



# Instituto Politécnico Nacional

## Escuela Superior de Cómputo

Análisis de Algoritmos, Sem: 2021-1, 3CV1, Práctica 1,  
17-10-2020

### Práctica 1: Determinación experimental de la complejidad temporal de un algoritmo.

De la Cruz Sierra Diana Paola

Raya Chávez Samuel Antonio

dianapaodcs@gmail.com, samiraya1323@gmail.com

**Resumen.** Se presentarán 2 algoritmos, los cuales serán analizados para obtener su tiempo de ejecución aproximado, y sus respectivas gráficas para visualizar dicho análisis.

**Palabras clave.** Algoritmo, Suma binaria, Euclides, C++

# 1. Introducción

Los algoritmos son una secuencia lógica, detallada y ordenada de pasos que tiene como objetivo resolver un problema. Las propiedades de un algoritmo son las siguientes:

- El algoritmo debe ser preciso e indicar el orden de realización de cada paso.
- El algoritmo debe ser definido, si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- El algoritmo debe ser finito, si se sigue un algoritmo se debe terminar en algún momento; o sea debe tener un número finito de pasos.

Los algoritmos son generalmente implementados en las computadoras mediante lenguajes de programación de alto nivel, que no son directamente ejecutables, pero que sí son “compilables”.

Un programa, por otro lado, es un algoritmo que ha sido codificado en algún lenguaje de programación.

El análisis de algoritmos se ocupa de comparar algoritmos con base en la cantidad de recursos computacionales que utiliza cada algoritmo. Queremos ser capaces de considerar dos algoritmos y decir que uno es mejor que el otro, porque es más eficiente en su uso de esos recursos o simplemente tal vez porque utiliza una menor cantidad de tiempo. En este sentido, para determinar el rendimiento de un algoritmo se deben considerar dos aspectos:

- Cantidad de datos de entrada a procesar.
- Tiempo necesario de procesamiento.

El tiempo de ejecución depende del tipo de datos de entrada, clasificados en tres casos:

- Caso optimo (mejor caso): datos de entrada con las mejores condiciones. Ejemplo: que el Caso óptimo conjunto de datos se encuentre completamente ordenado.
- Caso medio: Conjunto estándar de datos de entrada. Ejemplo: que el 50 % de los datos estén ordenados y el 50 % restante no.
- Peor caso: Datos de entrada más desfavorable. Ejemplo: que los datos se encuentren completamente desordenados.

Diversos problemas requieren algoritmos diferentes. Un problema puede llegar a tener más de un algoritmo que lo solucione, más la dificultad se centra en saber cuál está mejor implementado, es decir, el que, dependiendo del tipo de datos a procesar, tenga un tiempo de ejecución óptimo.

## 2. Conceptos básicos

Algunos conceptos a enfatizar son:

- **Notación Big O (O).** Esta determina una cota superior en la tasa de crecimiento de una función, dentro de un factor constante.
- **Notación Omega ( $\Omega$ ).** Esta determina una cota inferior en la tasa de crecimiento de una función, dentro de un factor constante.
- **Notación Theta ( $\Theta$ ).** Esta determina una cota superior e inferior en la tasa de crecimiento de una función, dentro de un factor constante. (Fillottrani P., 2017).

Para verlo mejor, se presentarán unas gráficas que representen lo dicho.

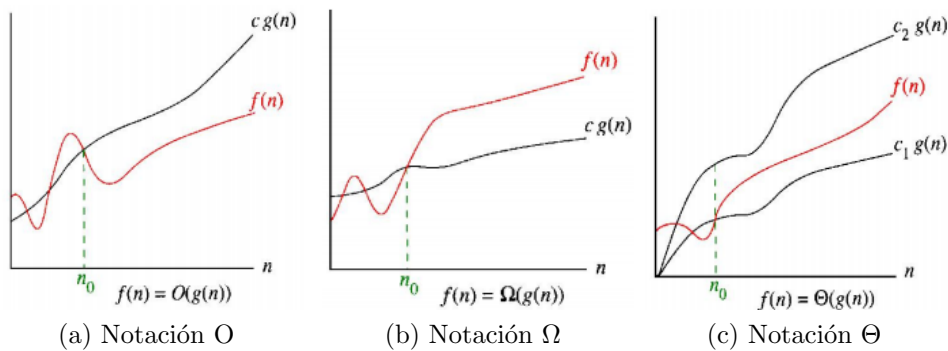


Figura 1: Las diferentes gráficas de las notaciones. (Fillottrani P., 2017).

