

## PRÁCTICA 1: DETERMINACIÓN EXPERIMENTAL DE LA COMPLEJIDAD TEMPORAL DE UN ALGORITMO

Mendoza Martínez Eduardo, Aguilar Gonzalez Daniel.

Escuela Superior de Cómputo  
Instituto Politécnico Nacional, México  
*edoomm8@gmail.com, daquilar glz97@gmail.com*

**Resumen:** La presente practica pretende mostrar el funcionamiento de dos algoritmos (suma binaria y mínimo común divisor mediante el algoritmo de Euclides) a través del tiempo de ejecución.

**Palabras Clave:** Algoritmo, Complejidad algorítmica, suma binaria, algoritmo de Euclides para mcd, tiempo de un algoritmo.

### 1 Introducción

Los algoritmos están presentes en nuestra vida cotidiana y los usamos de manera inconsciente, ejemplo de esto es que los usamos para vestirnos, atarnos la agujetas, llegar a la escuela o trabajo, solucionar un problema de cualquier índole, manual de usuario, etc. Así como existen diferentes problemas de igual manera existen diferentes algoritmos de los cuales cada uno tiene diferente complejidad y tiempo de ejecución para llevarse a cabo. En esta practica se mostrará lo explicado ya que se implementan 2 algoritmos basicos una suma binaria de numeros de misma longitud y el llamado Algoritmo de Euclides con el cual se obtiene el maximo comun divisor de dos numeros. En estos se prentende mostrar los diferentes tiempos de ejecución en que tardan cada uno y la utilización de recursos que requieren.

### 2 Conceptos Básicos

Para esta práctica, se requiere conocer algunos conceptos básicos relacionados al análisis de los algoritmos, así como el funcionamiento de estos.

## 2.1 Notación Theta

La notación  $\Theta$  nos indica simultáneamente los límites superior e inferior de ejecución, cuando estos están determinados (en una misma función) por diferentes múltiplos reales positivos.

Es decir,  $t(n) \in \Theta(f(n))$  si y solo si  $t(n)$  pertenece simultáneamente a  $O(f(n))$  y a  $\Omega(f(n))$ .

Expresado más formalmente decimos que  $\Theta = (f(n)) = \Omega(f(n)) \cap O(f(n))$ . A esto también se le conoce como orden exacto de  $f(n)$  ya que su tiempo de ejecución está acotado por arriba y por abajo al mismo tiempo.

## 2.2 Notación O

Se dice que dada una función  $g(n)$ , se define  $O(g(n))$  como :

$$O(g(n)) = \{f(n) | \exists n C_1 > 0 \text{ y } n_0 > 0 \text{ II } 0 \leq f(n) \leq C_1 g(n)\}$$

## 2.3 Notación Omega Mayúscula

Contrario a la notación  $O$ , la notación  $\Omega$  se usa para determinar los límites inferiores en el consumo de recursos  $f(n) = \Omega(g(n))$  o lo que es lo mismo si existe una  $f(n) \geq \epsilon \Omega(g(n))$  constante real  $C > 0$  y una constante entera  $n_0 \geq 1$  tal que  $f(n) \geq Cg(n)$  para cada entero  $n \geq n_0$ . Aquí va todo lo necesario para comprender el trabajo. En el caso de esta práctica podrían colocar los conceptos de  $\Theta$ ,  $O$  y  $\Omega$ . Además comentar que algoritmos se desarrollarán y mostrar los algoritmos, pueden mostrar ejemplos del funcionamiento del algoritmo implementado.

## 2.4 Algoritmo

Serie de pasos bien definidos y finitos para resolver un problema.

## 2.5 Suma binaria

La suma o adición binaria es análoga a la de los números decimales. La diferencia radica en que en los números binarios se produce un acarreo (carry) cuando la suma excede de uno mientras en decimal se produce un acarreo cuando la suma excede de nueve(9). La suma binaria puede presentar las siguientes combinaciones:

### 2.5.1 Combinaciones

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 10$

Note que al sumar  $1 + 1$  es 10, es decir, llevamos 1 a la siguiente posición de la izquierda (acarreo). Esto es equivalente, en el sistema decimal a sumar  $9 + 1$ , que da 10: cero en la posición que estamos sumando y un 1 de acarreo a la siguiente posición.

Los números o sumandos se suman en paralelo o en columnas, colocando un número encima del otro. Todos los números bajo la misma columna tienen el mismo valor posicional.

El orden de ubicación de los números no importa (propiedad conmutativa).

## 2.6 Algoritmo de Euclides para mcd

El algoritmo de Euclides es una técnica para encontrar rápidamente el MCD de dos enteros. El algoritmo es el siguiente:

### 2.6.1 Combinaciones

- Si  $A = 0$  entonces  $\text{MCD}(A,B)=B$ , ya que el  $\text{MCD}(0,B)=B$ , y podemos detenernos.
- Si  $B = 0$  entonces  $\text{MCD}(A,B)=A$ , ya que el  $\text{MCD}(A,0)=A$ , y podemos detenernos.
- Escribe  $A$  en la forma cociente y residuo ( $A = B - Q + R$ ).
- Encuentra  $\text{MCD}(B,R)$  al usar el algoritmo de Euclides, ya que  $\text{MCD}(A,B) = \text{MCD}(B,R)$ .

## 2.7 Complejidad Algorítmica

La complejidad algorítmica representa la cantidad de recursos (temporales) que necesita un algoritmo para resolver un problema y por tanto permite determinar la eficiencia de dicho algoritmo.

## 3 Experimentación y Resultados

### 3.1 Suma binaria

Para esta parte de la práctica fue necesario implementar un algoritmo capaz de realizar una suma binaria entre dos términos A y B. El algoritmo se muestra a continuación en pseudocódigo.

```

suma (A[1, ..., n], B[1, ..., n])
  i ← n - 1
  acarreo ← 0
  C[n]

  while i != -1 do
    if acarreo == 0 then
      if A[i] == 0 and B[i] == 0 then
        acarreo ← 0
        c ← 0
      else if A[i] == 1 and B[i] == 1 then
        acarreo ← 1
        c ← 0
      else
        acarreo ← 0
        c ← 1
    else
      if A[i] == 0 and B[i] == 0 then
        acarreo ← 0
        c ← 1
      else if A[i] == 1 and B[i] == 1 then
        acarreo ← 1
        c ← 1
      else
        acarreo ← 1
        c ← 1

    C[0] ← c
    i ← i - 1

  if acarreo == 1 then
    C[0] ← acarreo

```

Las pruebas de escritorio se implementaron en Python y a continuación se muestran algunas pruebas con números binarios aleatorios de número de bits no requeridos, esto con el fin de verificar si las sumas las hace correctamente.

```

C:\Windows\System32\cmd.exe

C:\Users\52556\Documents\ESCOM\Análisis de algoritmos\P1>py suma.py
[1]
+
[0]
[1]

[1, 0]
+
[1, 1]
[1, 0, 1]

[1, 0, 1]
+
[1, 1, 1]
[1, 1, 0, 0]

[0, 1, 0, 0]
+
[0, 1, 0, 0]
[1, 0, 0, 0]

[0, 1, 0, 1, 1]
+
[1, 1, 1, 1, 0]
[1, 0, 1, 0, 0, 1]

[1, 0, 1, 1, 1, 0]
+
[1, 1, 1, 0, 1, 0]
[1, 1, 0, 1, 0, 0, 0]

[1, 0, 1, 1, 0, 1, 0]
+
[1, 0, 1, 1, 0, 0, 0]
[1, 0, 1, 1, 0, 0, 1, 0]

```

Una vez verificado los resultados, procedimos a cambiar el número de bits a los acordados, es decir, a potencias de 2, y en vez de mostrar la suma, mostramos el tiempo de ejecución, como se muestra a continuación.

