```
1.num = int(input("Enter the number: "))
revr_num = 0     # initial value is 0. It will hold the reversed number
def recur_reverse(num):
    global revr_num    # We can use it out of the function
    if (num > 0):
        Reminder = num % 10
        revr_num = (revr_num * 10) + Reminder
        recur_reverse(num // 10)
    return revr_num
revr_num = recur_reverse(num)
print("n Reverse of entered number is = %d" % revr_num)
```

2 . Write a program to find the perfect number.

```
def is_perfect(num):
    sum_divisors = sum([i for i in range(1, num) if num % i == 0])
    return sum_divisors == num


perfect_numbers = [num for num in range(1, 10000) if is_perfect(num)]
print(perfect_numbers)
```

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

```
# Python code demonstrating the analysis of time complexity using example algorithms


# Example algorithm 1
def example_algorithm_1(n):
    for i in range(n):
        print(i)

# Example algorithm 2
def example_algorithm_2(n):
    total = 0
    for i in range(n):
        total += i
    return total

# Call example algorithms with input size n
n = 5
example_algorithm_1(n)
result = example_algorithm_2(n)
print(result).
```

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

```
# Non-recursive algorithm analysis
def non_recursive_analysis(n):
    for i in range(n):
        print(i)

# Recursive algorithm analysis
def recursive_analysis(n):
    if n > 0:
        print(n)
        recursive_analysis(n-1)


# Demonstrate non-recursive algorithm analysis
non_recursive_analysis(5)


# Demonstrate recursive algorithm analysis
recursive_analysis(5)
```

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.

```
# Master Theorem Method
def master_theorem(a, b, k):
    return f"O(n^{k})"


# Substitution Method
def substitution_method():
    return "O(n)"


# Iteration Method
def iteration_method(T, n):
    for i in range(1, n):
        T[i] = T[i-1] + 1
    return T[n-1]


# Example Usage
```

```
print(master_theorem(2, 2, 1))

print(substitution_method())

T = [0] * 10

print(iteration_method(T, 10))
```

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):

  res = set(nums1) & set(nums2)

  return list(res)


nums1 = [1,2,2,1]

nums2 = [2,2]

print(intersection(nums1, nums2))
```

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


class Solution:

    def intersect(self, nums1, nums2):

        count1, count2 = Counter(nums1), Counter(nums2)

        return list((count1 & count2).elements())
```

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 = [12, 4, 7, 1, 9, 3]
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.

```
def divide_odd_even(nums):
    odd_nums = [num for num in nums if num % 2 != 0]
    even_nums = [num for num in nums if num % 2 == 0]
    return odd_nums, even_nums


# Example Usage
nums = [1, 2, 3, 4, 5, 6]
odd_numbers, even_numbers = divide_odd_even(nums)
print("Odd Numbers:", odd_numbers)
print("Even Numbers:", even_numbers)
```

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