A continuación se mostraran los pseudo-códigos de los algoritmos para la suma binaria y el algoritmo de Euclides.

Para este primer algoritmo, se va a realizar una suma binaria unidimensional entre dos cadenas de tamaño  $m$  y  $n$ .

---

**Algorithm 1** Pseudódigo: Suma binaria

---

```
1: procedure SUMA( $A[r], B[r], r$ )
2:   for  $i \leftarrow i - 1$  to  $i \leftarrow 0$  do
3:     if HarCarry then
4:       if  $A[i] == 0$  then
5:         if  $B[i] == 0$  then
6:            $C[i] \leftarrow 1$ 
7:           HayCarry  $\leftarrow false$ 
8:         else
9:            $C[i] \leftarrow 0$ 
10:        end if
11:      else
12:        if  $B[i] == 0$  then
13:           $C[i] \leftarrow 0$ 
14:        else
15:           $C[i] \leftarrow 1$ 
16:        end if
17:      end if
18:    else
19:      if  $A[i] == 0$  then
20:        if  $B[i] == 0$  then
21:           $C[i] \leftarrow 0$ 
22:        else
23:           $C[i] \leftarrow 1$ 
24:        end if
25:      else
26:        if  $B[i] == 0$  then
27:           $C[i] \leftarrow 1$ 
28:        else
29:           $C[i] \leftarrow 0$ 
30:          HayCarry  $\leftarrow true$ 
31:        end if
32:      end if
33:    end if
34:  end for
35:  if HayCarry then
36:    Agregar un 1 al inicio de C
37:  end if
end procedure
```

---

Para este algoritmo se va a desarrollar un algoritmo de Euclides que calcule m.c.d de dos números  $m$  y  $n$ .

---

**Algorithm 2** Pseudódigo: Algoritmo de Euclides

---

```

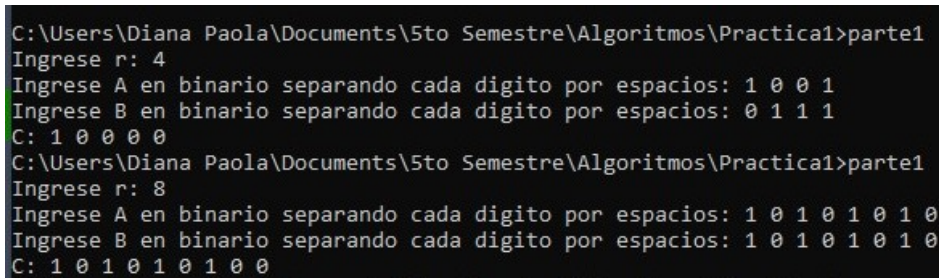
1: procedure EUCLIDES( $m, n$ )
2:   while  $n \neq 0$  do
3:      $k \leftarrow m \% n$ 
4:      $m \leftarrow n$ 
5:      $n \leftarrow k$ 
6:   end while
7: end procedure

```

---

### 3. Experimentación y resultados

#### 3.1. Algoritmo 1: suma binaria



```

C:\Users\Diana Paola\Documents\5to Semestre\Algoritmos\Practical\>parte1
Ingrese r: 4
Ingrese A en binario separando cada dígito por espacios: 1 0 0 1
Ingrese B en binario separando cada dígito por espacios: 0 1 1 1
C: 1 0 0 0 0
C:\Users\Diana Paola\Documents\5to Semestre\Algoritmos\Practical\>parte1
Ingrese r: 8
Ingrese A en binario separando cada dígito por espacios: 1 0 1 0 1 0 1 0
Ingrese B en binario separando cada dígito por espacios: 1 0 1 0 1 0 1 0
C: 1 0 1 0 1 0 1 0 0

```

Figura 2: Prueba1

En la figura 2 se pueden observar dos ejemplos con  $r$  siendo potencia de 2, de dos números en notación binaria.

