1. Write a program to find the reverse of a given number using recursive.

```
def reverse_number(num, rev=0):
  if num == 0:
    return rev
  return reverse_number(num // 10, rev * 10 + num % 10)

num = int(input("Enter a number: "))
print(f"Reversed number: {reverse_number(num)}")
```

output

Reversed number : 321

Time complexity

O(log n)

2. Write a program to find the perfect number.

```
def perfect(num,summ=0):
  if num <= 1:
    return 1
  for i in range(1,num):
    if(num%i==0):
      summ = summ+i
  if(summ==num):
    print(f'{num} is a perfect number')
  else:
    print(f'{num} is not perfect number')
num = 6
perfect(num)
```

output

6 is a perfect number

Time complexity

O(sqrt(n))

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

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void constantTimeExample(int arr[], int size) {
  printf("First element: %d\n", arr[0]);
}

void linearTimeExample(int arr[], int size) {
  int sum = 0;
  for(int i = 0; i < size; i++) {
    sum += arr[i];
  }
  printf("Sum: %d\n", sum);
}
```

```
void quadraticTimeExample(int arr[], int size) {
  for(int i = 0; i < size; i++) {
    for(int j = 0; j < size; j++) {
      printf("%d ", arr[i] * arr[j]);
    }
    printf("\n");
  }
}

int main() {
  int size = 5;
  int arr[] = {1, 2, 3, 4, 5};

  constantTimeExample(arr, size);
  linearTimeExample(arr, size);
  quadraticTimeExample(arr, size);

  return 0;
}
```

Output

First element: 1

Sum: 15

1 2 3 4 5

2 4 6 8 10

3 6 9 12 15

4 8 12 16 20

5 10 15 20 25

Time complexity

O(1)

O(n)

O(n^2)

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

```
#recursion
def fibonacci(n):
  if n <= 1:
    return n
  else:
    return fibonacci(n-1) + fibonacci(n-2)

def fibonacci_series(n):
  if n <= 0:
    return
  else:
    for i in range(n):
      print(fibonacci(i), end=" ")

num = 10
```

```
fibonacci_series(num)
print()

#non recursion
print("non recursion")
n = 10
fib_series = [0, 1]
for i in range(2, n + 1):
    fib_series.append(fib_series[i-1] + fib_series[i-2])
print(fib_series)
```

output
0 1 1 2 3 5 8 13 21 34
non recursion
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
Time complexity
O(2^n)
O(n)

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.

```
#include <stdio.h>
int masterTheoremExample(int n) {
    if (n <= 1) {
        return n;
    }
    return 2 * masterTheoremExample(n / 2) + n;
}

int main() {
    int n = 16;
    printf("Result using Master Theorem: %d\n", masterTheoremExample(n));
    return 0;
}
#include <stdio.h>
int substitutionMethodExample(int n) {
    if (n <= 1) {
        return 0;
    }
    return substitutionMethodExample(n / 2) + 1;
}

int main() {
    int n = 16;
    printf("Result using Substitution Method: %d\n", substitutionMethodExample(n));
    return 0;
}
#include <stdio.h>
int iterationMethodExample(int n) {
```

```
    if (n <= 1) {
        return n;
    }
    return 3 * iterationMethodExample(n / 3) + n;
}

int main() {
    int n = 27;
    printf("Result using Iteration Method: %d\n", iterationMethodExample(n));
    return 0;
}
```

Output

Result using Master Theorem: 80

Result using Substitution Method: 4

Result using Iteration Method: 108

Time complexity

O(n^2)

O(log n)

O(n^(log3(a)/b))

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_unique(nums1, nums2):
    return list(set(nums1) & set(nums2))

nums1 = [1,2,3,5]
nums2 =[2,3]
print(f"Intersection (unique elements): {intersection_unique(nums1, nums2)}")
```

output

[2,3]

Time complexity

O(m + n)

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_with_duplicates(nums1, nums2):
    count1 = Counter(nums1)
    count2 = Counter(nums2)
    result = []
    for num in count1:
        if num in count2:
            result.extend([num] * min(count1[num], count2[num]))
    return result

nums1 = [2,3,4,5]
nums2 = [1,2,3,3,4]
print(f"Intersection (including duplicates):
{intersection_with_duplicates(nums1, nums2)}")
```

output
Intersection (including duplicates):[2,3,3,3,4,4]
Time complexity
O(n log n)

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 = [10,1,9,23,22,19,15,14]
print(f"Sorted array: {merge_sort(nums)}")
```

output
[1,9,10,14,15,19,22,23]
Time complexity
O(nlog(n))

9. Given an array of integers nums, half of the integers in nums are odd, and the other half are even. 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(nums):
 i, j = 0, 1
  while i < len(nums) and j < len(nums):
    while i < len(nums) and nums[i] % 2 == 0:
      i += 2

    while j < len(nums) and nums[j] % 2 == 1:
      j += 2

    if i < len(nums) and j < len(nums):
      nums[i], nums[j] = nums[j], nums[i]
      i += 2
      j -= 2

  return nums
nums = [4, 2, 5, 7]
sorted_nums = sort(nums)
print(sorted_nums)
```

output

[4,5,2,7]

Time complexity

O(n)