Análisis de Algoritmos, Sem: 2018-1, 3CV2 Práctica 2, 31 de Agosto del 2017

Práctica 2: Funciones Recursivas vs Iterativas

Luis Daniel Martinez Berumen

Escuela Superior de Computo
Instituto Politecnico Nacional
dany.berumen@gmail.com

Resumen

En la presente practica vamos a analizar, las diferencias entre funciones recursivas vs funciones iterativas, implementando cada una de ellas al mismo problema conocido como la serie de Fibonacci demostrando formalmente el porque este algoritmo es de orden lineal en su forma iterativa, ademas de propone una funcoin g(n) tal que T(n) $\in$ O(g(n)) con T(n) a partir de graficas de un algoritmo que recibe un entero positivo n y nos regresa la sumna de los primeros n cubos.

Palabras Clave

- Algoritmo
- Fucion
- Recursividad
- Iteracion

1. Introducción

Los algoritmos recursivos e iterativos son la base de esta practica, es importante analizar estos 2 aspectos de suma importancia en la programacion, las funciones recursivas e iterativas,
estas a pesar de ser aplicables a un mismo problema muchas veces una de estas es mas eficiente, es por esto que debemos de probar con ambas y aplicar algun enfoque como el de Divide y venceras que evidentemente es mejor cuanto mas grande es el caso.
Lo cierto es que puede resultar mas lento que el algoritmo clasico en casos que sean demasiado pequenos. por tanto el algoritmo de divide y venceras debe de evitar seguir avanzando recursivamente cuando el tamaño de los casos no lo justifique.

2. Conceptos Básicos

Para la correcta comprension de este trabajo, es necesario definir algunos terminos tales como θ, O y Ω.
θ(n):
Sea g(n) una función. Se define θ (g(n)) como:
θ(g(n)) = $\{f(n) \quad | \quad \exists c1, c2 > 0 \quad \& \quad n_0 > 0 \quad | \quad \forall n >= n_0 \quad 0 <= c1g(n) <= f(n) <= c2g(n)\}$
O(n):

Sea g(n) una función, O(n) se define como:
O(n)=$\{f(n) \quad | \quad \exists c > 0 \quad \& \quad n_0 > 0 \quad | \quad f(n) <= Cg(n) \quad \forall n >= n_0\}$ Ω(n):

Sea g(n) una función. Se define Ω (g(n)) como:
Ω(g(n)) =$\{f(n) \quad | \quad \exists c > 0 \quad \& \quad n_0 > 0 \quad | \quad 0 <= cg(n) <= f(n) \quad \forall n >= n_0\}$
Iteracion:

acto de repetir un proceso con la intención de alcanzar una meta deseada, objetivo o resultado. Cada repetición del proceso también se le denomina una ïteración”, y los resultados de una iteración se utilizan como punto de partida para la siguiente iteración. Recursividad:
Es la forma en la cual se especifica un proceso basado en su propia definición.(ver Recursividad)
Realizaremos el analisis de 2 algoritmos en forma de su orden de complejidad programando tal

algoritmos y segun el inciso verificaremos su orden de manera iterativa o recursiva,
ademas de un tercer ejercicio de manera teorica.
Nuestro primer algoritmo es la sucesion de Fibonacci, recordemos que los terminos de esta secuencia se definen mediante la recurrencia siguiente:

0 1 1 2 3 5 8 13 21
34 55 89 144 233 377
610 987 1597 2584....

Figura 1

Esto debido a que consiste en tomar 2 enteros proximos entre si e ir sumando el resultado con el contiguo predecesor del resultado, nuestro algoritmo recibe n que es el numero de los primeros n dijitos de la sucesion que deseamos ver o calcular.

El siguiente algoritmo nos habla de la suma de los primeros n numeros naturales en su potencia cubica, de manera iterativa y recursiva, nuestro codigo pide un numero n que sera el numero de elementos tomados en la suma.