Haciendo un análisis Aposteriori del peor caso en el algoritmo 1 se obtiene que:  $T(n) = 7r + 6$

Esto debido a que: la declaración de las variables se ejecuta 1 vez. Para el for se divide en inicialización, condición y actualización que se ejecutan 1 vez,  $r+1$  veces y  $r$  veces respetivamente. El primer if entrante se ejecuta  $r$  veces, su if anidado se ejecuta  $r$  veces y su tercer if al tener 2 líneas se ejecuta  $r+2*r$  veces. El último if se ejecuta 2 veces ya con su cuerpo y 1 vez el return.

La tabla 1 muestra  $r$  vs  $T(n)$  y la gráfica 3 muestra la gráfica de dichos puntos, si se analiza de mejor forma la gráfica es una lineal, por lo que una buena propuesta que acote dichos puntos sería:  $10r$ .

Y mostrando que esa propuesta es valida en la figura 4 se muestra el resultado, siendo la recta roja 10r y la verde los puntos gráficosados anteriormente.

<b>r</b>	<b>T(n)</b>
2	20
4	34
8	62
16	118
32	230
64	454
128	902

Cuadro 1: Tabla puntos.

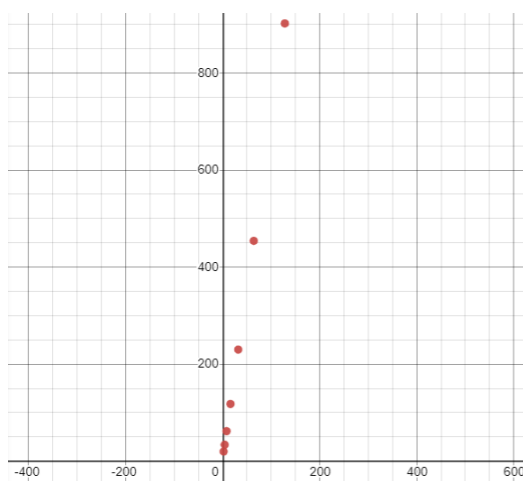


Figura 3: Puntos de r vs T(n)

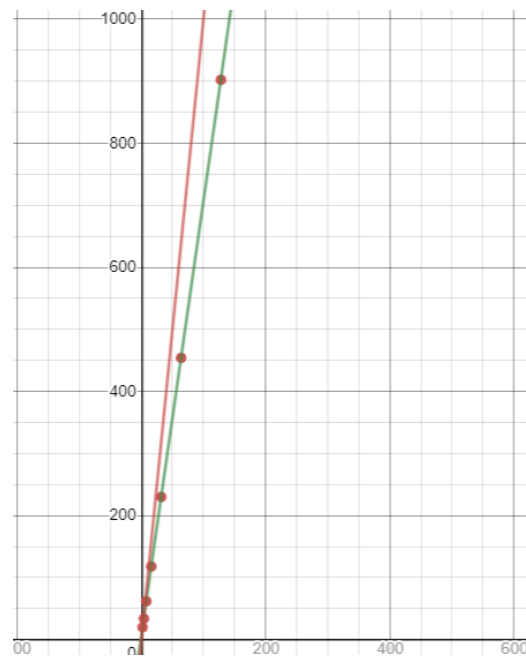


Figura 4: ecuacion propuesta  $10r$  y  $T(n)=7r+6$

### 3.2. Algoritmo 2: Algoritmo de Euclides

El algoritmo mostrará, por ejemplo, los siguientes resultados.

Input m	Input n	GCD	Execution Time (seconds)
240	199	1	27.09
138	39	3	5.914
792	45	9	8.143
534	28	2	7.622

Figura 5: Algunos resultados del programa de algoritmo de Euclides

En la figura 5 se observan cuatro ejemplos con  $n$  siendo menor que  $m$ .

Haciendo un análisis a posteriori en el algoritmo 2 se obtiene que:  
 $T(n) \leftarrow 4n-3$

¿Por qué?, bien, para el ciclo *while* se ejecuta  $n$  veces la comprobación de que  $k$  sea diferente de 0, posteriormente, las 3 sentencias dentro del *while* se ejecutan  $n - 1$  veces, se ejecuta una vez menos que la comprobación de la primera línea *while* ya que si  $k=0$ , estas sentencias no se ejecutarán.

