INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

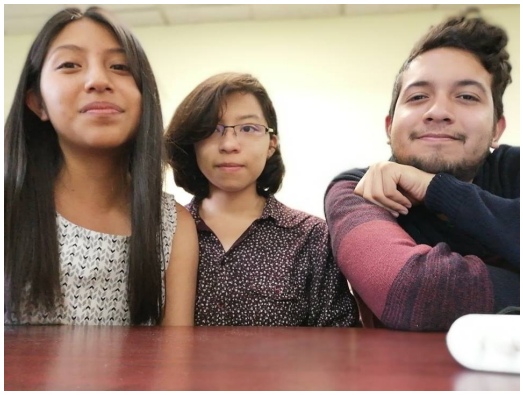
Práctica 2 - Análisis temporal y notación de orden (Algoritmos de búsqueda)

Unidad de aprendizaje: Análisis de Algoritmos

Grupo: 3CM3

Alumnos(a): "La naranja mecánica"
Nicolás Sayago Abigail
Parra Garcilazo Cinthya Dolores
Ramos Díaz Enrique

Profesor(a): Edgardo Adrián Franco Martínez



10 de Octubre 2018

Índice

1. Planteamiento del problema

Existen diversos métodos de búsqueda numérica, en este documento se analizaran 3, se observará y comparará el comportamiento de cada uno, para determinar el mejor de todos.

Se tomaran resultados experimentales en una plataforma determinada para determinar la complejidad temporal y su orden de complejidad (cota O mayúscula) de los siguientes algoritmos: Búsqueda Lineal, Búsqueda Binaria y Árbol de Búsqueda Binario.

Además, se analizará la efectividad y velocidad de la implementación de éstos algoritmos con ejecución de hilos simultáneos, para determinar si vale o no la pena al momento de ahorrar recursos de nuestra computadora para tamaños de problema muy grandes.

2. Plataforma Experimental

Especificaciones de Hardware:

- CPU: Intel Core-i5 6500 3.2 GHz
- Memoria: RAM DDR4 5.9 GB 2133 MHz

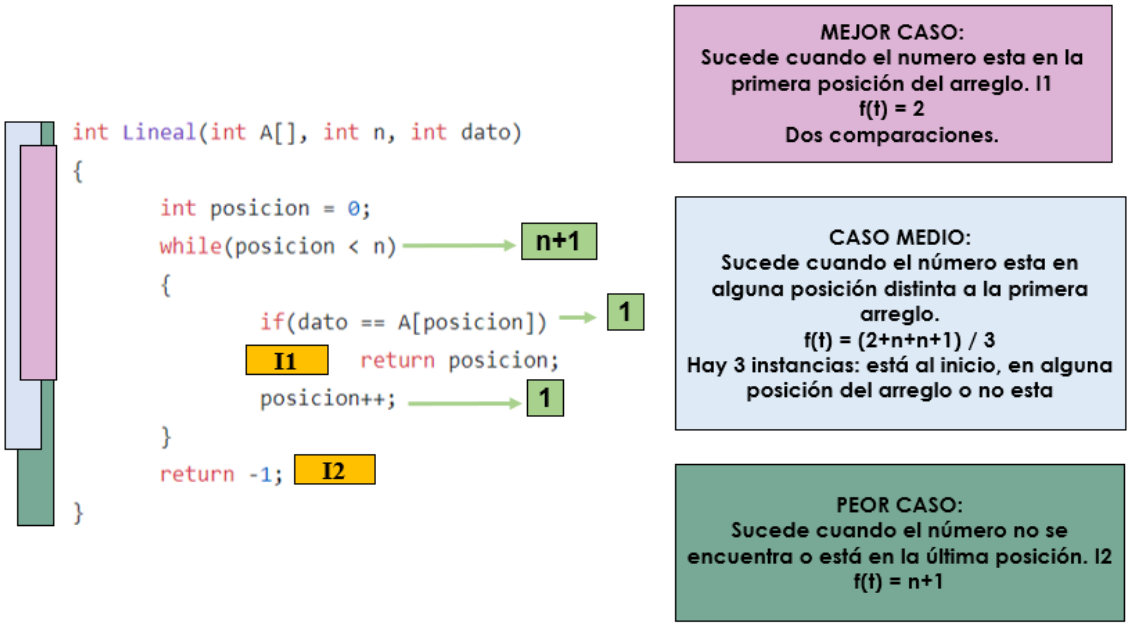
Compilador: GCC version 7.3.0 desde la Terminal