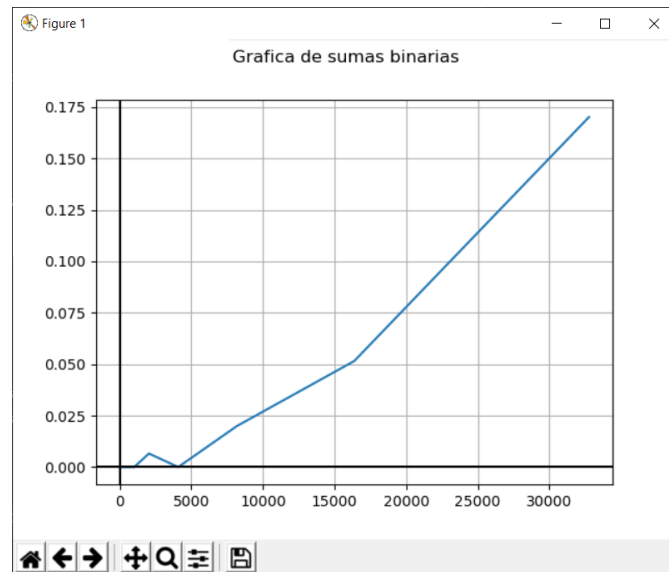
```

Select C:\Windows\System32\cmd.exe - py suma.py

C:\Users\52556\Documents\ESCOM\Análisis de algoritmos\P1>py suma.py
1 -- Suma con dos terminos de 2 bits, tiempo: 0.0 segundos --
2 -- Suma con dos terminos de 4 bits, tiempo: 0.0 segundos --
3 -- Suma con dos terminos de 8 bits, tiempo: 0.0 segundos --
4 -- Suma con dos terminos de 16 bits, tiempo: 0.0 segundos --
5 -- Suma con dos terminos de 32 bits, tiempo: 0.0 segundos --
6 -- Suma con dos terminos de 64 bits, tiempo: 0.0 segundos --
7 -- Suma con dos terminos de 128 bits, tiempo: 0.0 segundos --
8 -- Suma con dos terminos de 256 bits, tiempo: 0.0 segundos --
9 -- Suma con dos terminos de 512 bits, tiempo: 0.0 segundos --
10 -- Suma con dos terminos de 1024 bits, tiempo: 0.0 segundos --
11 -- Suma con dos terminos de 2048 bits, tiempo: 0.006578683853149414 segundos --
12 -- Suma con dos terminos de 4096 bits, tiempo: 0.0 segundos --
13 -- Suma con dos terminos de 8192 bits, tiempo: 0.019946575164794922 segundos --
14 -- Suma con dos terminos de 16384 bits, tiempo: 0.051573753356933594 segundos --
15 -- Suma con dos terminos de 32768 bits, tiempo: 0.17020225524902344 segundos --

```

La gráfica de estos primeros tiempos quedo conformada de la siguiente forma

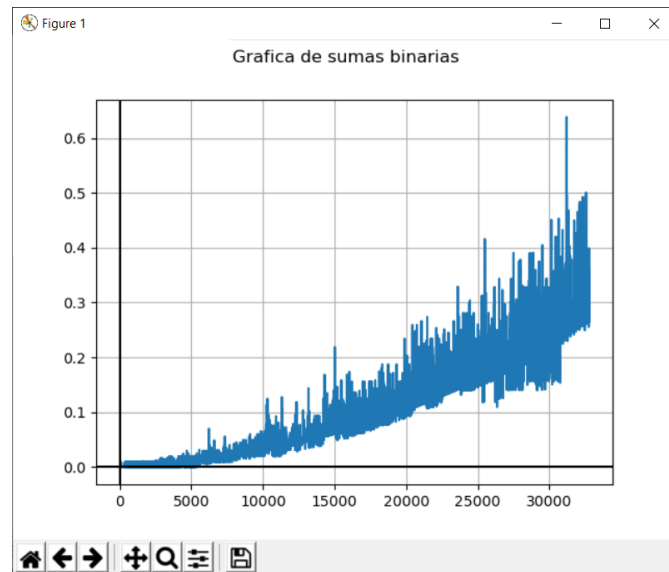


Este mismo experimento se realizo incrementando el número de bits uno a uno, tal que fue posible observar la forma de la curva que va tomando el algoritmo implementado.

```

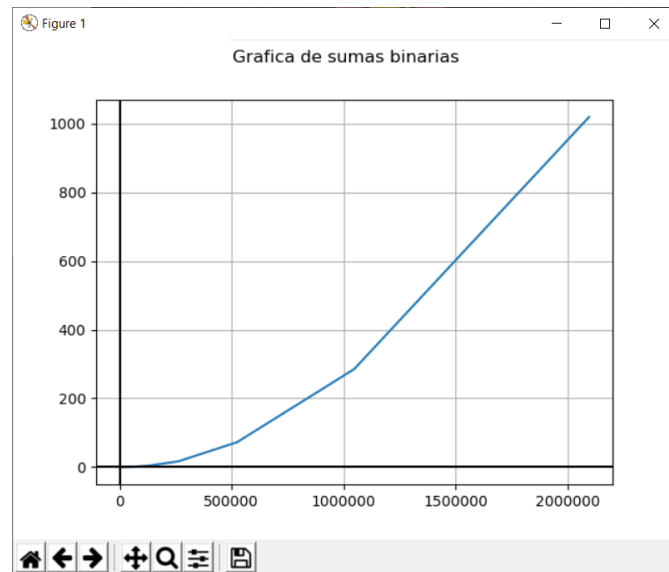
C:\Windows\System32\cmd.exe - py sumapy
32747 -- Suma con dos terminos de 32747 bits, tiempo: 0.2553746700286805 segundos --
32748 -- Suma con dos terminos de 32748 bits, tiempo: 0.34366822242736816 segundos --
32749 -- Suma con dos terminos de 32749 bits, tiempo: 0.28147053718566895 segundos --
32750 -- Suma con dos terminos de 32750 bits, tiempo: 0.2652900218963623 segundos --
32751 -- Suma con dos terminos de 32751 bits, tiempo: 0.26625061031515625 segundos --
32752 -- Suma con dos terminos de 32752 bits, tiempo: 0.363631010055542 segundos --
32753 -- Suma con dos terminos de 32753 bits, tiempo: 0.2655925750732422 segundos --
32754 -- Suma con dos terminos de 32754 bits, tiempo: 0.26529455184936523 segundos --
32755 -- Suma con dos terminos de 32755 bits, tiempo: 0.2652904987335205 segundos --
32756 -- Suma con dos terminos de 32756 bits, tiempo: 0.2652904987335205 segundos --
32757 -- Suma con dos terminos de 32757 bits, tiempo: 0.2652924060821533 segundos --
32758 -- Suma con dos terminos de 32758 bits, tiempo: 0.26429009437561035 segundos --
32759 -- Suma con dos terminos de 32759 bits, tiempo: 0.26628684997558594 segundos --
32760 -- Suma con dos terminos de 32760 bits, tiempo: 0.2652897034777832 segundos --
32761 -- Suma con dos terminos de 32761 bits, tiempo: 0.26628708839416504 segundos --
32762 -- Suma con dos terminos de 32762 bits, tiempo: 0.3989331722295215 segundos --
32763 -- Suma con dos terminos de 32763 bits, tiempo: 0.26529383659362793 segundos --
32764 -- Suma con dos terminos de 32764 bits, tiempo: 0.2652900218963623 segundos --
32765 -- Suma con dos terminos de 32765 bits, tiempo: 0.26528120040893555 segundos --
32766 -- Suma con dos terminos de 32766 bits, tiempo: 0.2652888298034668 segundos --
32767 -- Suma con dos terminos de 32767 bits, tiempo: 0.26628732681274414 segundos --

```



Y se repitió el experimento con potencias de 2, primero se muestra con  $2^{21}$  bits con un tiempo tardado de 17 minutos aproximadamente

```
C:\Windows\System32\cmd.exe - py sumapy
C:\Users\S2556\Documents\ESCOM\Análisis de algoritmos\PI>py suma.py
1 -- Suma con dos terminos de 2 bits, tiempo: 0.0 segundos --
2 -- Suma con dos terminos de 4 bits, tiempo: 0.0 segundos --
3 -- Suma con dos terminos de 8 bits, tiempo: 0.0 segundos --
4 -- Suma con dos terminos de 16 bits, tiempo: 0.0 segundos --
5 -- Suma con dos terminos de 32 bits, tiempo: 0.0 segundos --
6 -- Suma con dos terminos de 64 bits, tiempo: 0.0 segundos --
7 -- Suma con dos terminos de 128 bits, tiempo: 0.0 segundos --
8 -- Suma con dos terminos de 256 bits, tiempo: 0.0 segundos --
9 -- Suma con dos terminos de 512 bits, tiempo: 0.0 segundos --
10 -- Suma con dos terminos de 1024 bits, tiempo: 0.0 segundos --
11 -- Suma con dos terminos de 2048 bits, tiempo: 0.0009968280792236328 segundos --
12 -- Suma con dos terminos de 4096 bits, tiempo: 0.003989458884106445 segundos --
13 -- Suma con dos terminos de 8192 bits, tiempo: 0.014960289801464844 segundos --
14 -- Suma con dos terminos de 16384 bits, tiempo: 0.04886817932128906 segundos --
15 -- Suma con dos terminos de 32768 bits, tiempo: 0.1845107078552246 segundos --
16 -- Suma con dos terminos de 65536 bits, tiempo: 0.8127932548522949 segundos --
17 -- Suma con dos terminos de 131072 bits, tiempo: 4.1483893394470215 segundos --
18 -- Suma con dos terminos de 262144 bits, tiempo: 16.53691029548645 segundos --
19 -- Suma con dos terminos de 524288 bits, tiempo: 72.21686922149658 segundos --
20 -- Suma con dos terminos de 1048576 bits, tiempo: 285.54307746887207 segundos --
21 -- Suma con dos terminos de 2097152 bits, tiempo: 1019.3919959068298 segundos --
```



