Name: M. Charan

Reg.no: 192324115

Course code: CSA0676

Couse Name: DAA

1.

Aim: Write a program to Print Fibonacci Series using recursion.

Algorithm:-

- Check if input n is less than or equal to 0.retrun the string "invalid input".
- The input should be a positive number .
- Check if n is equal to 2.If true ,return 1.This is second number in Fibonacci series.
- If the above conditions are not met ,recursively call the Fibonacci () function with n-1 and n-2, and return the sum of the two recursive calls.
- Using print() with can end parameter set to"to display the numbers.

Time complexity:-

- Time complexity for thr recursive Fibonacci algorithm is O(2^n).
- Space complexity for the recursive Fibonacci algorithm is O(n).

Program in python:-

```
import time

def recur_fibo (0)

it n<= 1:

    return n

else:

        return recur-fibo (n-1) + recur_fibo(n-2)

 nterms = 5

if nterms <= 0:

    print (Fibonacci Sequence.")

     start_time = time.time()

     for i in range (nterms):

     print (recur-fibo(i))

      end_time = time.time()

     execution_time = end_time start-time
```

print("Execution time:", execution_time, "Second")

Output:

Fibonacci Sequence.. 0 1 1 2 3

Execution time: 0.00011563301086425781 Seconds.

Result: The Code Was executed Successfully.

2.

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

Algorithm:

- Define a helper recursive function to compute the sum of the digits each raised to the power of the number of digits.
- Define the main function to check if a given number is an Armstrong Number.
- In the main function, Compule the number of digits in the number.
- Use the helper recursive function to compute the sum.
- Compare the Sum with the Original number to determine if it is an Armstrong Number.

Time Complexity:-

It's O(log(n) "log (log(n)) Log(n) for iteration.

In pow (a,b) the Complexity is log(b). So hereyour b = log(n)

so log (log(n)) for pow function.

Program in python:

```
det Sum of powers (n, power):
if n == o
return 0
else:
last_digit= n % 10
return (last_digit** power) + Sum_of_powers(n// 10, Power)
det is_armstrong (number):
num-digits = len(str (number))
Sum-power = Sum-of-powers (number, num_digits)
number 153.
it it is armstrong(number):
```

print(f" (number) is an Armstrong number:")

else:

print (f" (number) is not an Armstrong number.").

Output: 153 is an Armstrong Number.

Result: The Code was executed Successfully.

3.

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

Algorithm:

→Define the GCD function:

•Function 'ged(a,b)

•Input: Two integers 'a' and 'b'.

•Output: The Goo of 'a' and 'b'

- If'b' is O, return'a' (base case)
- Otherwise, recursively call' gcd(b, ab).

Time Complexity:

•Time Complexity of the this algorithm is O. (log (min (a,b))), where a and bo are the input numbers. This is because the size of the number, reducy roughly by half with each recursive call.

Program in python:

```
det gcd (a,b):
    if b == 0;
       return a
   else:
       return gcd (b, a % b)
a=64
b = 80
result = gcd (a, b) print (f" The GCD of (a) and (b) is (result). ").
```

Output: The GCD of 64 and 80 is 6.

Execution time". 2.113970246 Seconds.

Result: The code was Executed Successfully. Execution time". 2.113970246 Seconds.

4.

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

Algorithm:

Initialize a Variable' max-element with the Value of the first element in the array.

- Iterate through the array starting from the Second element:

If the current element is greater than 'max_element' Update 'max_element' with the current element.

- After the loop finishes, 'mark_element will hold the largest element in the array.

Time Complexity:- O(n).

Program in python:

```
det find-largest-element (arr):
   if not arr:
      raise ValueError("The Array is empty").
max-element = arr[0]
for element in arr[1:]:
if element > max_element.
max_element = element
return max_element.
array = [3, 5, 7, 8,2,1,4].
largest-element = find-largest-element (array)
print ("The largest element in the array is", largest-element).  end time = time.time()
execution_time = end time - start time
print (Execution time:", execution-time, "seconds")
```

Output: The longest element in the array is:8

Execution time 0.00145798 seconds.

Result: The code was executed Successfully.

5.

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

Algorithm:

- Define a base case: If the number is O or 1, return 1.
- For any other number n, return multipliedn by the factorial of (n-1).

Time Complexity: 0(n)

Program for python:

```
def factorial(n):
if n== 0 or n==1:
    return 1
else:
    return n* factorial (n-1).
number = 6
result = factorial (number)
print(f" The factorial of {number} is: {result}")
     End time = time.time()
execution time = end time_ Start time.
print("Execution time:", execution_time,"Seconds")
```

Output: The factorial of 6 is: 720

Execution Time: 3.005421376 Seconds

Result: The Code was executed successfully.