Sistema Operativo: Linux Ubuntu 18.04.1 LTS x64

3. Actividades y Pruebas

3.1. Búsqueda Lineal

3.1.1. Análisis teórico a priori



3.1.2. Ejecución del algoritmo

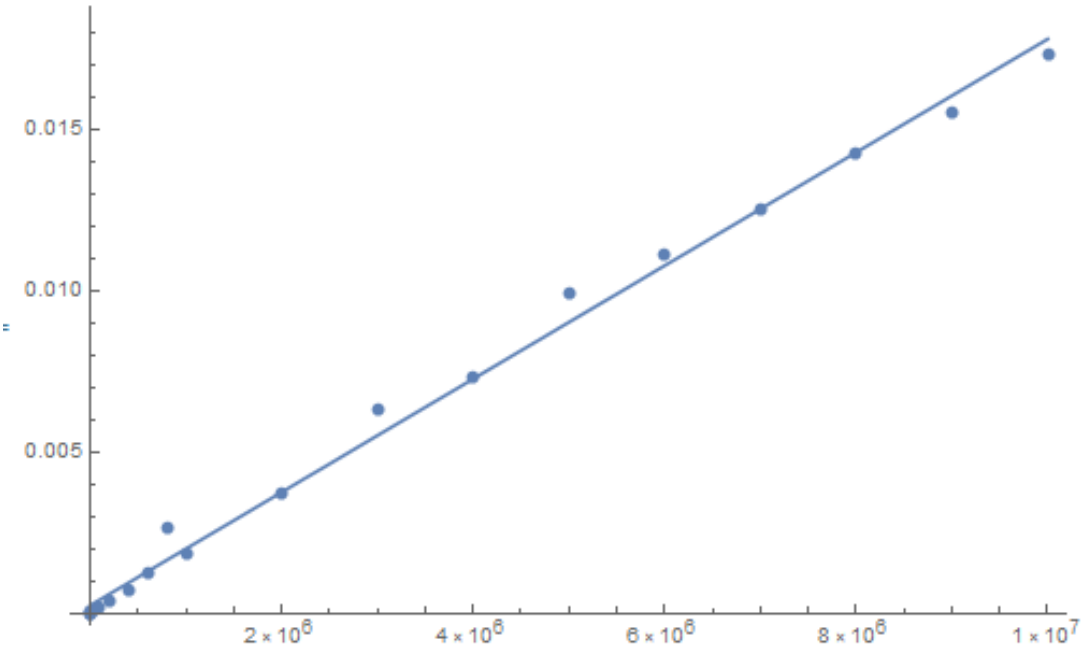
```
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$ gcc tiempo.c -c
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$ gcc Lineal.c tiempo.o -o Lineal
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$ ./Lineal 10000000 <ordenados.txt

Busqueda Lineal n = 10000000

2109248666 SI : 9822540
Total 2.441692352294922e-02
CPU 2.435199999999998e-02
E/S 0.000000000000000e+00
CPU/Wall 99.73410441 %

Promedio Tiempo Total: 0.01938166655600070953 s

-----------------------------------
```

3.1.3. Análisis temporal promedio

Búsqueda lineal	Número a buscar	Encontrado	Tiempo Real (seg)	Tiempo CPU (seg)	Tiempo E/S (seg)	%CPU/Wall	Tiempo Real Promedio (seg)
100	2109248666	No	1.19E-06	1.00E-06	0.00E+00	83.88608	8.70228E-07
1,000	2109248666	No	3.10E-06	3.00E-06	0.00E+00	96.79163077	2.77758E-06
5,000	2109248666	No	2.48E-05	0.00E+00	2.50E-05	100.8246154	1.56999E-05
10,000	2109248666	No	2.29E-05	0.00E+00	2.30E-05	100.4885333	2.10762E-05
50,000	2109248666	No	1.11E-04	0.00E+00	1.11E-04	100.1220955	0.000100839
100,000	2109248666	No	2.29E-04	7.60E-05	1.53E-04	99.94751467	0.000190556
200,000	2109248666	No	4.44E-04	4.44E-04	0.00E+00	100.014553	0.000375354
400,000	2109248666	No	9.05E-04	9.04E-04	0.00E+00	99.88542719	0.000731313
600,000	2109248666	No	1.50E-03	1.50E-03	0.00E+00	99.80080344	0.001224697
800,000	2109248666	No	1.85E-03	1.85E-03	0.00E+00	99.88497155	0.002665651
1,000,000	2109248666	No	2.43E-03	2.43E-03	0.00E+00	99.92312471	0.001823199
2,000,000	2109248666	No	4.80E-03	4.80E-03	0.00E+00	99.95557225	0.003687847
3,000,000	2109248666	No	7.28E-03	7.20E-03	0.00E+00	99.00888678	0.006287658
4,000,000	2109248666	No	9.82E-03	9.78E-03	9.78E-03	99.59231453	0.007306921
5,000,000	2109248666	No	1.22E-02	1.22E-02	0.00E+00	99.94390291	0.009912563
6,000,000	2109248666	No	1.47E-02	1.47E-02	0.00E+00	99.93852244	0.011102915
7,000,000	2109248666	No	1.67E-02	1.66E-02	0.00E+00	99.67072743	0.012512947
8,000,000	2109248666	No	2.11E-02	2.10E-02	0.00E+00	99.55314942	0.014274967
9,000,000	2109248666	No	2.13E-02	2.12E-02	0.00E+00	99.8724432	0.015519405
10,000,000	2109248666	Si	2.34E-02	2.33E-02	0.00E+00	99.7050527	0.017345857

3.1.4. Gráfica de comportamiento

3.1.5. Aproximación Polinomial



Grado 1: $0.000236308 + 1.75588 \times 10^{-9}x$

3.1.6. Tiempo por cada operación básica

Tomamos la función de complejidad temporal del peor caso. Peor Caso $f_{tpc}(n) = n + 1$

Luego, tomamos el polinomio obtenido en la aproximación polinomial en función de n:

$P(n) = 0.000236308 + 1.75588 \times 10^{-9}x$

Utilizaremos un tamaño de problema n = 10000000 para el cálculo.