3. Experimentación y Resultados

3.1. Fibonacci

3.1.1. Pseudocódigo de Fibonacci Recursivo

fiboRecu(n):
Entrada: un entero positivo n
Salida: El numero de la serie de Fibonacci con la posicion n
1.- if n==1 or n==2:
2.- return 1
3.- else:
4.- return fiboRecu(n-1) + fiboRecu(n-2)

3.1.2. Pseudocódigo de Fibonacci Iterativo

fiboIteracion(n):
Entrada: un entero positivo n
Salida: El numero de la serie de Fibonacci con la posicion n
1.- a ← 1
2.-b ← 1
3.-for i=0 to i¡n do:
4.- a ← b
5.- b← a + b
6.-return a

ii)Demostrar de manera formal que el algoritmo de fibonacci iterativo tiene orden lineal.

fiboIteracion(n)	Costo	Pasos
1.- a ← 1	C1	1
2.- b ← 1	C2	1
3.- for i=0 to i<n do:	C3	n
4.- a ← b	C4	n-1
5.- b← a + b	C5	n-1
6.- return a	C6	1

ii)Demostracion. Tenemos que:
T(n) = C1 + C2 + C3n + C4(n-1) + C5(n-1) + C6
‘ T(n) = C1 + C2 + C3n +C4n -C4 + C5n - C5 + C6
Al factorizar nos queda:
T(n) = (C3+C4+C5)n + (C1+C2-C4-C5+C6)
$\therefore$
T(n) $\in \theta$(n)

Mostrar con graficas que la proposicion anterior es cierta.

Tomando valores desde 1 a 20 nos da:

```
C:\Python27\python.exe "D:/ESCOM/5to/analisis de algoritmos/Practica 2/EJ1.py"
Para 0  Fibo ite entra: 0
Para 1  Fibo ite entra: 1
Para 2  Fibo ite entra: 2
Para 3  Fibo ite entra: 3
Para 4  Fibo ite entra: 4
Para 5  Fibo ite entra: 5
Para 6  Fibo ite entra: 6
Para 7  Fibo ite entra: 7
Para 8  Fibo ite entra: 8
Para 9  Fibo ite entra: 9
Para 10  Fibo ite entra: 10
Para 11  Fibo ite entra: 11
Para 12  Fibo ite entra: 12
Para 13  Fibo ite entra: 13
Para 14  Fibo ite entra: 14
Para 15  Fibo ite entra: 15
Para 16  Fibo ite entra: 16
Para 17  Fibo ite entra: 17
Para 18  Fibo ite entra: 18
Para 19  Fibo ite entra: 19

Process finished with exit code 0
```

Figura 2

La grafica para esta tabla es:

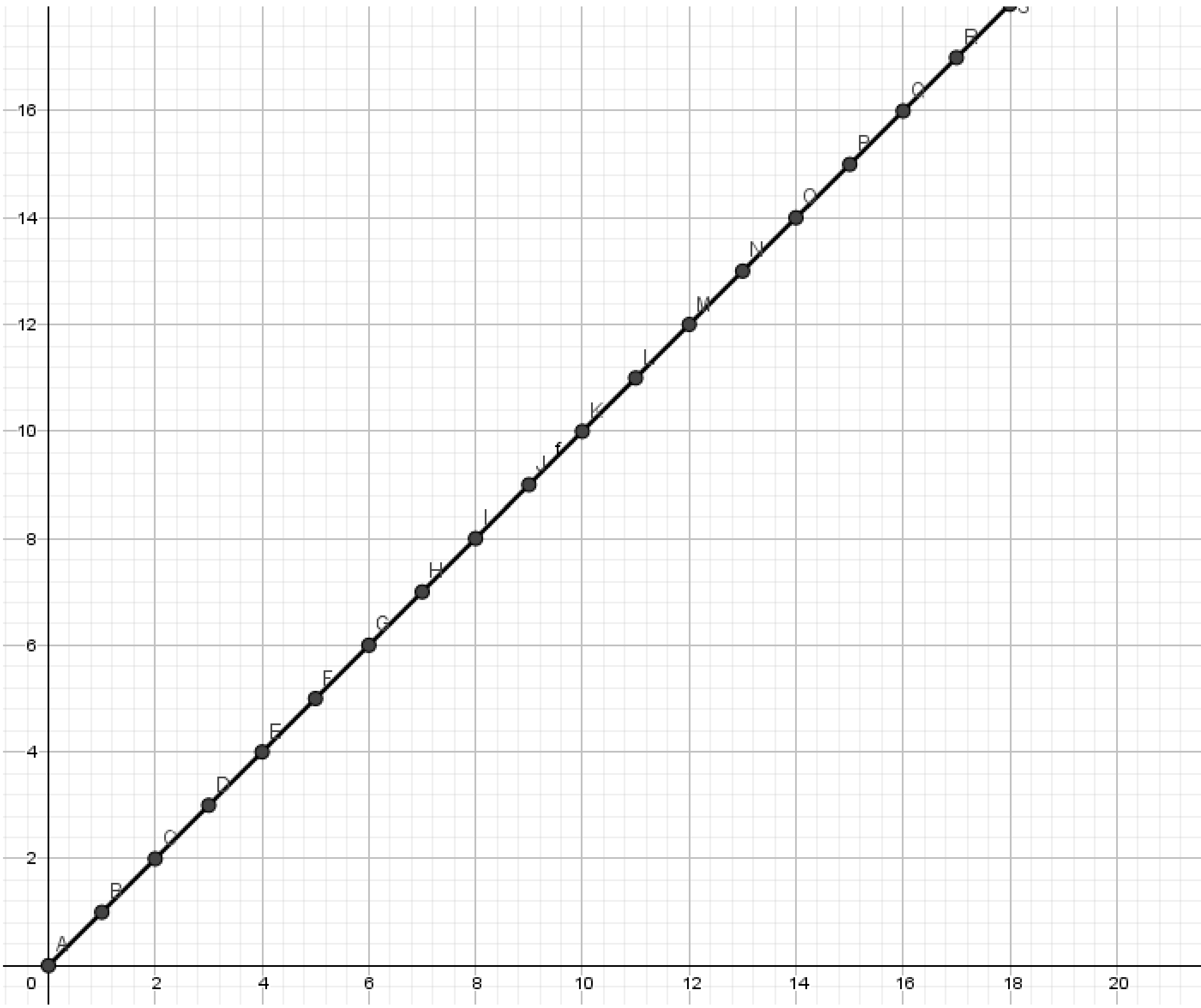


Figura 3

Podemos observar que la grafica para los valores obtenidos en la ejecucion del algoritmo muestra un comportamiento lineal, por lo el resultado tras la ejecucion, apoya la demostracion anterior

A partir de graficas, proponer el orden de complejidad para el algoritmo recursivo.

Tomando valores desde 1 a 20 nos da:

```
C:\Python27\python.exe "D:/ESCOM/5to/analisis de algoritmos/Practica 2/EJ1.py"
Para 0 Fibo rec  1
Para 1 Fibo rec  1
Para 2 Fibo rec  3
Para 3 Fibo rec  5
Para 4 Fibo rec  9
Para 5 Fibo rec  15
Para 6 Fibo rec  25
Para 7 Fibo rec  41
Para 8 Fibo rec  67
Para 9 Fibo rec  109
Para 10 Fibo rec  177
Para 11 Fibo rec  287
Para 12 Fibo rec  465
Para 13 Fibo rec  753
Para 14 Fibo rec  1219
Para 15 Fibo rec  1973
Para 16 Fibo rec  3193
Para 17 Fibo rec  5167
Para 18 Fibo rec  8361
Para 19 Fibo rec  13529

Process finished with exit code 0
```

Figura 3

La grafica para esta tabla es:

