME COMPLEXITY OF SOME EXAMPLE ALGORITHMS.

EXAMPLE ONE:-

Linear search:-

```
def linear_search(arr, target):
    for element in arr:
        if element == target:
            return True
    return False
```

EXAMPLE TWO :-

binary search :-

```
def binary_search(arr, target):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return True
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return False
```

4. WRITE PROGRAMS THAT DEMONSTRATE THE MATHEMATICAL ANALYSIS OF NON-RECURSIVE AND RECURSIVE ALGORITHMS.

NON-RECURSIVE

```
def iterative_factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```

RECURSIVE

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

5. WRITE 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.

```
# Master Theorem
def master_theorem(a, b, f_n):
    return f"Theta(n log n)" if f_n == "n" else "Other cases not implemented"


# Substitution Method
def substitution_method(n):
    # This is a conceptual demonstration and not an actual program.
    return "O(n log n)"


# Iteration Method
def iteration_method(n):
    # This is a conceptual demonstration and not an actual program.
    return "O(n log n)"


print(master_theorem(2, 2, "n"))
print(substitution_method(16))
print(iteration_method(16))
```

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.

```
def intersection(nums1, nums2):
    set_nums1 = set(nums1)
```

```
    set_nums2 = set(nums2)

    result_set = set_nums1.intersection(set_nums2)

    return list(result_set)

nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
```

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.

```
def intersect(nums1, nums2):
    counts = {}
    for num in nums1:
        counts[num] = counts.get(num, 0) + 1

    result = []
    for num in nums2:
        if num in counts and counts[num] > 0:
            result.append(num)
            counts[num] -= 1

    return result

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.

```
def merge_sort(nums):
    if len(nums) > 1:
```

```
    mid = len(nums) // 2
    left_half = nums[:mid]
    right_half = nums[mid:]

    merge_sort(left_half)
    merge_sort(right_half)
    i = j = k = 0
    while i < len(left_half) and j < len(right_half):
        if left_half[i] < right_half[j]:
            nums[k] = left_half[i]
            i += 1
        else:
            nums[k] = right_half[j]
            j += 1
        k += 1

    while i < len(left_half):
        nums[k] = left_half[i]
        i += 1
        k += 1

    while j < len(right_half):
        nums[k] = right_half[j]
        j += 1
        k += 1

  return nums

nums = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
sorted_nums = merge_sort(nums)
print(sorted_nums)  # Output: [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]
```

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

```
def separate_even_odd(nums):
    evens = [num for num in nums if num % 2 == 0]
    odds = [num for num in nums if num % 2 != 0]
    return evens, odds

nums = [1, 2, 3, 4, 5, 6]
evens, odds = separate_even_odd(nums)
print("Even numbers:", evens)
print("Odd numbers:", odds)
```

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.

```
def sort_by_parity(nums):
    even_index = 0
    odd_index = 1
    size = len(nums)

   while even_index < size and odd_index < size:
       if nums[even_index] % 2 == 0:
           even_index += 2
       elif nums[odd_index] % 2 == 1:
           odd_index += 2
       else:
           nums[even_index], nums[odd_index] = nums[odd_index], nums[even_index]
           even_index += 2
           odd_index += 2

    return nums

nums = [4, 2, 5, 7]
```

```
print(sort_by_parity(nums))  # Output could be: [4, 5, 2, 7]
```