Calculamos el número de operaciones para un tamaño de problema n con la función de complejidad temporal del peor caso:

$$f_{tpc}(10000000) = 10000000 + 1 = 10000001\ operaciones$$

Ahora, calculamos el tiempo en segundos con el polinomio:

$$P(10000000) = 0.0177951\ segundos$$

El tiempo en que tarda cada operación básica es:

$$Tiempo\ x\ op.\ básica = \frac{0.0177951}{10000001} = 1.7795 \times 10^{-9}\ segundos$$

3.1.7. Evaluación de tamaños de problema n's

Para n = 50000000

$P(50000000) = 0.0880303\ segundos$

Para n = 100000000

$P(100000000) = 0.175824\ segundos$

Para n = 500000000

$P(500000000) = 0.878176\ segundos$

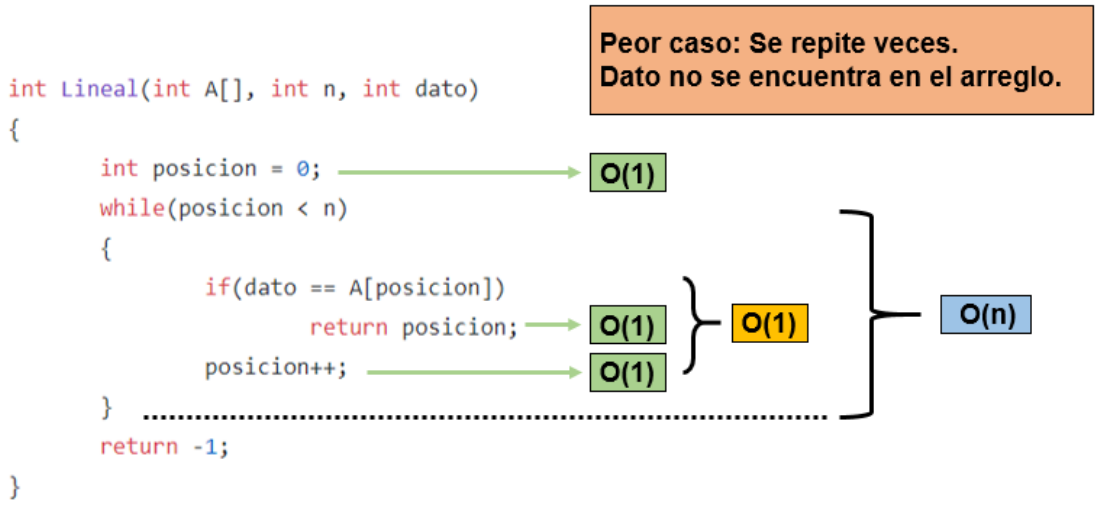
Para n = 1000000000

$P(1000000000) = 1.75612\ segundos$

Para n = 5000000000

$P(5000000000) = 8.77964\ segundos$

3.1.8. Cota O mayúscula del algoritmo



3.1.9. Cota O mayúscula del polinomio

Al tratarse de un polinomio de grado 1, su cota de complejidad O sera $O(n)$.

3.2. Búsqueda Lineal (Hilos)

3.2.1. Funcionamiento

Según la cantidad n de hilos que se envíen como parámetro al ejecutar el programa, estos dividirán el arreglo en n partes. En cada una de ellas se ejecutará de manera simultanea el algoritmo de búsqueda lineal indicado anteriormente; va buscando posición por posición el número en el pedazo del arreglo. Con ayuda del número de hilos podemos establecer los rangos, con un índice (posición del arreglo) de inicio y otro de fin, que le corresponderán a cada hilo en su ejecución.

3.2.2. Ejecución del algoritmo

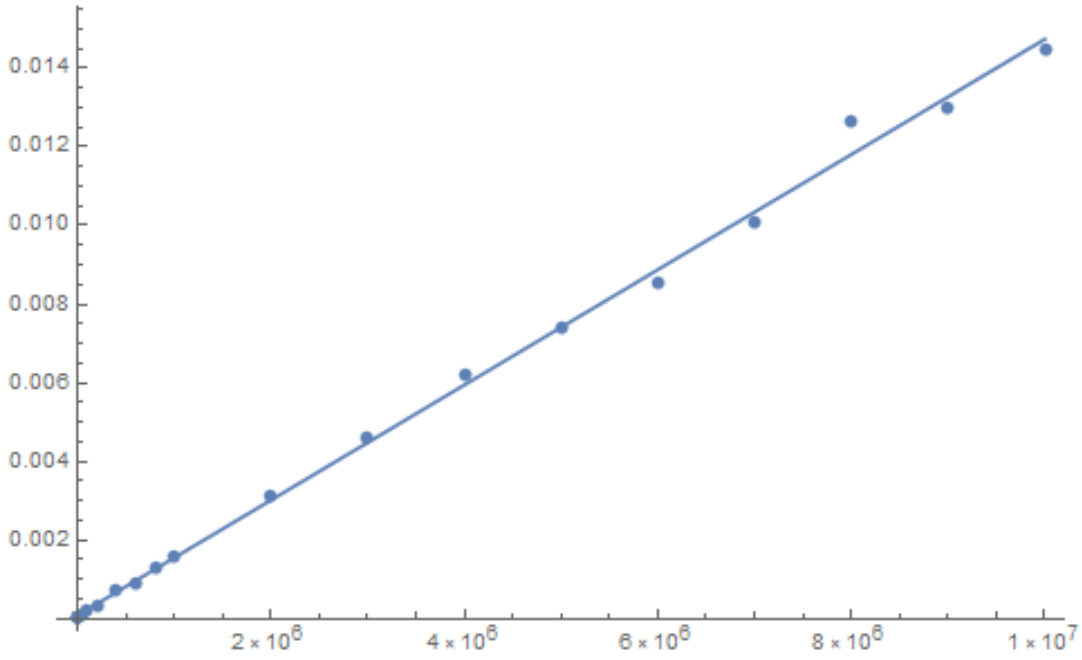
```
pointer-to-int-cast]
  int n_thread=(int)id, inicio, fin;
              ^
LinealHilos.c: In function ‘main’:
LinealHilos.c:105:53: warning: cast to pointer from integer of different size [-
Wint-to-pointer-cast]
    if (pthread_create(&thread[i], NULL, LinealHilos,(void*)i) != 0 )
                                                     ^
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$ ./LinealHilos 2 10000000 <ordenados.txt

Busqueda Lineal Hilos n = 10000000


2109248666 SI : 0
Total  2.137088775634766e-02
CPU's 2.126299999999998e-02
Hilos 1.063149999999999e-02
E/S 0.000000000000000e+00
CPU/Wall 99.49516484 %


Promedio Tiempo Total: 0.01612327061593532562 s
```