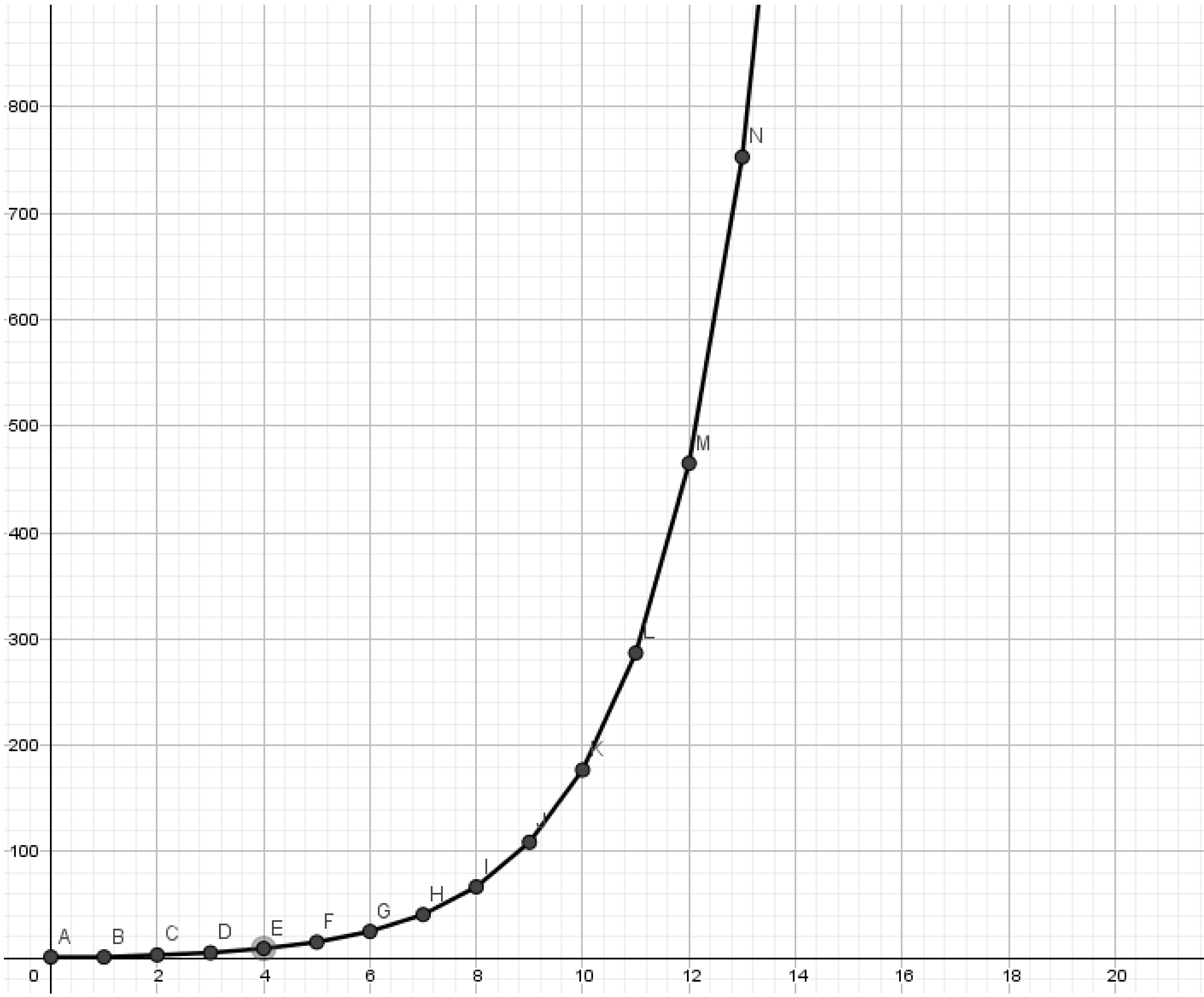


Figura 4

La grafica que nos arrojan los resultados obtenmidos en los experimentros es de tipo exponencial, por lo que se podria proponer O(n^2)

3.2. Suma Cubica

3.2.1. Apartir de graficas proponga una funcion g(n) tal que T(n) $\in$ O(g(n)) con T(n) como el tiempo computacional del algoritmo