Y por último se muestra con  $2^{23}$  bits, donde los últimos puntos de la gráfica se vieron un poco afectados por la computadora que estuvo realizando otras tareas, ya que para este proceso se tardó en ejecutar un aproximado de 5 horas y media

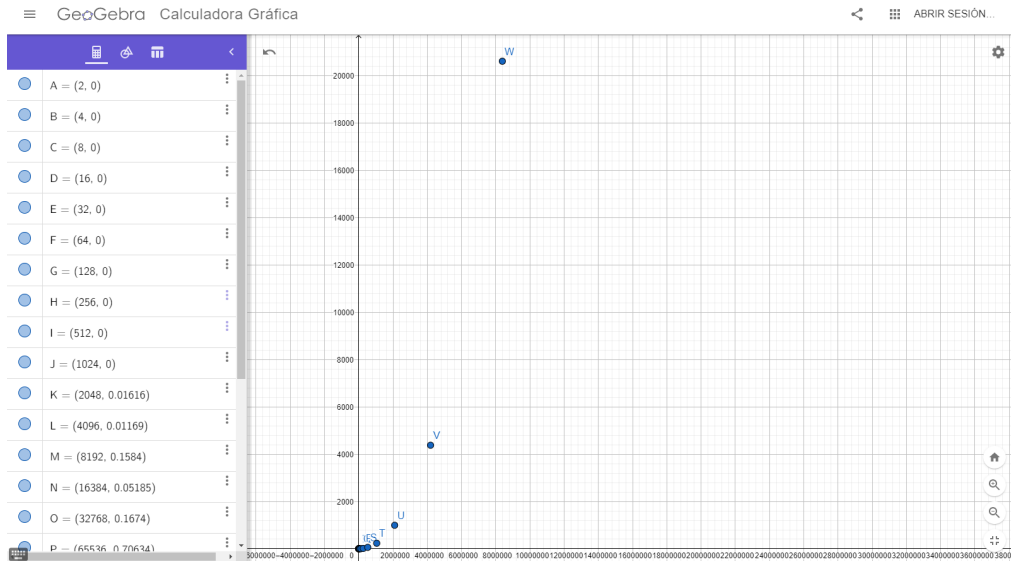
```

C:\Windows\System32\cmd.exe

2 -- Suma con dos terminos de 4 bits, tiempo: 0.0 segundos --
3 -- Suma con dos terminos de 8 bits, tiempo: 0.0 segundos --
4 -- Suma con dos terminos de 16 bits, tiempo: 0.0 segundos --
5 -- Suma con dos terminos de 32 bits, tiempo: 0.0 segundos --
6 -- Suma con dos terminos de 64 bits, tiempo: 0.0 segundos --
7 -- Suma con dos terminos de 128 bits, tiempo: 0.0 segundos --
8 -- Suma con dos terminos de 256 bits, tiempo: 0.0 segundos --
9 -- Suma con dos terminos de 512 bits, tiempo: 0.0 segundos --
10 -- Suma con dos terminos de 1024 bits, tiempo: 0.0 segundos --
11 -- Suma con dos terminos de 2048 bits, tiempo: 0.01616048812866211 segundos --
12 -- Suma con dos terminos de 4096 bits, tiempo: 0.011699914932250977 segundos --
13 -- Suma con dos terminos de 8192 bits, tiempo: 0.01584005355834961 segundos --
14 -- Suma con dos terminos de 16384 bits, tiempo: 0.051850318908691406 segundos --
15 -- Suma con dos terminos de 32768 bits, tiempo: 0.16740965843200684 segundos --
16 -- Suma con dos terminos de 65536 bits, tiempo: 0.706343412399292 segundos --
17 -- Suma con dos terminos de 131072 bits, tiempo: 3.812422037124634 segundos --
18 -- Suma con dos terminos de 262144 bits, tiempo: 13.440423727035522 segundos --
19 -- Suma con dos terminos de 524288 bits, tiempo: 57.93163728713989 segundos --
20 -- Suma con dos terminos de 1048576 bits, tiempo: 231.87008452415466 segundos --
21 -- Suma con dos terminos de 2097152 bits, tiempo: 991.9913923740387 segundos --
22 -- Suma con dos terminos de 4194304 bits, tiempo: 4375.415351629257 segundos --
23 -- Suma con dos terminos de 8388608 bits, tiempo: 20607.315758228382 segundos --

```



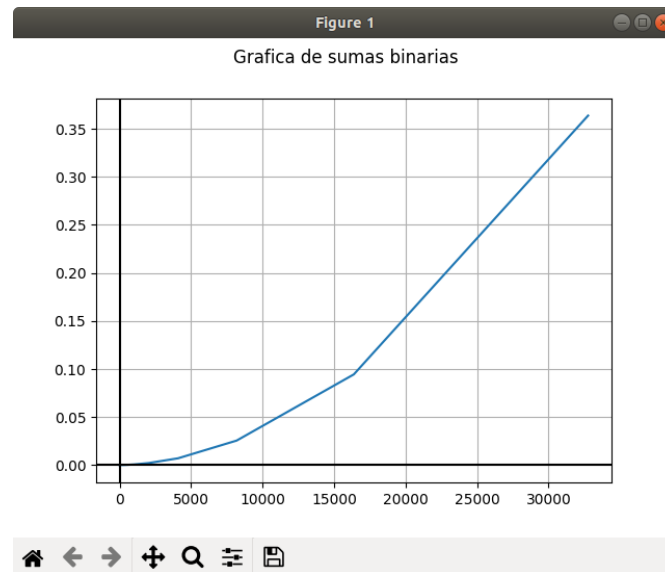


Para proponer una función  $O(g(n))$  observamos que si se realiza esta prueba en otra computadora, los tiempos pueden variar muy poco