3.2.3. Análisis temporal promedio

Búsqueda Lineal - Hilos	Numero a buscar	Encontrado	Tiempo Real (seg)	Tiempo CPU (seg)	Tiempo por Hilo - 2 (seg)	Tiempo E/S (seg)	% CPU/Wall	Tiempo Real Promedio (seg)
100	2109248666	No	1.60E-05	0.00E+00	0.00E+00	1.40E-05	87.64%	1.91808E-05
1,000	2109248666	No	1.72E-05	0.00E+00	0.00E+00	1.50E-05	87.38133333	2.19584E-05
5,000	2109248666	No	2.72E-05	0.00E+00	0.00E+00	2.50E-05	91.98035088	3.016E-05
10,000	2109248666	No	3.60E-05	0.00E+00	0.00E+00	3.40E-05	94.44128212	4.41432E-05
50,000	2109248666	No	1.25E-04	0.00E+00	0.00E+00	1.23E-04	98.45408244	0.00011034
100,000	2109248666	No	2.39E-04	2.36E-04	1.18E-04	0.00E+00	98.68950588	0.000185752
200,000	2109248666	No	4.23E-04	3.20E-05	1.60E-05	3.88E-04	99.30144758	0.000329947
400,000	2109248666	No	8.66E-04	8.62E-04	4.31E-04	0.00E+00	99.54543084	0.000737524
600,000	2109248666	No	1.22E-03	1.22E-03	6.08E-04	0.00E+00	99.6690663	0.000922275
800,000	2109248666	No	1.78E-03	1.77E-03	8.87E-04	0.00E+00	99.78135036	0.001324618
1,000,000	2109248666	No	2.12E-03	2.11E-03	1.06E-03	0.00E+00	99.81716375	0.001581335
2,000,000	2109248666	No	4.39E-03	4.38E-03	2.19E-03	0.00E+00	99.77649509	0.003110409
3,000,000	2109248666	No	6.53E-03	6.42E-03	3.21E-03	0.00E+00	98.40835974	0.004633558
4,000,000	2109248666	No	8.56E-03	8.55E-03	4.27E-03	0.00E+00	99.78987189	0.006227768
5,000,000	2109248666	No	1.06E-02	1.06E-02	5.29E-03	0.00E+00	99.6613171	0.007418943
6,000,000	2109248666	No	1.30E-02	1.30E-02	6.49E-03	0.00E+00	99.8541174	0.008559358
7,000,000	2109248666	No	1.47E-02	1.46E-02	7.32E-03	0.00E+00	99.76802058	0.010078263
8,000,000	2109248666	No	1.69E-02	1.69E-02	8.46E-03	0.00E+00	99.79272106	0.012636209
9,000,000	2109248666	No	1.93E-02	1.92E-02	9.61E-03	0.00E+00	99.71988623	0.013014209
10,000,000	2109248666	Si	2.20E-02	2.19E-02	1.10E-02	0.00E+00	99.55040934	0.01450572

3.2.4. Gráfica de comportamiento

3.2.5. Aproximación Polinomial



Grado 1: $0.0000858628 + 1.46546 * 10^{-9}x$

3.2.6. Evaluación de tamaños de problema n's

Para n = 50000000

$P(50000000) = 0.0733589\ segundos$

Para n = 100000000

$P(100000000) = 0.146632\ segundos$

Para n = 500000000

$P(500000000) = 0.732816\ segundos$

Para n = 1000000000

$P(1000000000) = 1.46555\ segundos$

Para n = 5000000000

$P(5000000000) = 7.32739\ segundos$

3.2.7. Cotas O mayúscula del polinomio

Al tratarse de un polinomio de grado 1, su cota de complejidad O sera $O(n)$.

3.3. Búsqueda Binaria

3.3.1. Análisis teórico a priori

```
int Binaria(int A[], int n, int dato)
{
   //centro: subíndice central del intervalo
     //inf: límite inferior del intervalo
     //sup: límite superior del intervalo
   int centro, inf = 0, sup = n-1;
   while(inf <= sup)                          log2 n
   {
      centro = ((sup + inf)/2);                  1
      if(A[centro] == dato)                      1
      {                  I1
         /*Para imprimir la posicion y el dato de
         printf("SI %d : %d", A[centro], centro);
         return centro;
      }
      else if (dato < A[centro])                 1
         sup = centro - 1;                       1      I2
      else
         inf = centro + 1;                       1
   }                    I3
   return -1;
}
```

MEJOR CASO:
Sucede cuando el numero esta justo en la mitad del arreglo. I1
f(t) = 3
Dos comparaciones y una asignación.

