Pablo Emilio Lopez Avila Matemáticas Computacionales matricula 1740087

Durante esta etapa de la materia nos enfocamos a dos aspectos importantes o que en su momento fueron descubrimientos muy importantes; La series de Fibonacci y Los números primos.

Para la series de Fibonacci vimos dos códigos, los cuales más adelante pondré; pero hay que hablar un poco de que sucede con las Series de Fibonacci.

Leonardo de Pisa (Fibonacci): Era un matemático italiano que nació en 1175 y murió en 1240, él era reconocido por el patrón que creo (Serie) que se obtenía mediante las parejas de "conejos" que se iban multiplicando al pasar los meses. El problema era algo así Leonardo tiene una pareja de conejos pequeños pasa un mes para que se reproduzcan, entonces segundo mes se embaraza la conejita pero no los tiene hasta el 3er mes, en ese tercer mes tiene dos parejas una bebe y otra grande los cuales para el cuarto mes tienen 3 parejas los bebes que tuvieron los más grandes, los medianos que ya son capaces de reproducirse y finalmente la nueva parejita. De Esta manera iba creciendo su sucesión hasta el infinito dependiendo del tiempo;

Su sucesión, cada término se obtenía sumando los dos anteriores.

1, 1, 2, 3, 5, 8, 13,21…….

Tenemos dos códigos para obtener el "Fibonacci" de un mes exacto.

El Fibonacci simple

```
cnt=0 #contador de operaciones

def fibonacci(n): #Definimos nuestra funcion
    global cnt
    cnt=cnt+1  #añadimos una unidad al contador
    if n==0 or n==1:
        return(1)  #El fibonacci de 1 o 0 es 1, ya que no hay términos anteriores a ellos
    return  fibonacci(n-2) + fibonacci(n-1) # si no es el Fibonacci de 0 o 1, se sumaran los dos anteriores términos del Fibonacci en este caso se hará esto hasta que se reduzcan a los mínimos términos
```

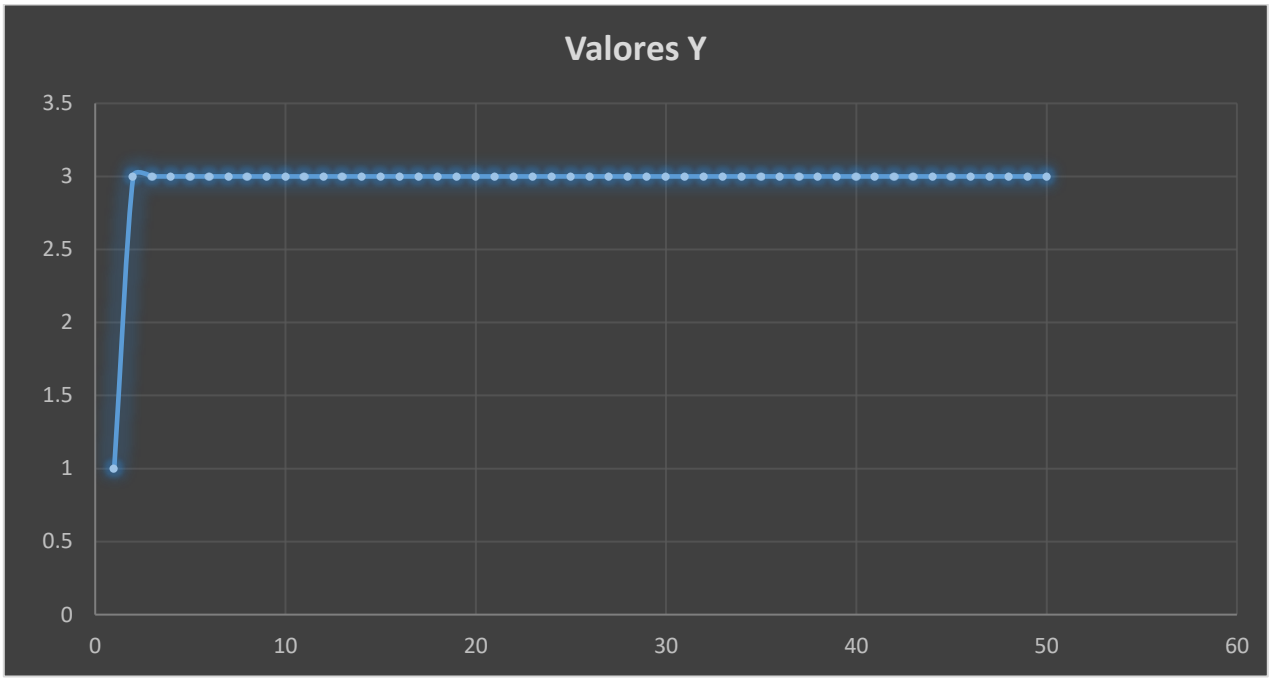
Para mostrar el comportamiento de este algoritmo usaremos una repetición para los primeros 50 términos del Fibonacci contando cuantas acciones hace cada uno, mediante el siguiente ciclo:

```
for i in range (1,51):
    cnt=0 #quita números basura de la variable cada vez que entremos al ciclon
    print(cnt,fibonacci(i)) #imprime el contador y el fibonacci de i
```