6.

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

Algorithm:

- Define a recursive function that takes two. Parameters: the Original string and an index.
- If the indeve is Equal to the length of the Original String, return empty string. an
- Otherwise, Concentrate the character at the Current index with the result of the recur -sive call to the function with the indexe
- Call this function Starting with the Original string and index 0.

Time Complexity: O(m)

Here m is the length of string S1.

Program for Python:

```
import time
print("Enter the string: ")
textone = input()
Start-time = time.time()
text Two = “ ”
```

```
for x in textone:
    textTwo += x
end.time = time.time()
execution_time end_time start_time
print ("\n Original String = ", text one)
print ("\n Copied String = = ", ", text Two)
print (f" \n Time of execution: {execution_time: 10f}seconds")
```

Output:

```
  Enter a string: hello.
  Original string = hello.
   Copied string = hello.
   Time of execution: 0.0000116825 Seconds.
```

Result: The code was executed Successfully.

7.

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

Algorithm:

- Define a recursive function that takes two parameters: the Original string and an index
- If the inder is less than O, return an empty string (base case)

Otherwise, Concentrate the character at the current indere with the result of the..... recursive call to the function with the previous index.

- Call this function starting with the Original String and the last index of the string.

Time Complexity:-O(n)

Program for python:

```
import time
def reverse_string_recursively (original, index):
if index < 0:
    return “ “
return Original [index] + reverse_string_recursively (original,index-1)
Original_string = "Hello, world!"
Start time = time.time()
reversed_string = reverse_string_recursively (Original) string, len (original string)-1)
```

end_time = time. time()

execution time = end-time-start_time

print("Original String:", Original_string)

print("Reversed string:", reversed_string)

print (f" Time of execution: {execution_time: 10f} Seconds”).

Output:

Original String: Hello, world!

Reversed string! dirow, olleH

Time of execution: 0000066757 Seconds.

Result: The code was executed successfully.

8.

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

Algorithm:

- Define Check if a recursive function 'is_prime to a number & prime.
- Define a recursive function' generate_primes to generate prime numbers upto a given limit.
- The generate primes' function will Check if the current number is greater than the limit, if so, return an empty list (baye Case).

• Use the is prime function to check it the, Current number is prime to sell

• If the current number is prime, include it in the list and call the generate_primes function with the next number.

• If the Current number is not prime, call the 'generate primes' function with the next number without including the current number.

Program for python:

```
def is_prime(n, i=2):

if n <= 2:

return True if n == 2 else False

if n % i == 0:

return False

if i * i > n:

return True

return is_prime(n, i + 1)

def generate_primes(n):
```

```
if n > 1:

generate_primes(n - 1)

if is_prime(n):

print(n)

n = int(input("Enter a number: "))

print(f"Prime numbers up to {n}:")

generate_primes(n)
```

input:n=10

Output: prime numbers upto 10 are 1,3,5,7

Time Complexity: the time complexity of this Method is O(N) as we are traversing almost N numbers in case the number is prime.

Execution Time: 3.6001205444335942-05 seconds.

Result: The code way executed Successfully.

9.

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

Algorithm:

Base Case:

If n is less than or equal to 1, return False (1 and numbers less than 1 are not prime).

If current_divisor is greater than the square root of n, return True (no divisors found means n is prime).

Recursive Case:

If n is divisible by current_divisor, return False (not a prime number).

Otherwise, recursively check the next divisor by calling the function with current_divisor + 1.

Program in python:

```
def is_prime(num, div=2):

if num <= 2:

return num == 2

if num % div == 0:

return False

if div * div > num:

return True

return is_prime(num, div + 1)
```

```
num = int(input("Enter a number: "))

if is_prime(num):

print(num, "is a prime number")

else:

print(num, "is not a prime number")
```

input n=5

Output: 5 is a prime number

TIME COMPLEXITY :

O(sqrt(n)) where n is the input

10.

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

Algorithm:

Base Case:

If the length of the string is 0 or 1, return True (since an empty string or a single character string is a palindrome).

Recursive Case:

Check if the first and last characters of the string are the same.

If they are, make a recursive call with the substring that excludes these two characters.

If they are not, return False

Program:

```
def is_palindrome(s):

s = s.lower().replace(" ", "")

if len(s) <= 1:

return True

if s[0] != s[-1]:

return False

return is_palindrome(s[1:-1])

input_string = "A man a plan a canal Panama"

if is_palindrome(input_string):

print(f"{input_string} is a palindrome.")
```

else:

print(f"{input_string} is not a palindrome.")

Output: a man a plan a canal panama is a palindrome

Time complexity: O(n) where n is the input size