CASO MEDIO:
Sucede cuando el número esta en alguna posición diferente a la mitad del arreglo.
f(t) = $(3 + 5\log_2 n + 5\log_2 n)$ / 3
Hay 3 instancias: está en el centro, en el segmento izquierdo o en el segmento derecho.

PEOR CASO:
Sucede cuando el número no se encuentra o está en la última posición. I3
f(t) = $5\log_2 n$

3.3.2. Ejecución del algoritmo

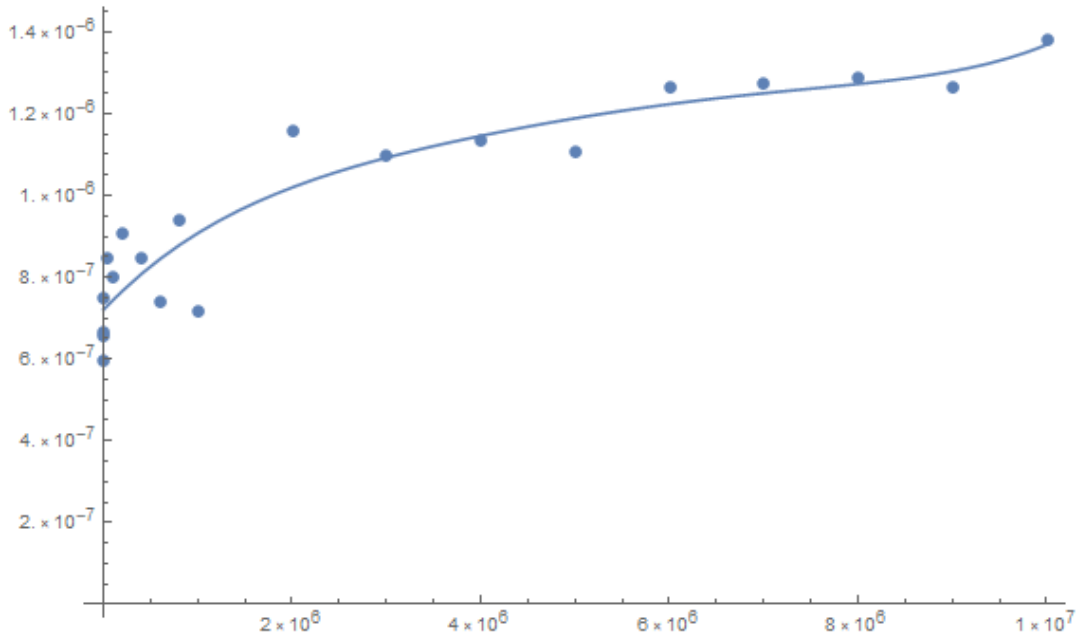
```
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$ gcc tiempo.c -c
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$ gcc Binaria.c tiempo.o -o Binaria
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$ ./Binaria 10000000 <ordenados.txt

Busqueda Binaria n = 10000000

2109248666 SI : 9822540
Total 9.536743164062500e-07
CPU 0.000000000000000e+00
E/S 1.999999999835467e-06
CPU/Wall 209.71519998 %

Promedio Tiempo Total: 0.00000137090682983398 s

-----------------------------------
```

3.3.3. Análisis temporal promedio

Búsqueda Binaria	Numero a buscar	Encontrado	Tiempo Real (seg)	Tiempo CPU (seg)	Tiempo E/S (seg)	% CPU/Wall	Tiempo Real Promedio (seg)
100	2,109,248,666	No	1.19E-06	0	0	0	6.67572E-07
1,000	2,109,248,666	No	0	1.00E-06	0	inf	7.51019E-07
5,000	2,109,248,666	No	9.54E-07	0	0	0	6.55651E-07
10,000	2,109,248,666	No	9.54E-07	1.00E-06	0	104.8576	5.96046E-07
50,000	2,109,248,666	No	9.54E-07	0	1.00E-06	104.8576	8.46386E-07
100,000	2,109,248,666	No	0	0	0	- nan	7.98702E-07
200,000	2,109,248,666	No	9.54E-07	0	1.00E-06	104.8576	9.05991E-07
400,000	2,109,248,666	No	0	0	1.00E-06	inf	8.46386E-07
600,000	2,109,248,666	No	9.54E-07	0	0	0	7.39098E-07
800,000	2,109,248,666	No	9.54E-07	0	1.00E-06	104.8576	9.41753E-07
1,000,000	2,109,248,666	No	0	0	1.00E-06	inf	7.15256E-07
2,000,000	2,109,248,666	No	0	0	1.00E-06	inf	1.15633E-06
3,000,000	2,109,248,666	No	9.54E-07	0	1.00E-06	104.8576	1.09673E-06
4,000,000	2,109,248,666	No	9.54E-07	0	1.00E-06	104.8576	1.13249E-06
5,000,000	2,109,248,666	No	9.54E-07	1.00E-06	1.00E-06	209.7152	1.10865E-06
6,000,000	2,109,248,666	No	9.54E-07	0	1.00E-06	104.8576	1.26362E-06
7,000,000	2,109,248,666	No	9.54E-07	0	0	0	1.27554E-06
8,000,000	2,109,248,666	No	0	0	1.00E-06	inf	1.28746E-06
9,000,000	2,109,248,666	No	9.54E-07	0	1.00E-06	104.8576	1.26362E-06
10,000,000	2,109,248,666	Si	1.91E-06	0	2.00E-06	104.8576	1.38283E-06