Para los valores del 1 al 50, el algoritmo arrojo los siguientes resultados:

```
C:\Python27\python.exe "D:/ESCOM/5to/analisis de algoritmos/Practica 2/ej2.py"
 para 1   ContRec= 1 contIte = 1
 para 2   ContRec= 2 contIte = 2
 para 3   ContRec= 3 contIte = 3
 para 4   ContRec= 4 contIte = 4
 para 5   ContRec= 5 contIte = 5
 para 6   ContRec= 6 contIte = 6
 para 7   ContRec= 7 contIte = 7
 para 8   ContRec= 8 contIte = 8
 para 9   ContRec= 9 contIte = 9
 para 10   ContRec= 10 contIte = 10
 para 11   ContRec= 11 contIte = 11
 para 12   ContRec= 12 contIte = 12
 para 13   ContRec= 13 contIte = 13
 para 14   ContRec= 14 contIte = 14
 para 15   ContRec= 15 contIte = 15
 para 16   ContRec= 16 contIte = 16
 para 17   ContRec= 17 contIte = 17
 para 18   ContRec= 18 contIte = 18
 para 19   ContRec= 19 contIte = 19
 para 20   ContRec= 20 contIte = 20
 para 21   ContRec= 21 contIte = 21
 para 22   ContRec= 22 contIte = 22
 para 23   ContRec= 23 contIte = 23
 para 24   ContRec= 24 contIte = 24
 para 25   ContRec= 25 contIte = 25
 para 26   ContRec= 26 contIte = 26
 para 27   ContRec= 27 contIte = 27
 para 28   ContRec= 28 contIte = 28
 para 29   ContRec= 29 contIte = 29
 para 30   ContRec= 30 contIte = 30
 para 31   ContRec= 31 contIte = 31
 para 32   ContRec= 32 contIte = 32
 para 33   ContRec= 33 contIte = 33
 para 34   ContRec= 34 contIte = 34
```