A continuación se mostrará la tabla de los diferentes resultados establecidos con diferentes puntos arbitrarios. Se comparará  $m$ , que es el número mayor, con el tiempo de ejecución que tardará.

<b>m</b>	<b>T(n)</b>
47	13
138	13
240	21
325	25
482	9
534	5
672	13
717	17
842	21

Cuadro 2: Tabla con diferentes puntos arbitrarios.

Y graficando los puntos del cuadro 2 queda:

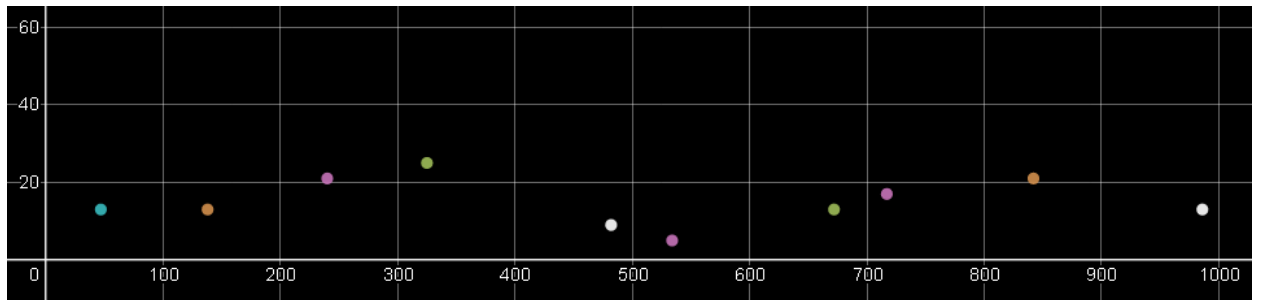


Figura 6: Gráfica de los puntos arbitrarios.

Ahora analizando los resultados del mismo algoritmo pero en el peor de los casos, que se obtiene al insertar valores consecutivos de la sucesión de Fibonacci, se tiene el siguiente cuadro:



<b>m</b>	<b>T(n)</b>
8	13
55	29
144	37
233	41
377	45
610	49
987	53

Cuadro 3: Tabla con diferentes puntos de Fibonacci.

Y graficando los puntos del cuadro 3 queda:

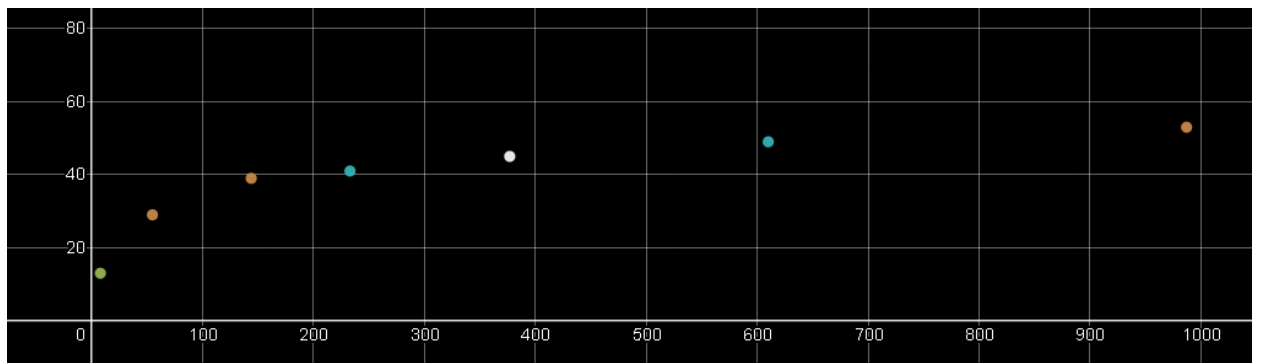


Figura 7: Gráfica de los puntos de Fibonacci.