3.3.4. Gráfica de comportamiento

3.3.5. Aproximación Polinomial



Grado 5: $7.20374 \times 10^{-7} + 2.42704 \times 10^{-13}x - 6.58073 \times 10^{-20}x^2 + 1.16759 \times 10^{-26}x^3 - 1.10283 \times 10^{-33}x^4 + 4.15462 \times 10^{-41}x^5$

3.3.6. Tiempo por cada operación básica

Tomamos la función de complejidad temporal del peor caso. Peor Caso $f_{tpc}(n) = 5log_2 n$

Luego, tomamos el polinomio obtenido en la aproximación polinomial en función de n: $P(n) = 7.20374 \times 10^{-7} + 2.42704 \times 10^{-13}n - 6.58073 \times 10^{-20}n^2 + 1.16759 \times 10^{-26}n^3 - 1.10283 \times 10^{-33}n^4 + 4.15462 \times 10^{-41}n^5$

Utilizaremos un tamaño de problema n = 10000000 para el cálculo.

Calculamos el número de operaciones para un tamaño de problema n con la funcion de complejidad temporal del peor caso:

$$f_{tpc}(10000000) = 5log_2 10000000 = 116\ operaciones$$

Ahora, calculamos el tiempo en segundos con el polinomio:

$$P(10000000) = 1.3689 \times 10^{-6}\ segundos$$

El tiempo en que tarda cada operación básica es:

$$Tiempo\ x\ op.\ básica = \frac{1.3689 \times 10^{-6}}{116} = 1.18 \times 10^{-8}\ segundos$$

3.3.7. Evaluación de tamaños de problema n's

Para n = 50000000

$P(50000000) = 0.00739832\ segundos$

Para n = 100000000

$P(100000000) = 0.316222\ segundos$

Para n = 500000000

$P(500000000) = 1230.84\ segundos$

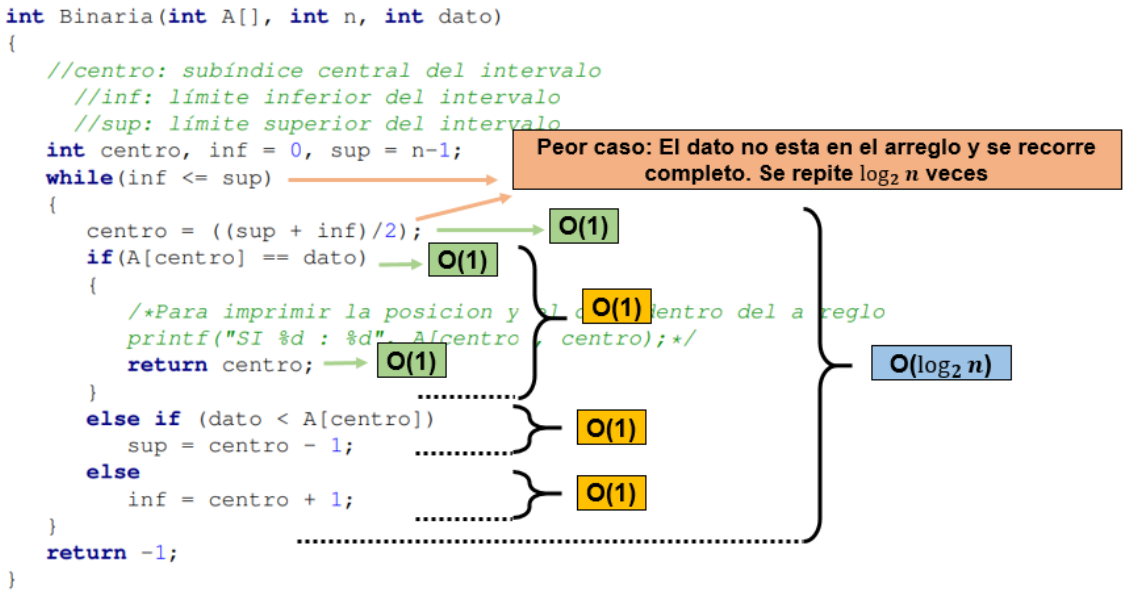
Para n = 1000000000

$P(1000000000) = 40455\ segundos$

Para n = 5000000000

$P(5000000000) = 1.29144 \times 10^8\ segundos$

3.3.8. Cota O mayúscula del algoritmo



3.3.9. Cota O mayúscula del polinomio

Al tratarse de un polinomio de grado 5, su cota de complejidad O sera $O(n^5)$.

3.4. Búsqueda Binaria (Hilos)

3.4.1. Funcionamiento

Según la cantidad n de hilos que se envíen como parámetro al ejecutar el programa, estos dividirán el arreglo en n partes. En cada una de ellas se ejecutará de manera simultanea el algoritmo de búsqueda binaria indicado anteriormente; se divide en dos cada pedazo del arreglo y se recorre hasta encontrar el número, ya sea a la izquierda o derecha. Con ayuda del número de hilos podemos establecer los rangos, con un índice (posición del arreglo) de inicio y otro de fin, que le corresponderán a cada hilo en su ejecución.