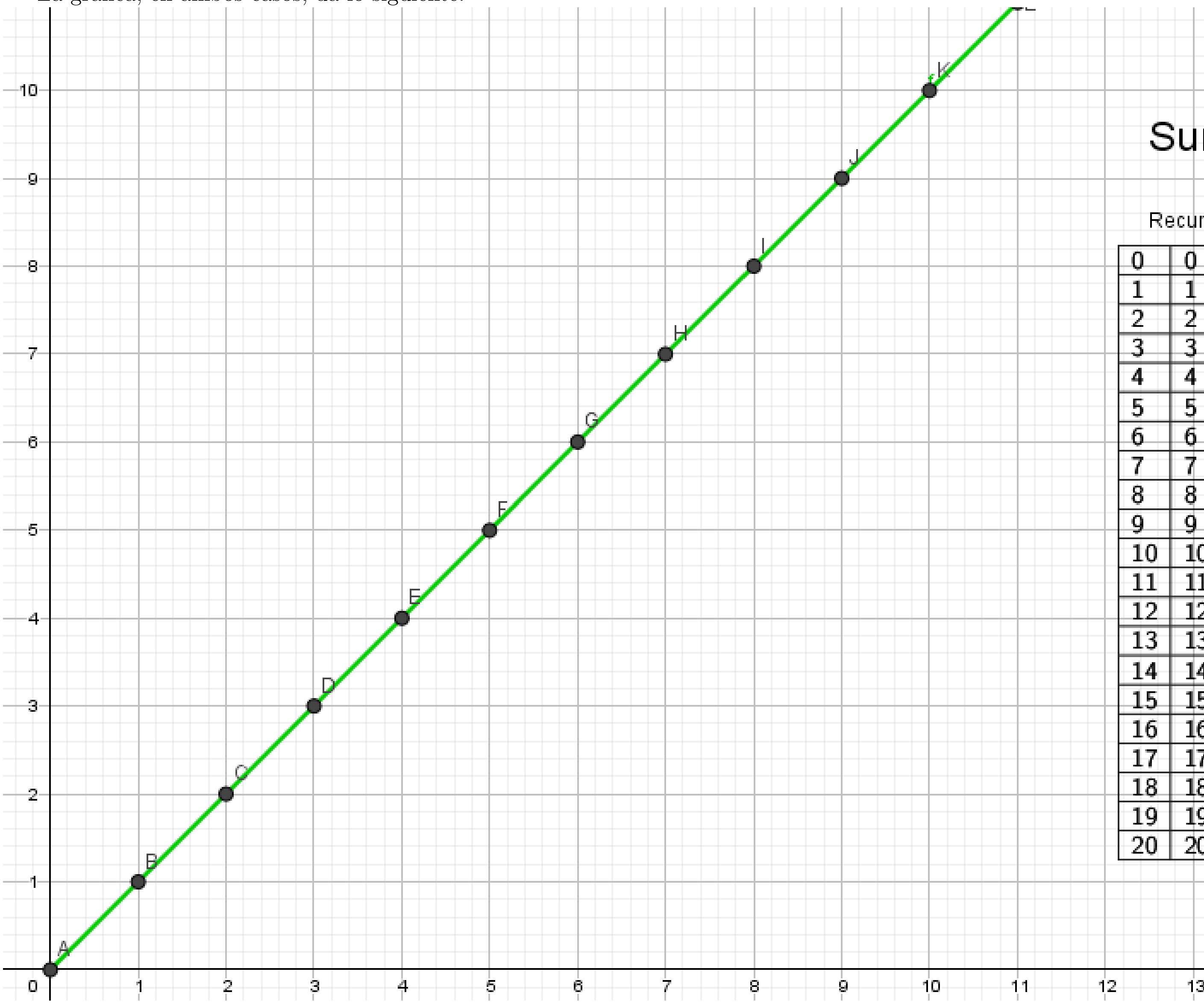
Figura 5

```
para 34   ContRec= 34 contIte = 34
para 35   ContRec= 35 contIte = 35
para 36   ContRec= 36 contIte = 36
para 37   ContRec= 37 contIte = 37
para 38   ContRec= 38 contIte = 38
para 39   ContRec= 39 contIte = 39
para 40   ContRec= 40 contIte = 40
para 41   ContRec= 41 contIte = 41
para 42   ContRec= 42 contIte = 42
para 43   ContRec= 43 contIte = 43
para 44   ContRec= 44 contIte = 44
para 45   ContRec= 45 contIte = 45
para 46   ContRec= 46 contIte = 46
para 47   ContRec= 47 contIte = 47
para 48   ContRec= 48 contIte = 48
para 49   ContRec= 49 contIte = 49
para 50   ContRec= 50 contIte = 50

Process finished with exit code 0
```

Figura 6

La grafica, en ambos casos, da lo siguiente:



-iii)Calcule el tiempo computacional del algoritmo recursivo e iterativo de manera formal. Compare sus resultados

Forma Recursiva:

Sea A(n) el número de sumas que se realizan para calcular S(n). Entonces:

A(n) =

Luego

A(n) = A(n-1) + 1

A(n) = A(n-2) + 1 + 1 = A(n-2) + 2

A(n) = A(n-3) + 1 + 2 = A(n-3) + 3

...

A(n) = A(n - (n - 1)) + n -1 = A(1) + n -1
A(n) = 1+ n -1

A(n) = n

$\therefore T(n) \quad \in \quad \theta(n)$

Forma Iterativa:

	Algoritmo S(n)	Costo	#Pasos
1.-	y ← 0	A	1
2.-	while n ⩾ 0 ← 1	B	n + 1
3.-	y ← suma y con x*x*x	C	n
4.-	n ← n - 1	D	n
5.-	return y	E	1

Con ello tenemos que:
T(n) = A + B(n + 1) + Cn + Dn + E

T(n) = (B + C + D)n + (A + B + E)