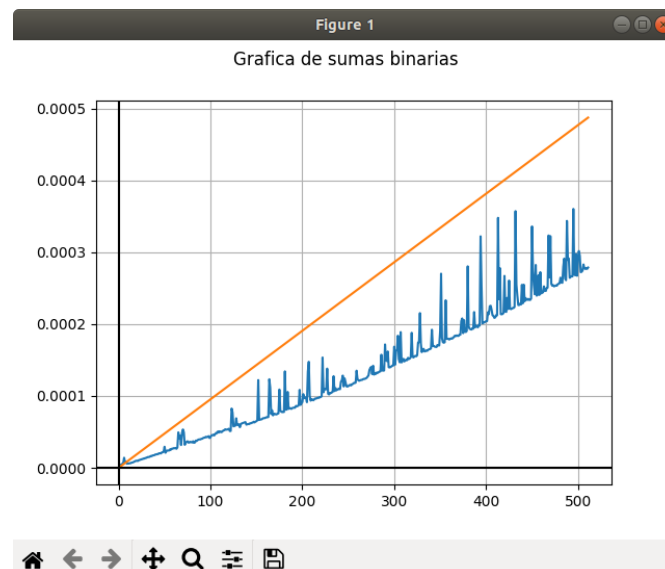
```

File Edit View Search Terminal Help
edoomm@Satellite-L735: ~/Documents/ESCOM
Pygments, readme-renderer, pyparser, cffi, cryptography, jeepney, SecretStorage, keyring, twine, SimpleWebSocketServer, pyloco, matplotlib
Successfully installed Pygments-2.5.2 SecretStorage-3.1.2 SimpleWebSocketServer-0.1.1 bleach-3.1.0 certifi-2019.11.28 cffi-1.13.2 chardet-3.0.4
cryptography-2.8 cytoolz-0.10.0 docutils-0.16 idna-2.8 importlib-metadata-1.5.0 jeepney-0.4.2 keyring-21.1.0 kiwisolver-1.1.0 matplotlib-3.1.3 m
numpy-1.18.1 pkginfo-1.5.0.1 pyparser-2.19 pyloco-0.0.139 pyparsing-2.4.6 python-dateutil-2.8.1 readme-renderer-24.0 requests-2.22.0
requests-toolbelt-0.9.1 setuptools-45.1.0 six-1.14.0 tqdm-4.42.0 twine-3.1.1 typing-3.7.4.1 urllib3-1.25.8 ushlex-0.99.1 webencodings-0.5.1
websocket-client-0.57.0 zipp-2.1.0
edoomm@Satellite-L735:~/Documents/ESCOM$ python3 suma.py
1 -- Suma con dos terminos de 2 bits, tiempo: 6.4373016357421875e-06 segundos --
2 -- Suma con dos terminos de 4 bits, tiempo: 7.152557373046875e-06 segundos --
3 -- Suma con dos terminos de 8 bits, tiempo: 8.821487426757812e-06 segundos --
4 -- Suma con dos terminos de 16 bits, tiempo: 1.5020370483398438e-05 segundos --
5 -- Suma con dos terminos de 32 bits, tiempo: 2.5272369384765625e-05 segundos --
6 -- Suma con dos terminos de 64 bits, tiempo: 4.887580871582031e-05 segundos --
7 -- Suma con dos terminos de 128 bits, tiempo: 0.00010132789611816406 segundos --
8 -- Suma con dos terminos de 256 bits, tiempo: 0.00011801719665527344 segundos --
9 -- Suma con dos terminos de 512 bits, tiempo: 0.0002810955047607422 segundos --
10 -- Suma con dos terminos de 1024 bits, tiempo: 0.0007350444793701172 segundos --
11 -- Suma con dos terminos de 2048 bits, tiempo: 0.0022237300872802734 segundos --
12 -- Suma con dos terminos de 4096 bits, tiempo: 0.007136344909667969 segundos --
13 -- Suma con dos terminos de 8192 bits, tiempo: 0.025573253631591797 segundos --
14 -- Suma con dos terminos de 16384 bits, tiempo: 0.09441065788269043 segundos --
15 -- Suma con dos terminos de 32768 bits, tiempo: 0.36365771293640137 segundos --

```



Por lo tanto, si este algoritmo es analizado a priori, podemos observar que tendrá  $O(n)$ , esto de igual forma se verifica con el análisis a posteriori realizado en la práctica y lo podemos ver claramente en la siguiente gráfica



Así también, se puede acotar por otra función  $c_1g(n)$ , pero en la notación  $O$  los coeficientes pueden ser omitidos. Así bien,

$$Suma \in O(n)$$

### 3.2 Mínimo común divisor mediante algoritmo de Euclides

Al ingresar un par de números enteros positivos ('m' y 'n'), se evalúan mediante el algoritmo de Euclides (descrito anteriormente), y se muestra al usuario en pantalla el mínimo común divisor. A continuación se presenta el algoritmo de Euclides implementado en Python

```
Euclides (M[1, ..., n], N[1, ..., n])

while n!=0 do

    r <- m mod n

    m <- n

    n <- r

return m

mcd = Euclides(m, n)
```

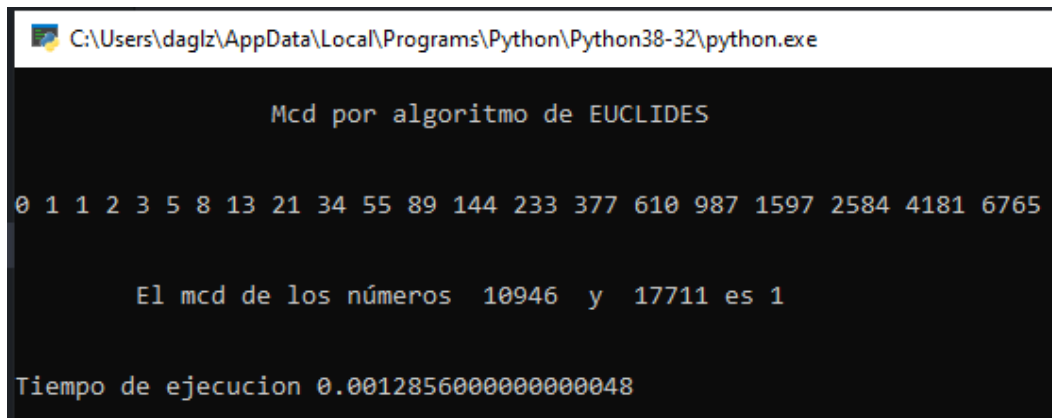
Para obtener el mcd tambien se implemento el algoritmo para desarrollar la Serie de Fibonacci, el cual es el siguiente:

```
Fibonacci (M[1, ..., n], N[1, ..., n])
m, n <- 0,1

while m < o do

    m, n <- n, m+n
```

Se realizaron pruebas de escritorio implementadas en python las cuales se muestran a continuación con numeros aleatorios para determinar el tamaño de la serie de Fibonacci. En la siguiente pantalla se muestra una prueba con una cantidad menor a 10,000, mostrando la serie hasta el numero más cercano al seleccionado y se saca el mcd de los 2 numeros siguientes al ultimo junto con el tiempo de ejecución.



```
C:\Users\daglz\AppData\Local\Programs\Python\Python38-32\python.exe

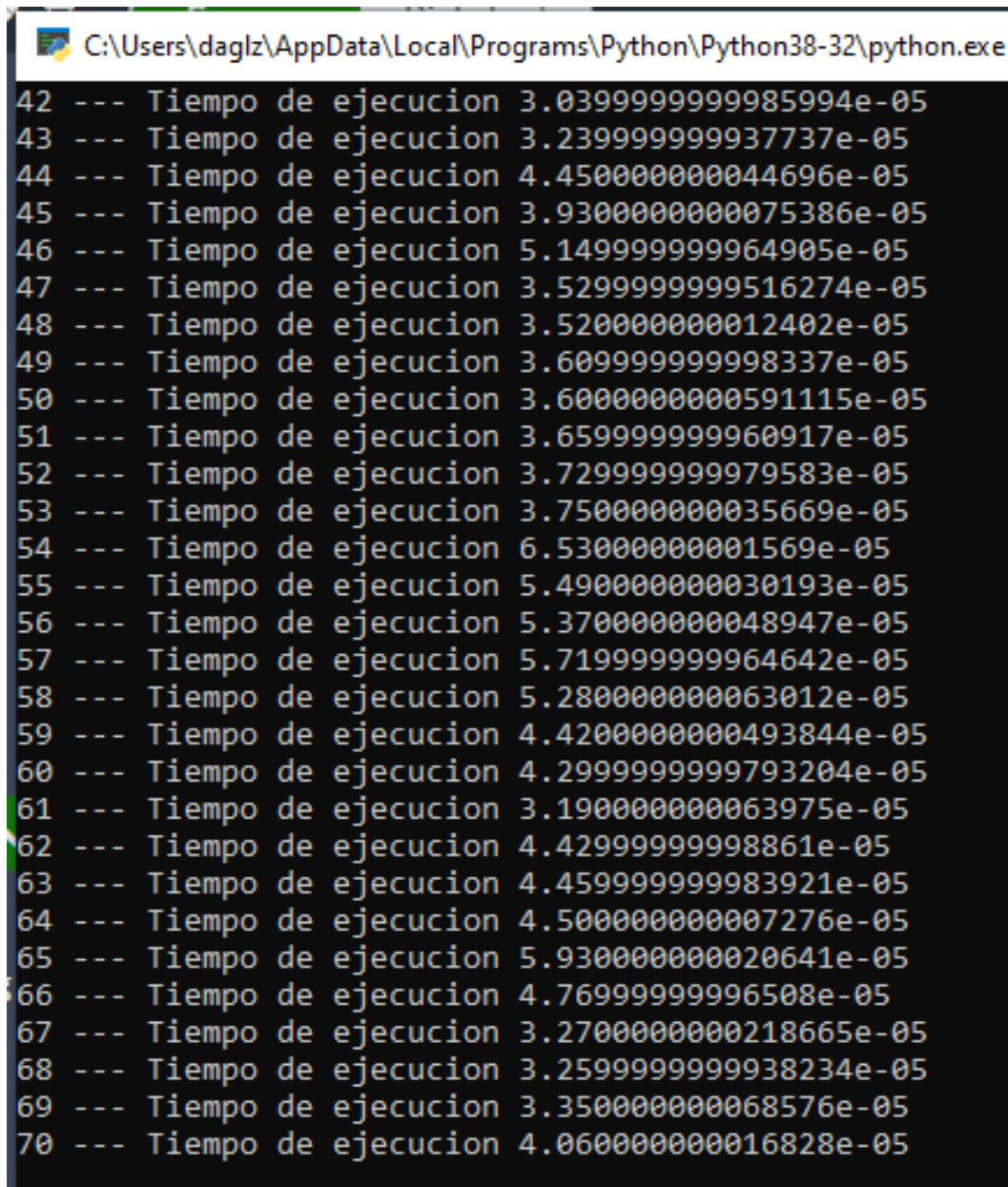
Mcd por algoritmo de EUCLIDES

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765

El mcd de los números 10946 y 17711 es 1

Tiempo de ejecucion 0.0012856000000000048
```