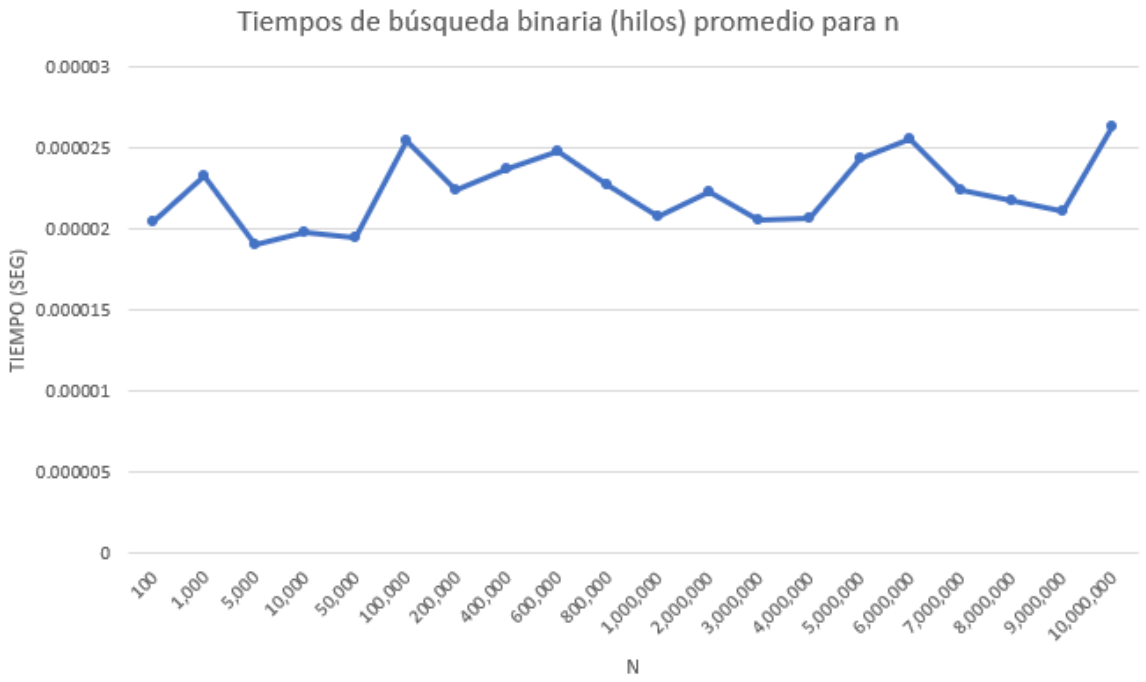
3.4.2. Ejecución del algoritmo

```
Wpointer-to-int-cast]
  int n_thread=(int)id, inicio, fin;
               ^
BinariaHilos.c: In function ‘main’:
BinariaHilos.c:112:54: warning: cast to pointer from integer of different size [
-Wint-to-pointer-cast]
    if (pthread_create(&thread[i], NULL, BinariaHilos,(void*)i) != 0 )
                                                      ^
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$ ./BinariaHilos 2 10000000 <ordenados.txt

Busqueda Binaria Hilos n = 10000000


2109248666 SI : 9822540
Total  4.506111145019531e-05
CPU's 2.000000000002000e-06
Hilo 1.000000000001000e-06
E/S 4.099999999995774e-05
CPU/Wall 95.42596402 %


Promedio Tiempo Total: 0.00002386569940426853 s
```