Otro algoritmo para Fibonacci es :

```
funcion={}
cnt=0
def fibonacci(n):
    global funcion,cnt #lo utilizaremos en todo el programa
    cnt+=1 #añadimos una unidad
    if n==0 or n==1: #Sabemos que el termino 0 o 1 su fibonacci es 0
        return(1)
    if n in funcion:
        return funcion[n] #utilizamos memoria para almacenar los datos.
    else:
        val=fibonacci(n-2)+fibonacci(n-1) #variable temporal almacena los datos
        funcion[n]=val #valor de la función de n es el valor del fibonacci de n
        return val Regresa el valor del fibonacci de n
for i in range (1,51):
    print(i,cnt,fibonacci(i))
```

Para comprobar la eficacia d este algoritmo probaremos con los primeros 50 términos como el primer Fibonacci.



Conclusión-Ganador.

La diferencia entre los dos códigos es abismal, ya que el primer código, que va haciendo los cálculos uno por uno sin guardar nada, se tarda más de una hora, ya que llevaba 40 minutos o más y apenas iba en el término 40 de la sucesión, mientras que el segundo término que almacenaba en la función algunos valores en cuestión de 10 segundos o menos ya termino todo el proceso.

El otro código es el de Números primos:

Por definición de un número primo, es un número que únicamente es divisible por el 1 y por sí mismo: Pero ¿Cómo representarlo en un algoritmo? No es tan sencillo, pero se elige una función la cual va dividir el numero dentro de un ciclo desde el numero 2 hasta la raíz del número, de esta manera podremos saber si ese tal número es divisible por otro número que no sea el mismo, el 1 lo descartamos ya que todo número es divisible por 1,

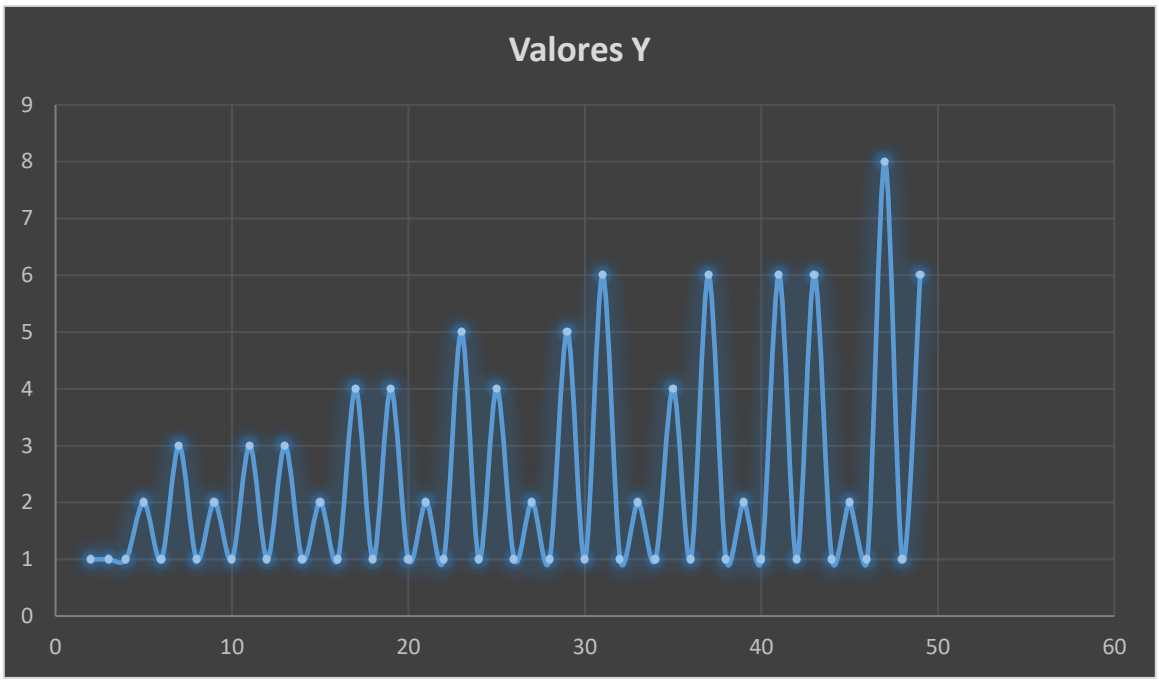
Mediante este código puedes determinar los numero primos que existen para los primeros 50 términos y también el número de operaciones que realiza para que se determine.

Conclusión de primos:

Mediante este código en alrededor de 10 segundos es posible determinar los primos, y de una manera eficiente, mostrando una alta calidad del algoritmo de baja complejidad.

```
cnt=0
def primo(n):
    global cnt
    for i in (2,round((n**(1/2)+1))):  #es suficiente probar hasta n a la ½ para saber que es primo
        cnt=cnt+1
        if ((n%i)==0): %si el residuo de alguna división es 0, no es primo.
            print("No Primo") #salida
            break

        else:
         print("Primo")
         break
    return n,cnt
for i in range(1,51): #Ciclo que imprime los primeros 50 numeros y sus contadores para ver si son primos o no
    cnt=0
    print(i,primo(i))
```


Valores Y
9
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60