1. Write a program to Print Fibonacci Series using recursion.

```
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)
def print_fibonacci_series(n):
    if n <= 0:
        return
    else:
        for i in range(n):
            print(fibonacci(i), end=" ")
    print(" Fibonacci Series:", end=" ")
    print_fibonacci_series(10)
```

output

0 1 1 2 3 5 8 13 21 34

Time complexity

$O(2^n)$

2. Write a program to check the given no is Armstrong or not using recursive function.

```
def is_armstrong(num):
    def armstrong_rec(n, sum):
        if n == 0:
            return sum
        else:
            return armstrong_rec(n//10, sum + ((n % 10) ** 3))

if num == armstrong_rec(num, 0):
        return True
    else:
        return False
```

```
print("Armstrong numbers up to 1000:", [num for num in range(1000) if is_armstrong(num)])
```

output

Armstrong numbers up to 1000: [0, 1, 153, 370, 371, 407]

Time complexity

O(log n)

3. Write a program to find the GCD of two numbers using recursive factorization

```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
print(" GCD of 24 and 36:", gcd(24, 36))
```

output

GCD of 24 and 36 : 12

Time complexity

O(log min(a, b))

4. Write a program to get the largest element of an array.

```
def get_largest_element(arr):
    if len(arr) == 1:
        return arr[0]
    else:
        return max(arr[0], get_largest_element(arr[1:]))
  print(" Largest element of [5, 10, 3, 7, 2]:", get_largest_element([5, 10, 3, 7, 2]))
```

output

Largest element of [5, 10, 3, 7, 2]: 10

Time complexity

O(n)

5. Write a program to find the Factorial of a number using recursion.

```
def factorial(n):
    if n == 0:
        return 1
```

```
    else:
        return n * factorial(n-1)
print("Factorial of 5:", factorial(5))
```

output

Factorial of 5: 120

Time complexity

O(n)

6. Write a program for to copy one string to another using recursion

```
def copy_string(source, dest, i=0):
    if i == len(source):
        return dest
    else:
        dest += source[i]
        return copy_string(source, dest, i+1)
print("Copy 'Hello' to another string:", copy_string("Hello", ""))
```

output

Copy 'Hello' to another string: Hello

Time complexity

O(n)

7. Write a program to print the reverse of a string using recursion

```
def reverse_string(string):
    if len(string) == 0:
        return string
    else:
        return reverse_string(string[1:]) + string[0]
print(" Reverse of 'Hello':", reverse_string("Hello"))
```

output

Reverse of 'Hello': olleH

Time complexity

O(n)

8. Write a program to generate all the prime numbers using recursion

```
def generate_primes(n):
    primes = []
    def is_prime(num, i=2):
        if num <= 2:
            return True
        elif num % i == 0:
            return False
        elif i * i > num:
            return True
        else:
            return is_prime(num, i+1)
for i in range(2, n+1):
        if is_prime(i):
            primes.append(i)
 return primes
print("Prime numbers up to 20:", generate_primes(20))
```

output

Prime numbers up to 20: [2, 3, 5, 7, 11, 13, 17, 19]

Time complexity

O(n * sqrt(n))

9. Write a program to check a number is a prime number or not using recursion.

```
def is_prime(num, i=2):
    if num <= 2:
        return True
    elif num % i == 0:
        return False
    elif i * i > num:
        return True
    else:
```

```
        return is_prime(num, i+1)
print("Is 17 a prime number?", is_prime(17))
```

output

Is 17 a prime number? True

Time complexity

O(sqrt(num)).

10.Write a program for to check whether a given String is Palindrome or not using recursion

```
def is_palindrome(string):
    if len(string) <= 1:
        return True
    elif string[0] != string[-1]:
        return False
    else:
        return is_palindrome(string[1:-1])
print(" Is 'racecar' a palindrome?", is_palindrome("racecar"))
```

output

Is 'racecar' a palindrome? True

Time complexity

O(n)