Saveetha School of Engineering.

SIMATS Chennai, TamilNadu.

Name:R.Maruthi.

Reg.no: 192325055.

Course code:CSA0867

Couse Name: DAA[Design and Analysis of Algorithm].

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 = 10

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

2

3

5

8

13

21

34

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 pyton:

det Sum of powers (n, power):

#Base Case: if n is O, return 0

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: It is Armstrong Number.

Result: The Code was executed Successfully.

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

Agorithm:

→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.

Pogram in python:

det gcd (a,b):

if b == 0;

return a

else:

return gcd (b, a % b)

a=48

b = 18

result = gcd (a, b) print (f" The GCD of (a) and (b) is (result). ").

Output: The GCD of 48 and 18 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:

det factorial(n):

if n== o or n==1:

return 1

else:

return n\* factorial (n-1).

number = 5

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 5 is: 1200"

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: Maruthi.

Original string = Maruthi.

Copied string = Maruthi.

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.

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

Program for python:

import time

def Prime-Number (n, i = 2):

if n<=1:

return False

if n == i:

return True

if n% i == 0:

return False

if i\*i > n:

return True

return Prime-Number (n. i+1)

n = 971

start\_time = time.time()

if Prime-Number (n):

print ("Yes, ", n, "is Prime")

else:

print ("No, ", n, "is not a prime)

end-time time. time()

execution\_time = end\_time – start\_time

print (f” Execution Time: {execution\_time} seconds”)

Output: Yes, 971 is prime

Execution Time: 3.6001205444335942-05 seconds.

Result: The code way executed Successfully.

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

Algorithm:

1. Defina a recursive function "is\_prime\_recursive that takes two parameters: the number'n' and a divisor 'div'

2. If 'n' is less than 2, return 'false' (base case for non-Prime numbers).

3. If 'div' is greater than the Square root of'n;

return 'True' (base case for prime numbers).

4. If 'n' is divisible by 'div'; return False (basecase for non -Prime Numbers).

5. Otherwise, recursively call' is\_prime\_recursive with the next divisor ('div+1').

Time Complexity: O(n²)

Program for python:

import time

import math

det is prime recursive (n, div=>):

if n < 2:

return False

if div> math. is qrt (n):

return True

if n % div = 0.

return False

return is-prime- recursive (n, div +1)

number =29

start\_time = time. time()

is\_prime = is\_prime\_recursive (number)

end\_time = time. time()

execution\_time = end\_time start\_time

if is\_prime:

print(f" {number} is a prime number.")

else:

print (f" {number} is not a prime number")

print (f" Execution time: {execution\_time} seconds”)

Output:-

29 is a prime number.

Execution time: 1.4305114746093752-09 seconds.

Result: The code was executed successfully.

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

Algorithm:

1. Define a recursive function' is\_palindrome that takes a string 's' and Optional indices 'left' and and 'right'.

2. If' left is greater than or Equal to "right return "True (base Case: the entire string has been checked).

3.If the characters at 'left' and 'right" are not the same, return 'false'.

4. Otherwise, recursively Call' is\_palindrome with "left +1' and 'right-1.

Time Complexity: O(N)

Program for python:

import time

def is\_palindrome\_recursive (s, start, end):

it start >= end:

return True

it s[start]!= s[end]:

return false

return is palindrome\_recursive (s, start+1, end-1)

input\_string = "madam"

Start\_time = time.time()

is\_palindrome = is\_palindrome\_recursive (input\_string, O, len (input\_string)-1)

end\_time = time.time()

execution\_time = end\_time\_start\_time

if is\_palindrome:

print (f"" {input\_string]' is a palindrome.")

else print(f" {input\_string}' is not a palindrome.")

print (f" Execution time: {execution\_time's seconds")

Output:-

'madam' is a palindrome.

Execution time: 2.6226043701171875e-06 second.

Result:- The Code was executed Successfully.