Al observar los primeros resultados y cambiar el limite de la serie para observar su tiempo de ejecucion, continuamos a trabajar con el  $i$ -esimo número de la serie de fibonacci, el cual nos representa el peor caso en el algoritmo de Euclides. A continuacion se muestra el tiempo de ejecución de ejecucion que tarda en realizarse cada operacion, el ejemplo se muestra con 70.



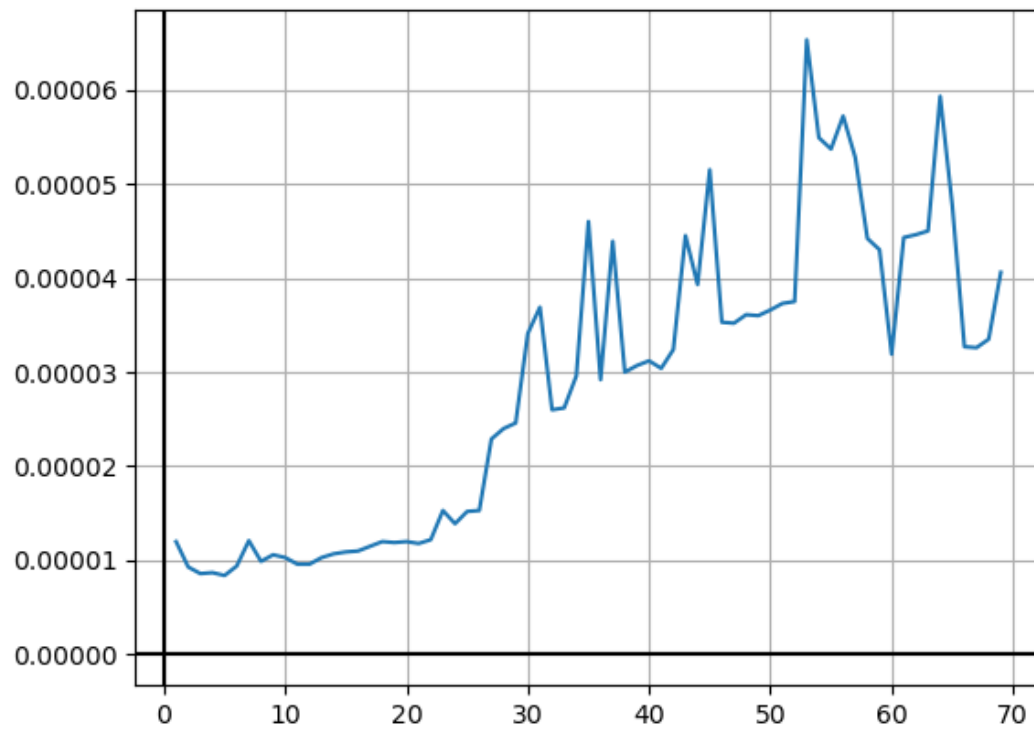
```

C:\Users\daglz\AppData\Local\Programs\Python\Python38-32\python.exe
42 --- Tiempo de ejecucion 3.0399999999985994e-05
43 --- Tiempo de ejecucion 3.2399999999937737e-05
44 --- Tiempo de ejecucion 4.4500000000044696e-05
45 --- Tiempo de ejecucion 3.93000000000075386e-05
46 --- Tiempo de ejecucion 5.149999999964905e-05
47 --- Tiempo de ejecucion 3.5299999999516274e-05
48 --- Tiempo de ejecucion 3.520000000012402e-05
49 --- Tiempo de ejecucion 3.609999999998337e-05
50 --- Tiempo de ejecucion 3.60000000000591115e-05
51 --- Tiempo de ejecucion 3.659999999960917e-05
52 --- Tiempo de ejecucion 3.729999999979583e-05
53 --- Tiempo de ejecucion 3.7500000000035669e-05
54 --- Tiempo de ejecucion 6.530000000001569e-05
55 --- Tiempo de ejecucion 5.4900000000030193e-05
56 --- Tiempo de ejecucion 5.3700000000048947e-05
57 --- Tiempo de ejecucion 5.719999999964642e-05
58 --- Tiempo de ejecucion 5.2800000000063012e-05
59 --- Tiempo de ejecucion 4.42000000000493844e-05
60 --- Tiempo de ejecucion 4.2999999999793204e-05
61 --- Tiempo de ejecucion 3.19000000000063975e-05
62 --- Tiempo de ejecucion 4.42999999998861e-05
63 --- Tiempo de ejecucion 4.459999999983921e-05
64 --- Tiempo de ejecucion 4.5000000000007276e-05
65 --- Tiempo de ejecucion 5.9300000000020641e-05
66 --- Tiempo de ejecucion 4.76999999996508e-05
67 --- Tiempo de ejecucion 3.27000000000218665e-05
68 --- Tiempo de ejecucion 3.2599999999938234e-05
69 --- Tiempo de ejecucion 3.3500000000068576e-05
70 --- Tiempo de ejecucion 4.0600000000016828e-05

```

La grafica de estos tiempo quedo de la siguiente manera

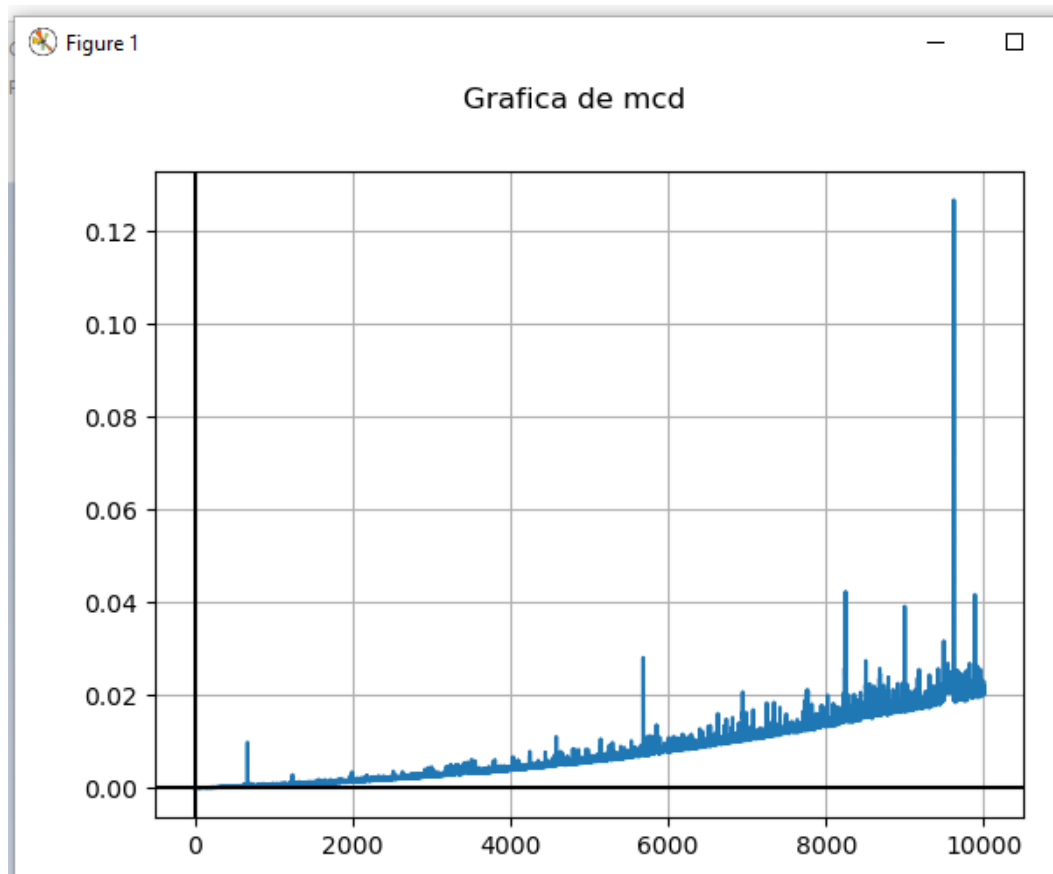
Grafica de mcd



A continuacion mostraremos tiempos en ejecución y su grafica con una cantidad mucho mayo la cual sera de 10000.

