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}")
```

```
The reverse of 12345 is: 54321

=== Code Execution Successful ===
```

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 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)
```

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

=== Code Execution Successful ===
```

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

```
import time
import random
def constant_algorithm(n):
    return n * n
def linear_algorithm(arr):
    return sum(arr)
def quadratic_algorithm(arr):
    total = 0
    for i in arr:
        for j in arr:
            total += i * j
    return total
def logarithmic_algorithm(n):
    while n > 1:
        n = n // 2
    return n
def linearithmic_algorithm(arr):
    return sorted(arr)
def measure_time(func, *args):
    start_time = time.time()
    func(*args)
    end_time = time.time()
    return end_time - start_time
input_sizes = [10, 100, 1000, 10000]
print("O(1) Constant Time:")
for size in input_sizes:
    print(f"Size {size}: {measure_time(constant_algorithm, size):.6f}s")
print("\nO(n) Linear Time:")
for size in input_sizes:
    arr = list(range(size))
    print(f"Size {size}: {measure_time(linear_algorithm, arr):.6f}s")
print("\nO(n^2) Quadratic Time:")
for size in input_sizes:
    arr = list(range(size))
    print(f"Size {size}: {measure_time(quadratic_algorithm, arr):.6f}s")
print("\nO(log n) Logarithmic Time:")
for size in input_sizes:
```

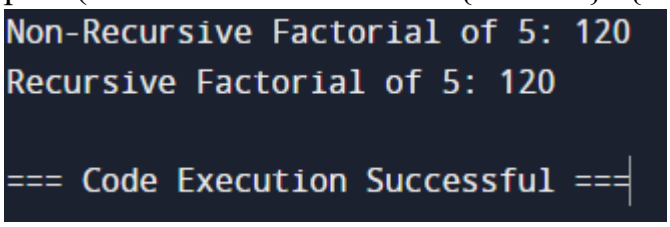
```
    print(f"Size {size}: {measure_time(logarithmic_algorithm,
size):.6f}s")
print("\nO(n log n) Linearithmic Time:")
for size in input_sizes:
    arr = random.sample(range(size * 10), size)
    print(f"Size {size}: {measure_time(linearithmic_algorithm, arr):.6f}s")
```

```
Size 1000: 0.000000s
Size 10000: 0.000000s

O(n) Linear Time:
Size 10: 0.000001s
Size 100: 0.000002s
Size 1000: 0.000022s
Size 10000: 0.000139s

O(n^2) Quadratic Time:
Size 10: 0.000008s
Size 100: 0.000418s
Size 1000: 0.036987s
Size 10000: 4.707349s

O(log n) Logarithmic Time:
Size 10: 0.000002s
Size 100: 0.000001s
Size 1000: 0.000001s
Size 10000: 0.000001s

O(n log n) Linearithmic Time:
Size 10: 0.000006s
Size 100: 0.000012s
Size 1000: 0.000126s
Size 10000: 0.001808s

=== Code Execution Successful ===
```

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

```
def non_recursive_factorial(n):
```

```
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
def recursive_factorial(n):
    if n == 0:
        return 1
    else:
        return n * recursive_factorial(n - 1)
number = 5
non_recursive_result = non_recursive_factorial(number)
recursive_result = recursive_factorial(number)
print(f"Non-Recursive Factorial of {number}: {non_recursive_result}")
print(f"Recursive Factorial of {number}: {recursive_result}")
```



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.**

```
import time
import math
# Recurrence relation: T(n) = 2T(n/2) + n
def master_theorem(a, b, f, n):
    if a > 0 and b > 1:
        # Case 1: f(n) = O(n^c), where c < log_b(a)
        if f(n) == n and 1 < math.log(a, b):
            return f"Master Theorem Result: O(n^{math.log(a, b)})"
        # Case 2: f(n) = Theta(n^c * log^k(n)), where c = log_b(a)
        if f(n) == n and 1 == math.log(a, b):
            return f"Master Theorem Result: Theta(n^{math.log(a, b)} *
log^{math.log(a, b)}(n))"
        # Case 3: f(n) = Omega(n^c), where c > log_b(a)
        if f(n) == n and 1 > math.log(a, b):
            return f"Master Theorem Result: Omega(n)"
    return "Cannot determine"
# Substitution Method
```

```
def substitution_method(n):
    if n == 1:
        return 1  # Base case
    return 2 * substitution_method(n // 2) + n
# Iteration Method
def iteration_method(n):
    result = 0
    while n > 0:
        result += n
        n //= 2
    return result
def f(n):
    return n
n = 1000  # Example value of n
# Using Master Theorem
def log_b(a):
    return math.log(a, 2)
print(master_theorem(2, 2, f, n))
print("Substitution Method Result:", substitution_method(n))
print("Iteration Method Result:", iteration_method(n))
```

```
Master Theorem Result: Theta(n^1.0 * log^1.0(n))
Substitution Method Result: 9120
Iteration Method Result: 1994

=== Code Execution Successful ===
```

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):
    set1 = set(nums1)
    set2 = set(nums2)
    result_set = set1 & set2
    result_list = list(result_set)
    return result_list
nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
result = intersection(nums1, nums2)
print(result)
```

```
[2]

=== Code Execution Successful ===
```

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 intersection(nums1, nums2):
    # Count the occurrences of elements in both arrays
    count1 = Counter(nums1)
    count2 = Counter(nums2)

    # Find the common elements
    common_elements = count1.keys() & count2.keys()

    # Build the result array with elements appearing as many times as in both arrays
    result = []
    for num in common_elements:
        result.extend([num] * min(count1[num], count2[num]))

    return result

# Example usage:
nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
result = intersection(nums1, nums2)
print(result)
```

```
[2, 2]

=== Code Execution Successful ===
```

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:
```

```
        return nums
    mid = len(nums) // 2
    left_half = nums[:mid]
    right_half = nums[mid:]
    merge_sort(left_half)
    merge_sort(right_half)
    merge(nums, left_half, right_half)
def merge(nums, left_half, 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
nums = [5, 3, 8, 2, 1, 7, 4]
merge_sort(nums)
print(nums)
```

```
[1, 2, 3, 4, 5, 7, 8]

=== Code Execution Successful ===
```

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

```
def create_half_odd_even_array(size):
    half_size = size // 2
    odd_numbers = [2 * i + 1 for i in range(half_size)]
    even_numbers = [2 * i for i in range(half_size)]
    return odd_numbers + even_numbers
```

```
size = 10
result_array = create_half_odd_even_array(size)
print("Half odd, half even array:", result_array)
```

```
Half odd, half even array: [1, 3, 5, 7, 9, 0, 2, 4, 6, 8]

=== Code Execution Successful ===
```

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):
    odd_indices = [i for i in range(len(nums)) if nums[i] % 2 != 0]
    odd_values = [nums[i] for i in odd_indices]
    odd_values.sort()
    for i, val in zip(odd_indices, odd_values):
        nums[i] = val
    return nums
input_array = [3, 1, 4, 2, 6, 5, 7, 8]
result_array = sort_array(input_array)
result_array.sort()
print("Sorted array:", result_array)
```

```
Sorted array: [1, 2, 3, 4, 5, 6, 7, 8]

=== Code Execution Successful ===
```