$\therefore T(n) \quad \in \quad \theta(n)$

4. Conclusión General

En esta practica pudimos analizar 2 algoritmos basicos en la programacion, esto con 2 posibles maneras de ser resultos esto es de manera iterativa y recursiva, en mi trayectoria axademica claro que habia desarrollado estos algoritmos de ambas formas pero nunca pense en cual podria sernos mas util o digamoslo asi computablemente viable, es por esto que para empezar me gustaria hablar del algoritmo de Fibonacci de este ser podia intuir un poco que el algoritmo iterativo nos seria mas conveniente, ya que la recursividad es conocida por ser un arma de doble filo, es poderosa cuando se sabe ocupar y detener, pero el algoritmo de la suma de los cubos me fue algo extraño de aceptar ya que los datos y las graficas no eran lo que esperaba, lo que mas ocupo mi trabajo en definitiva fue analizar los resultados ya que requerian de mas atencion que programar o graficar, pero convenientemente los resultados fueron satisfactorios y a mi consideracion de hoy en adelante seria de gran ayuda para cualquier programador un aviso cuando tengamos que programar Fibonacci nos recuerde o nos avise que es mejor programar este de manera iterativa.

5. Anexos

Algoritmo Bubble-Sort

Sea A.largo = n

BubbleSort(A)		Costo	Pasos
1.-	for i=1 to A.largo -1 do	C1	n
2.-	for j= A.largo downto i+1	C2	$\sum_{i=1}^{n-1} t_i$
3.-	if A[j] ¡A[j-1]	C3	$\sum_{i=1}^{n-1} (t_i - 1)$
4.-	exchange(A[j], A[j-1]	C4	$\sum_{i=1}^{n-1} R_i$

El Peor de los casos se presenta cuando el arreglo esta ordenado en forma decreciente, por lo que:

Para $i = 1, t_1 = n$

Para $i = 2, t_2 = n - 1$

Para $i = 3, t_3 = n - 3$

.

.

.

Para $i = k, t_k = n - k + 1$

Así $t_i = n - i + 1$

Y $R_i = t_i - 1$

Con ello, tenemos que:

T(n) = C1n + C2($\sum_{i=1}^{n-1}(n - i + 1()$ + C3($\sum_{i=1}^{n-1}(n - i)$) + C4($sum_{i=1}^{n-1}(n - i)$)

T(n) = C1n + C2n($\sum_{i=1}^{n-1} n$) - C2($\sum_{i=1}^{n-1} i$) + C2($\sum_{i=1}^{n-1} i$) - C3($\sum_{i=1}^{n-1} n$) -C3($\sum_{i=1}^{n-1} i$) + C4($\sum_{i=1}^{n-1} n$) - C4($\sum_{i=1}^{n-1} i$)

T(n) = C1n + (C2+C3+C4)($\sum_{i=1}^{n-1} n$) - (C2+C3+C4)($\sum_{i=1}^{n-1} i$) + C2(n-1)

T(n) = C1n + (C2+C3+C4)(n(n-1)) - (C2+C3+C4)($\frac{n(n+1)}{2}$) + C2(n-1)

T(n) = (C1 + $\frac{C2 + C3 + C4}{2}$)n + ($\frac{C2 + C3 + C4}{2}$)n^2

Por lo tanto, tenemos:

T(n) ∈ O(n^2)

El mejor de los casos se da cuando el arreglo ya esta ordenado de forma creciente.

En dicho caso, lo unico que hay que cambiar es la condicion del if, ya que nunca se cumple, por lo tanto:

$R_i = 0$

Por lo tanto:

T(n) = C1n + C2($\sum_{i=1}^{n-1}(n - i + 1)$) + C3($\sum_{i=1}^{n-1}(n - i)$)

T(n) = (C1+$\frac{C2 + C3}{2}$)n + ($\frac{C2 + C3}{2}$)n^2

$\therefore$

T(n) ∈ $\Omega(n^2)$

$\therefore$

T(n) ∈ $\theta(n^2)$

6. Bibliografía

- Brassard, G. (1997). Fundamentos de Algoritmia. España: Ed. Prentice Hall. ISBN 848966000X

- Harel, D. (2004). Algorithmics: The spirit of Computing (3rd. Ed). Estados Unidos de América: Addison Wesley. ISBN-13: 978-0321117847