C:\Users\daglz\AppData\Local\Programs\Python\Python38-32\python.exe

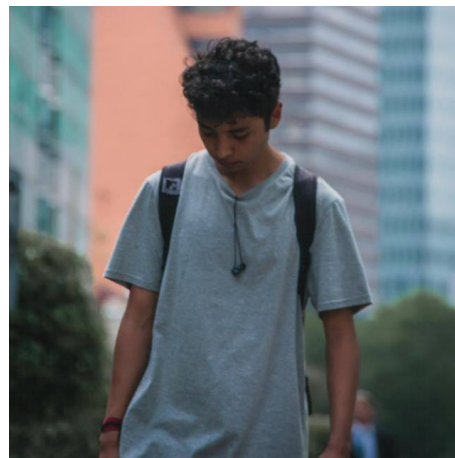
```
9972 --- Tiempo de ejecucion 0.020336799999995492
9973 --- Tiempo de ejecucion 0.0208397999999990305
9974 --- Tiempo de ejecucion 0.020722699999999321
9975 --- Tiempo de ejecucion 0.020027600000000592
9976 --- Tiempo de ejecucion 0.0208275999999989843
9977 --- Tiempo de ejecucion 0.020645500000000054
9978 --- Tiempo de ejecucion 0.0200644999999989494
9979 --- Tiempo de ejecucion 0.020543000000000353
9980 --- Tiempo de ejecucion 0.0217600000000000446
9981 --- Tiempo de ejecucion 0.0213031000000011455
9982 --- Tiempo de ejecucion 0.0200509000000001093
9983 --- Tiempo de ejecucion 0.020026500000000017
9984 --- Tiempo de ejecucion 0.021245899999999671
9985 --- Tiempo de ejecucion 0.020239300000000007
9986 --- Tiempo de ejecucion 0.0221303999999994665
9987 --- Tiempo de ejecucion 0.0216932999999995475
9988 --- Tiempo de ejecucion 0.020374599999999669
9989 --- Tiempo de ejecucion 0.0208355999999998067
9990 --- Tiempo de ejecucion 0.0209193999999996813
9991 --- Tiempo de ejecucion 0.0199580999999996648
9992 --- Tiempo de ejecucion 0.020332199999999858
9993 --- Tiempo de ejecucion 0.022289599999999347
9994 --- Tiempo de ejecucion 0.0206397999999997877
9995 --- Tiempo de ejecucion 0.0218364999999992043
9996 --- Tiempo de ejecucion 0.0202498000000001983
9997 --- Tiempo de ejecucion 0.0230796999999996706
9998 --- Tiempo de ejecucion 0.0209313000000000874
9999 --- Tiempo de ejecucion 0.020970200000000782
10000 --- Tiempo de ejecucion 0.0202432999999989972
```



## 4 Conclusiones

Eduardo Mendoza Martínez

Con esta práctica pudimos observar la complejidad que pueden tomar algunos algoritmos, lo cual es de importancia para nosotros como próximos ingenieros, al realizar sistemas que empleen algoritmos y también de los algoritmos ya existentes, hacer la elección del que sea de nuestra convivencia a corde lo que necesite nuestro sistema y funcione de manera eficiente. Algo que observé muy interesante fue el tiempo de ejecución que se tenía entre 2 computadoras diferentes, entre una laptop gaming del





año pasado con un procesador i5 de 9na generación y la otra laptop con un i5 de quinta generación, el tiempo que tardaba la laptop más nueva realizaba más rápido las tareas mientras la otra más lento, sin embargo la curva y puntos que generaban los algoritmos seguían teniendo la misma forma, lo cual es lo mismo que dice la teoría, no importa el lenguaje de programación o la computadora, todo se puede definir bajo el mismo modelo matemático, en este caso el  $O(g(n))$  de los algoritmos trabajados. También una cosa a destacar es el tiempo con el que trabaja Linux y Windows es muy diferente, mientras en Linux te maneja tiempos hasta de  $10^{-6}$  segundos y en windows a tiempos muy pequeños la consola arrojaba 0 segundos como tiempo de ejecución.

Daniel Aguilar Gonzalez

En mi opinion el analisis de algoritmos es muy importante. Conocer que tan tardado puede llegar a ser un algoritmo dependiendo de una entrada, y de la situacion en la que se encuentren, como vimos en clase por ejemplo, los algoritmos pueden tener el mejor caso o el peor caso y pueden ser funciones completamente distintas y por eso se tienen las diferentes notaciones (o incluso algunos no hay ni mejor ni peor caso).

En el caso de Euclides nos indica que aunque la entrada crece, el tiempo no crece demasiado. Por su parte la suma binaria crece conforme la entrada crece, a comparacion de Euclides, esta funcion requiere mas tiempo para ejecutarse.

En la practica logramos el objetivo de hacer la funcion correcta de los algoritmos que se debian implementar pero en lo personal me costo bastante incluso no poder entender correctamente la funcion  $g(n)$  que acota nuestras funciones de los algoritmos implementados.

Como se vio en la practica cada computadora cuenta con diferentes características y cada una de ellas varia su tiempo de ejecucion de algun programa algunas mas rapido o mas lento que en otras, esto debido a la complejidad que tiene cada algoritmo que se implementa, algunos



pueden requerir más recursos, etc. Es por esto que debemos calcularla para tener mas o menos una idea de a lo que nos enfrentamos.

## 5 Anexo

### 5.1 Select-Sort

**Select-Sort(A[0, . . . , 4 - 1])**      Se recibe un arreglo de 4 elementos

Iteración 1:

```
for j ← 0 to j ≤ n - 2 do      Para j=0 hasta j≤(4-2)
    k ← j      La variable k toma el valor de 0
    for i ← j + 1 to i ≤ n - 1 do      Para i= 0 + 1 hasta i≤ 4 - 1
        if A[i] > A[k] then      Si A[1] > A[0]
            k ← i      La variable k, se le asigna el valor de i=1
    Intercambia (A[j],A[k])      Se coloca el contenido de A[0] A[1]
```

Iteración 2:

```
for j ← 0 to j ≤ n - 2 do      Para j=1 hasta j≤(4-2)
    k ← j      La variable k toma el valor de 1
    for i ← j + 1 to i ≤ n - 1 do      Para i= 1 + 1 hasta i≤ 4 - 1
        if A[i] > A[k] then      Si A[2] > A[1]
            k ← i      La variable k, se le asigna el valor de i=2
    Intercambia (A[j],A[k])      Se coloca el contenido de A[1] A[2]
```

Iteración 3:

```
for j ← 0 to j ≤ n - 2 do      Para j=2 hasta j≤(4-2)
    k ← j      La variable k toma el valor de 2
    for i ← j + 1 to i ≤ n - 1 do      Para i= 2 + 1 hasta i≤ 4 - 1
        if A[i] > A[k] then      Si A[3] > A[2]
            k ← i      La variable k, se le asigna el valor de i=3
    Intercambia (A[j],A[k])      Se coloca el contenido de A[2] A[3]
```

#### 5.1.1 Pruebas de escritorio

Para este algoritmo el código final implementando en Python quedó conformado de la siguiente forma

```
def selectSort(A):
    j = 0
    while j <= len(A) - 2:
        k = j
        i = j + 1
        while i <= len(A) - 1:
```