Y visto de una mejor manera uniendo los puntos se obtiene:

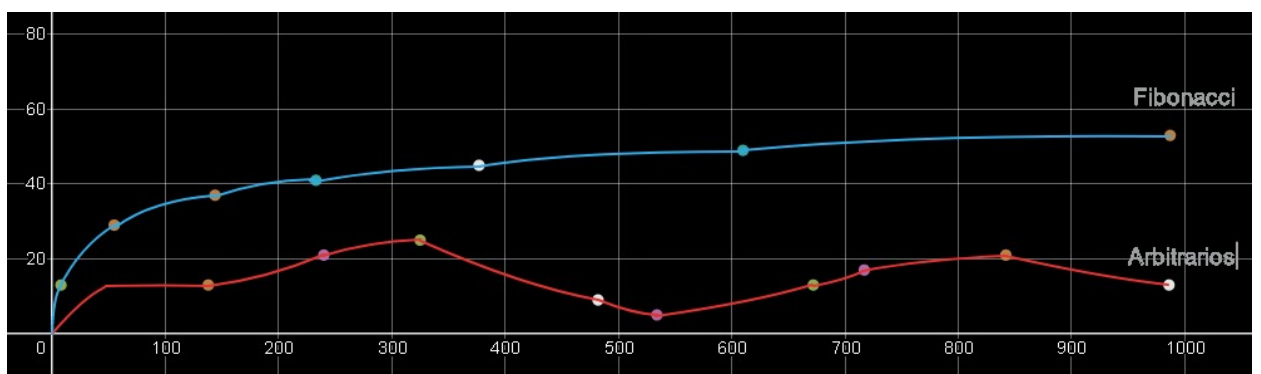


Figura 8: Gráfica del algoritmo de Euclides uniendo los puntos.

## 4. Conclusiones

Conclusiones Generales:

En la parte del algoritmo de Euclides al principio hubo más variables declaradas y más asignaciones, pero al ver que algunas de las variables eran obsoletas y que se podía optimizar de una forma mejor, quedo de una forma un poco más compacta.

También para la parte de las pruebas, se incluyó una variable que contara las veces que se realizaron las operaciones, de esta forma al momento de calcular  $T(n)$  sería de una forma más sencilla.

En el caso del algoritmo de suma se puede analizar que también hay maneras de hacer el algoritmo más compacto, sin embargo esa manera de utilizarse fue la más sencilla.

Diana Paola De la Cruz Sierra:

Análizar un algoritmo es importante, nos brinda información de los recursos que gasta y eso puede indicarnos si es ideal mejorar la manera en que se realiza, normalmente para hacerlo más rápido, ya que en espacio de memoria podría expandirse, pero para mejorar el tiempo no es posible más que haciendo el algoritmo con una complejidad más baja de la que se tiene en caso de ser posible.



Raya Chávez Samuel Antonio:

El ser humano siempre busca la forma de hacer todo de la forma más eficiente posible, así que para consumir el menor tiempo posible se

tienen que analizar las cosas para llegar al mejor resultado posible, que en este caso serían los algoritmos. Los algoritmos nos pueden ayudar desde cosas tan simples como la suma de dos números, hasta lo más complejo como la identificación de patrones de reconocimiento. Todo esto nos llega a ser útil, solo hay que encontrar la forma de darle una buena implementación.



- Adam, J., Rubio, T.(2016). *definición de algoritmo*. [PDF]. Disponible en:  
<http://fcasua.contad.unam.mx/apuntes/interiores/docs/20181/informatica/1/LI-1164-6097-A-Analisis-Diseno-Implantacion-Algoritmos-Plan2016.pdf>
- Importancia.org. (S.F). *Importancia de los algoritmos*. [En línea]. Disponible en:  
<https://www.importancia.org/algoritmos.php>
- Nurestone.academy. (S.F). *¿Qué es el análisis de algoritmos?*. [En línea]. Disponible en:  
<https://runestone.academy/runestone/static/pythoned/AlgorithmAnalysis/QueEsAnalysis>
- Fillottrani P. (2017). *Algoritmos y Complejidad*. Octubre 17, 2020, de Depto. Ciencias e Ingeniería de la Computación Universidad Nacional del Sur Sitio web:  
<http://www.cs.uns.edu.ar/prf/teaching/AyC17/downloads/Teoria/TecnicasHerramientas-1x1.pdf>