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}")
```

Output:

The reverse of 12345 is: 54321

2. 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 num in range(1, limit + 1):
```

```
        if is_perfect_number(num):
            perfect_numbers.append(num)
    return perfect_numbers

limit = 10000
perfect_numbers = find_perfect_numbers(limit)
print("Perfect numbers up to", limit, "are:", perfect_numbers)
```

Ouput:

Perfect numbers up to 100 are: [6, 28]

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

```
# Example algorithm with O(1) time complexity
def constant_time_complexity(n):
    return n * 2

# Example algorithm with O(n) time complexity
def linear_time_complexity(lst):
    for item in lst:
        print(item)

# Example algorithm with O(n^2) time complexity
def quadratic_time_complexity(lst):
    for i in lst:
        for j in lst:
```

```
        print(i, j)

# Example algorithm with O(log n) time complexity
def logarithmic_time_complexity(n):
    while n > 1:
        n = n // 2
        print(n)

# Example algorithm with O(n log n) time complexity
def n_log_n_time_complexity(lst):
    lst.sort()
    for item in lst:
        print(item)
```

4. Write python programs that demonstrate the mathematical analysis of non-recursive and recursive algorithms.

```
# Non-Recursive Algorithm
def non_recursive_factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

print(non_recursive_factorial(5))

 # Recursive Algorithm
```

```
def recursive_factorial(n):
    if n == 0:
        return 1
    else:
        return n * recursive_factorial(n - 1)
print(recursive_factorial(5))
```

Output:

120

120

5. Write python 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):
    if a > b ** k:
        return "T(n) = Θ(n^log_{%d}(%d))" % (b, a)
    elif a == b ** k:
        return "T(n) = Θ(n^log_{%d}(%d) * log n)" % (b, a)
    else:
        return "T(n) = Θ(n^%d)" % k

# Example Usage
print(master_theorem(3, 2, 1))

def substitution_method(T, n):
```

```
    if n == 0:
        return 1
    else:
        return 2 * substitution_method(T, n-1) + 1

# Example Usage
print(substitution_method(1, 3))


def iteration_method(a, b, n):
    result = a
    for i in range(1, n):
        result = result * b
    return result

# Example Usage
print(iteration_method(2, 3, 4))
```

Output:

```
T(n) = Θ(n^log_{2}(3))
```

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):
    return list(set(nums1) & set(nums2))
```

```
# Test the function
nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
print(intersection(nums1, nums2))  # Output: [2]

nums1 = [4, 9, 5]
nums2 = [9, 4, 9, 8, 4]
print(intersection(nums1, nums2))  # Output: [4, 9]

nums1 = [1, 2, 3, 4, 5]
nums2 = [6, 7, 8, 9, 10]
print(intersection(nums1, nums2))  # Output: []
```

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.

```
from collections import Counter

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

# Example Usage
nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
```

```
print(intersect(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.

```
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 = [64, 34, 25, 12, 22, 11, 90]
sorted_nums = merge_sort(nums)
print(sorted_nums)
```

Output:

```
[11, 12, 22, 25, 34, 64, 90]
```

9. 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)
```

Output:

```
[2, 4, 6, 1, 3, 5]
```

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_array(nums):
    next_even, next_odd = 0, len(nums) - 1
    while next_even < next_odd:
        if nums[next_even] % 2 == 0:
```

```
            next_even += 1
        else:
            nums[next_even], nums[next_odd] = nums[next_odd], nums[next_even]
            next_odd -= 1
    return nums

# Test the function
nums = [12, 10, 9, 45, 2, 10, 10, 45]
print(sort_array(nums))
```

Output:

```
[12, 10, 10, 10, 2, 45, 45, 9]
```