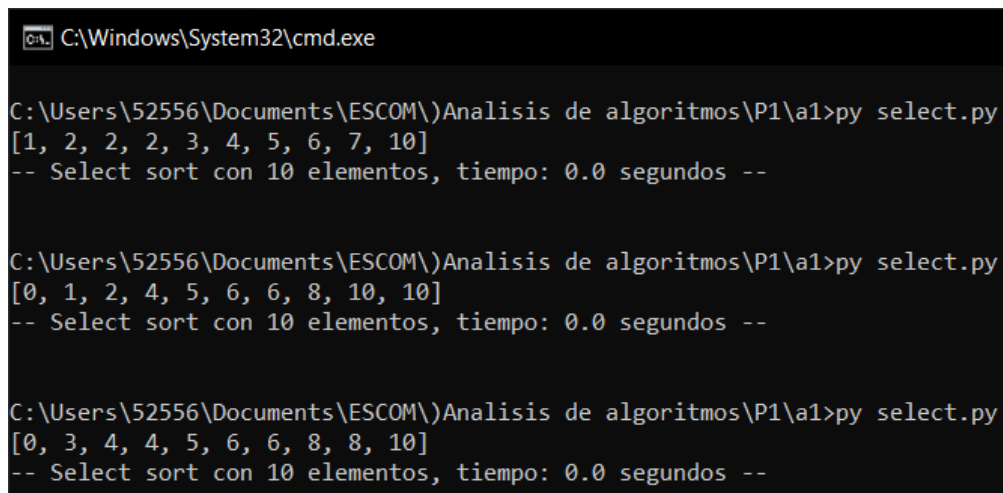
```

        if A[i] < A[k]:
            k = i
        i += 1
    A[j], A[k] = A[k], A[j]
    j += 1

print(A)

```

Y a continuación se muestran algunas pruebas con arreglos de tamaño 10, con elementos del 0 al 10



```

C:\Windows\System32\cmd.exe

C:\Users\52556\Documents\ESCOM\Análisis de algoritmos\P1\al>py select.py
[1, 2, 2, 2, 3, 4, 5, 6, 7, 10]
-- Select sort con 10 elementos, tiempo: 0.0 segundos --

C:\Users\52556\Documents\ESCOM\Análisis de algoritmos\P1\al>py select.py
[0, 1, 2, 4, 5, 6, 6, 8, 10, 10]
-- Select sort con 10 elementos, tiempo: 0.0 segundos --

C:\Users\52556\Documents\ESCOM\Análisis de algoritmos\P1\al>py select.py
[0, 3, 4, 4, 5, 6, 6, 8, 8, 10]
-- Select sort con 10 elementos, tiempo: 0.0 segundos --

```

### 5.1.2 Cálculo del orden de complejidad en el peor de los casos

Procedemos a reescribir las líneas de código y calcular sus costos en una tabla debajo de éstas.

**Select-Sort**( $A[0, \dots, n-1]$ )      Se recibe un arreglo de  $n$  elementos

```

for j ← 0 to j ≤ n - 2 do
    k ← j
    for i ← j + 1 to i ≤ n - 1 do
        if A[i] < A[k] then
            k ← i
    Intercambia (A[j], A[k])

```

Línea	Costo
$C_1$	n
$C_2$	n-1
$C_3$	$\sum_{i=1}^n t_i$
$C_4$	$\sum_{i=1}^n t_i - 1$
$C_5$	n-1

$$T(n) = C_1 n + (n-1)(C_2 + C_5) + C_3(\sum_{i=1}^n t_i) + C_4(\sum_{i=1}^n t_i - 1)$$

$$T(n) = C_1 n + (n-1)(C_2 + C_5) + C_3(\sum_{i=1}^n n_i) + C_4(\sum_{i=1}^n n_i - 1)$$

$$T(n) = (C_3 + C_4)n^2 + (C_5 - C_4 + C_2 + C_1)n - C_5 - C_2$$

$$\therefore T(n) \in O(n^2)$$

## 5.2 Máximo de un arreglo

### 5.2.1 Prueba de escritorio

Nuestro algoritmo quedó implementado como se muestra a continuación.

```
Maximo (A[0 , 1 , ... n-1])
    max = A[0]
    for i <- 0 to i < n do
        if A[i] > max then
            max <- A[i]
```

Y en seguida se muestran 4 pruebas que se hicieron con arreglos de tamaño 10, con elementos que van del 0 al 10

```
C:\Windows\System32\cmd.exe

C:\Users\52556\Documents\ESCOM\Análisis de algoritmos\P1\>py max.py
[5, 6, 5, 1, 4, 2, 2, 7, 6, 1]
Maximo elemento en el arreglo: 7

C:\Users\52556\Documents\ESCOM\Análisis de algoritmos\P1\>py max.py
[10, 0, 2, 2, 0, 4, 0, 6, 3, 9]
Maximo elemento en el arreglo: 10

C:\Users\52556\Documents\ESCOM\Análisis de algoritmos\P1\>py max.py
[10, 0, 9, 10, 7, 2, 1, 9, 10, 6]
Maximo elemento en el arreglo: 10

C:\Users\52556\Documents\ESCOM\Análisis de algoritmos\P1\>py max.py
[5, 4, 7, 6, 2, 3, 3, 3, 9, 5]
Maximo elemento en el arreglo: 9
```

### 5.2.2 Cálculo del orden de complejidad

Para calcular la complejidad del Algoritmo, reescribiremos el código y lo analizaremos línea por línea y registraremos en una tabla.

Maximo (A[0, 1, ... n-1]) Se recibe un arreglo de n elementos  
 max = A[0]  
 for i ← 0 to i < n do  
   if A[i] > max then  
     max ← A[i]

Línea	Costo
$C_1$	n
$C_2$	n-1
$C_3$	$\sum_{i=1}^n t_i$

$$\therefore T_{(n)} \in O(n^2)$$

### 5.3 Problema 8 del problemario

Para la primera parte se tuvo el siguiente algoritmo

Algoritmo1 (a[0, ..., n-1])  
 for i=n-1 to i>0 do  
   for j=0 to j<i  
     if a[j]>a[j+1] then  
       temp=a[j]  
       a[j]=a[j+1]  
       a[j+1]=temp

Asi pues, para calcular el orden de complejidad, analizamos linea por linea este algoritmo.

Línea	Costo
$C_1$	n
$C_2$	$\sum_{i=1}^{n-1} t_i$
$C_3$	$\sum_{i=1}^{n-1} (t_i - 1)$
$C_4$	$\sum_{i=1}^{n-1} (t_i - 1)$
$C_5$	$\sum_{i=1}^{n-1} (t_i - 1)$
$C_6$	$\sum_{i=1}^{n-1} (t_i - 1)$

Asi pues, notamos que en el mejor de los casos, el algoritmo no ejecuta el cuerpo del if, y en el peor caso, ejecuta el cuerpo del if. Podemos notar que las sentencias dentro del cuerpo del if son  $O(1)$ . Luego entonces, se tiene.

i	ti
1	n
2	n-1
...	...
n	n-i+1

Luego,

$$T(n) = C_1n + C_2\sum_{i=1}^{n-1}(n-i+1) + C_3\sum_{i=1}^{n-1}(n-i) + C_4\sum_{i=1}^{n-1}(n-i) + \\ C_5\sum_{i=1}^{n-1}(n-i) + C_6\sum_{i=1}^{n-1}(n-i)$$

Después,

$$T(n) = C_1n + C_2\sum_{i=1}^{n-1}n - C_2\sum_{i=1}^{n-1}i + C_2\sum_{i=1}^{n-1}1 + C_3\sum_{i=1}^{n-1}n - C_3\sum_{i=1}^{n-1}i + \\ C_4\sum_{i=1}^{n-1}n - C_4\sum_{i=1}^{n-1}i + C_5\sum_{i=1}^{n-1}n - C_5\sum_{i=1}^{n-1}i + C_6\sum_{i=1}^{n-1}n - C_6\sum_{i=1}^{n-1}i$$

Es decir,

$$T(n) = C_1n + (C_2 + C_3 + C_4 + C_5 + C_6)(n^2 - n) - \\ (C_2 + C_3 + C_4 + C_5 + C_6)\frac{n^2 - n}{2} + C_2(n - 1)$$

Por lo tanto,

$$T(n) = \frac{1}{2}(C_2 + C_3 + C_4 + C_5 + C_6)(n^2 - n) + (C_1 + C_2)n - C_2$$

Así podemos observar que  $T(n)$  es de la forma,

$$T(n) = a + bn + cn^2$$

$$\therefore T(n) \in O(n^2)$$

Para la segunda parte se tuvo el siguiente algoritmo

```

Algoritmo2 (A[0 , ... , n-1],x)
  inf=0
  sup=n-1
  while sup>=inf do
    i=(inf+sup)/2
    if A[i]==x then
      return i
    else if x<A[i] then
      sup=i-1
    else
      inf=i+1

```

Comenzaremos el analisis linea por linea para poder calcular el orden de su complejidad.

Como se puede observar, el algoritmo implementado se trata de una busqueda binaria, así pues, nuestra tabla queda de la siguiente forma.

Línea	Costo
$C_1$	1
$C_2$	1
$C_3$	$\frac{n-1}{2}$
$C_4$	$\frac{n-3}{2}$
$C_5$	$\frac{n-3}{2}$
$C_6$	$\frac{n-3}{2}$
$C_7$	$\frac{n-3}{2}$
$C_8$	$\frac{n-3}{2}$
$C_9$	$\frac{n-3}{2}$
$C_{10}$	$\frac{n-3}{2}$

Así, en el mejor de los casos, podemos observar que se da cuando el elemento a buscar esta a la mitad de nuestro arreglo, por lo tanto la sentencia del while pasaria a convertirse de  $\frac{n-1}{2}$  a 1. Y todas las demas sentencias despues del return se invalidan.

Es decir,

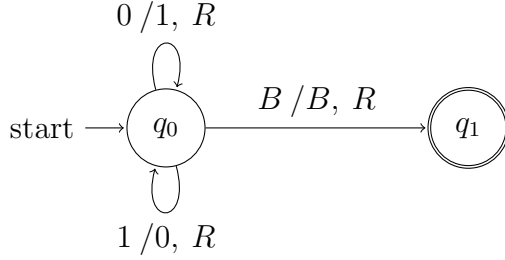
$$\begin{aligned}
 T(n) &= C_1(1) + C_2(1) + C_3(1) + C_4(1) + C_5(1) \\
 T(n) &= C_1 + C_2 + C_3 + C_4 + C_5 \\
 \therefore T(n) &\in \Omega(1)
 \end{aligned}$$

Luego, para el peor de los casos observamos que el while tendría que recorrer todas las mitades de nuestro arreglo, entonces se tendría que,

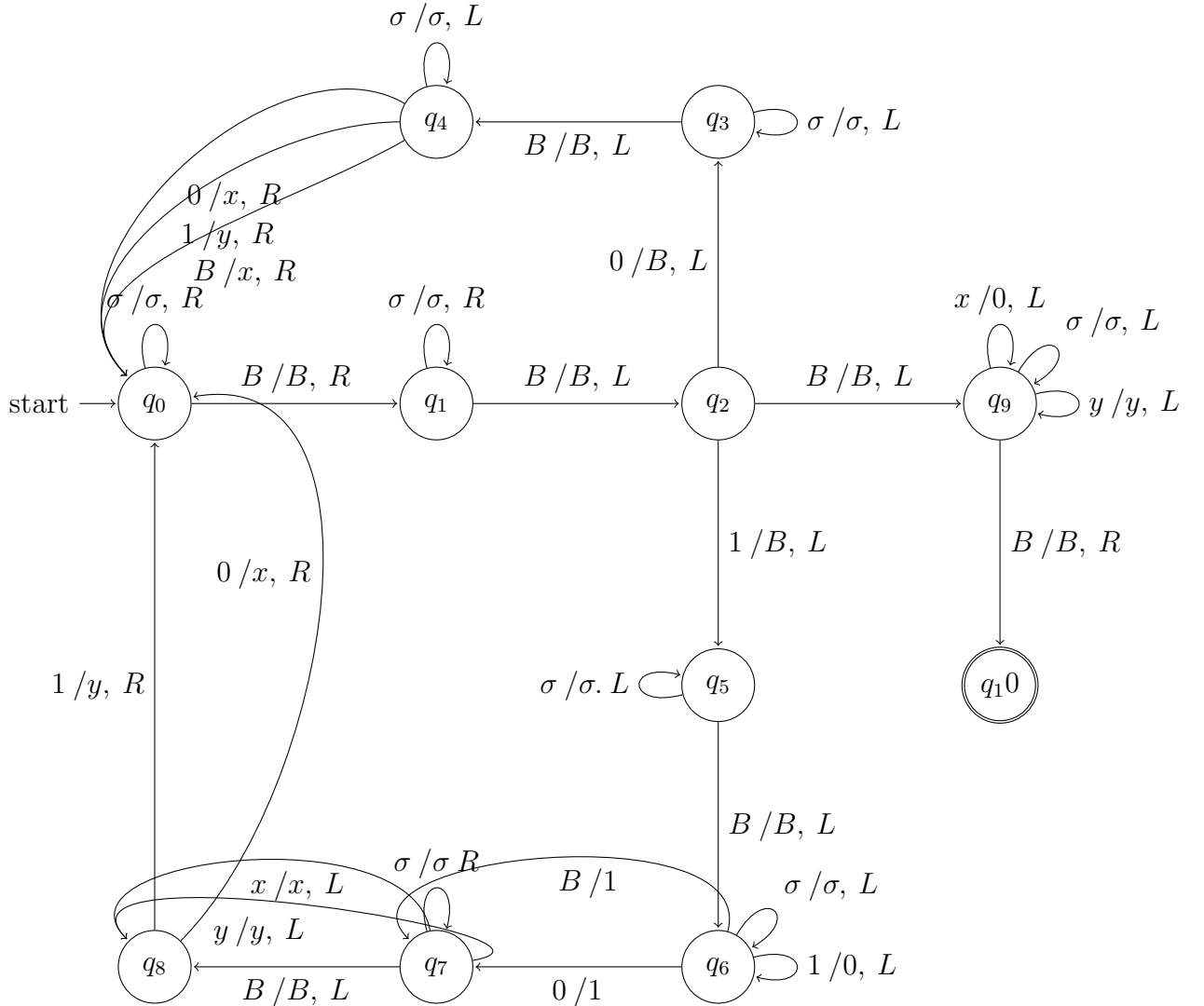
$$\begin{aligned}
 T(n) &= (C_1 + C_2)(1) + C_3\left(\frac{n-1}{2}\right) + (C_4 + C_5 + C_6 + C_7 + C_8 + C_9 + C_{10})\left(\frac{n-3}{2}\right) \\
 \therefore T(n) &\in O(n)
 \end{aligned}$$

## 5.4 Máquinas de Turing

Para la primer máquina de Turing se tuvo que implementar una máquina capaz de sacar el complemento de un número binario, es decir, convertir los 1 en 0.



La siguiente máquina de Turing pedida fue una capaz de realizar sumas binarias y quedo implementada como se muestra a continuación.





## 6 Bibliografía

- [1] Analisis de Algoritmos: Complejidad”, Lab.dit.upm.es, 1997. [Online].  
[http:// www.lab.dit.upm.es/ lprg/ material/apuntes/o/index.html](http://www.lab.dit.upm.es/lprg/material/apuntes/o/index.html).
- [2] Khan Academy. (2020). Notación Omega grande (Big-) (artículo) Khan Academy. [online] Available at:  
<https://es.khanacademy.org/computing/computer-science/algorithms/asymptotic-notation/a/big-big-omega-notation>
- [3] L. J. Aguilar, “Fundamentos de programación. algoritmos, estructura de datos y objetos, 4ta ed.” 2008.
- [4] Wikipedia, “Cota ajustada asintótica — wikipedia, la enciclopedia libre,” 2013, [Internet; descargado 30-agosto-2018]. [Online]. Available:  
<https://es.wikipedia.org/w/index.php?title=Cotaajustadaasint>
- [5] “Cota superior asintótica — wikipedia, la enciclopedia libre,” 2018, [Internet; descargado 30-agosto-2018]. [Online]. Available:  
<https://es.wikipedia.org/w/index.php?title=Cotasuperiorasint>
- [6] “Cota inferior asintótica — wikipedia, la enciclopedia libre,” 2016, [Internet; descargado 30-agosto-2018]. [Online]. Available:  
<https://es.wikipedia.org/w/index.php?title=Cotainferiorasint>