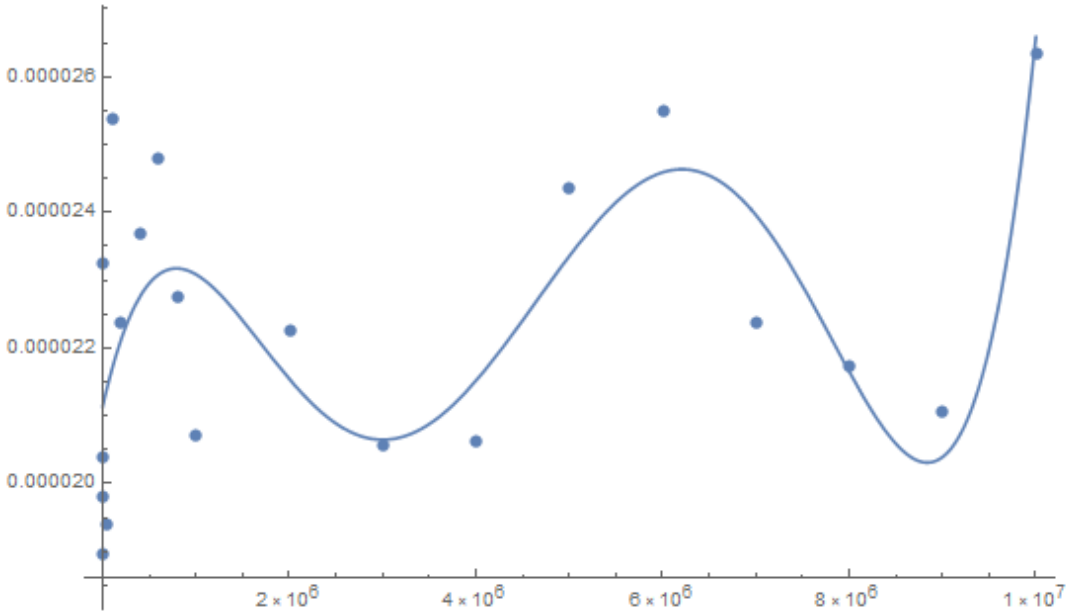
3.4.3. Análisis temporal promedio

Búsqueda Binaria - Hilos	Numero a buscar	Encontrado	Tiempo Real (seg)	Tiempo CPU (seg)	Tiempo por Hilo - 2 (seg)	Tiempo E/S (seg)	% CPU/Wall	Tiempo Real Promedio (seg)
100	2,109,248,666	No	1.60E-05	0	0	1.40E-05	87.64217313	2.03848E-05
1,000	2,109,248,666	No	1.72E-05	1.60E-05	8.00E-06	0	93.20675556	2.32577E-05
5,000	2,109,248,666	No	1.69E-05	7.00E-06	3.50E-06	7.00E-06	82.70458592	1.89662E-05
10,000	2,109,248,666	No	1.60E-05	1.40E-05	7.00E-06	0	87.64217313	1.98126E-05
50,000	2,109,248,666	No	1.69E-05	0	0	1.50E-05	88.61205634	1.94073E-05
100,000	2,109,248,666	No	1.60E-05	5.00E-06	2.50E-06	1.00E-05	93.90232836	2.53797E-05
200,000	2,109,248,666	No	1.69E-05	0	0	1.40E-05	82.70458592	2.23756E-05
400,000	2,109,248,666	No	1.60E-05	1.00E-06	5.00E-07	1.40E-05	93.90232836	2.3675E-05
600,000	2,109,248,666	No	1.60E-05	1.00E-06	5.00E-07	1.40E-05	93.90232836	2.48075E-05
800,000	2,109,248,666	No	1.69E-05	2.00E-06	1.00E-06	1.30E-05	88.61205634	2.27451E-05
1,000,000	2,109,248,666	No	1.69E-05	1.00E-06	5.00E-07	1.40E-05	88.61205634	2.07186E-05
2,000,000	2,109,248,666	No	1.69E-05	1.00E-06	5.00E-07	1.40E-05	88.61205634	2.22445E-05
3,000,000	2,109,248,666	No	1.69E-05	1.00E-06	5.00E-07	1.30E-05	82.70458592	2.05636E-05
4,000,000	2,109,248,666	No	1.69E-05	0	0	1.40E-05	82.70458592	2.06113E-05
5,000,000	2,109,248,666	No	6.51E-05	4.00E-06	2.00E-06	5.90E-05	96.79163077	2.43545E-05
6,000,000	2,109,248,666	No	1.72E-05	1.00E-06	5.00E-07	1.40E-05	87.38133333	2.54989E-05
7,000,000	2,109,248,666	No	1.69E-05	1.00E-06	5.00E-07	1.40E-05	88.61205634	2.23875E-05
8,000,000	2,109,248,666	No	1.81E-05	1.00E-06	5.00E-07	1.40E-05	82.78231579	2.17199E-05
9,000,000	2,109,248,666	No	1.69E-05	1.00E-06	5.00E-07	1.40E-05	88.61205634	2.10524E-05
10,000,000	2,109,248,666	Si	1.81E-05	0	0	1.60E-05	88.30113684	2.63453E-05

3.4.4. Gráfica de comportamiento



3.4.5. Aproximación Polinomial



Grado 5: $0.0000211462 + 6.03491 \times 10^{-12}x - 5.65412 \times 10^{-18}x^2 + 1.76786 \times 10^{-24}x^3 - 2.1856 \times 10^{-31}x^4 + 9.28242 \times 10^{-39}x^5$

3.4.6. Evaluación de tamaños de problema n's

Para n = 50000000

$P(50000000) = 1.74193\ segundos$

Para n = 100000000

$P(100000000) = 72.6801 \times 10^6\ segundos$

Para n = 500000000

$P(500000000) = 276635\ segundos$

Para n = 1000000000

$P(1000000000) = 9.06562 \times 10^6\ segundos$

Para n = 5000000000

$P(5000000000) = 2.88712 \times 10^{10}\ segundos$

3.4.7. Cota O mayúscula del polinomio

Al tratarse de un polinomio de grado 5, su cota de complejidad O sera $O(n^5)$.

3.5. Árbol de Búsqueda Binaria

3.5.1. Análisis teórico a priori

```
int ABB(Arbin *a, int elemento)
{
    posicion a_aux = *a;
    int numero; // Auxiliar para comparar
    do
    {
        numero = a_aux -> raiz;          -> 1
        if(numero == elemento)
        {
            a_aux = NULL;
            return 0;
        }
        else if(numero < elemento)
        {
            a_aux = a_aux -> der;        } n
        }
        else
        {
            a_aux = a_aux -> izq;        } n
        }
    }
    while (a_aux != NULL); // Apuntador nulo
    return -1;
}
```

MEJOR CASO:
Sucede cuando el número esta en la raíz del árbol.
$f(t)=1$

CASO MEDIO:
Sucede cuando un número esta a la mitad del árbol.
Con un ABB balanceado: f(t)=$(\log n)$
Con un ABB no balanceado: $f(t)$ =(n)

PEOR CASO:
Sucede cuando el número no se encuentra o está en la última posición.
Con un ABB balanceado: f(t)=$(\log_2 n)$
Con un ABB no balanceado: $f(t)$ =(n)

3.5.2. ¿Por qué no usar hilos?

Por que no se puede hacer la segmentación de manera correcta como en los demás algoritmos. La segmentación y recorrido del árbol en diferentes rangos va desechando aquellos nodos en los que no se encuentra el número a buscar, por lo que solo causaría que en muchos de ellos no buscaran nada, lo cual seria un gasto de recursos al crear los hilos.

3.5.3. Ejecución del algoritmo

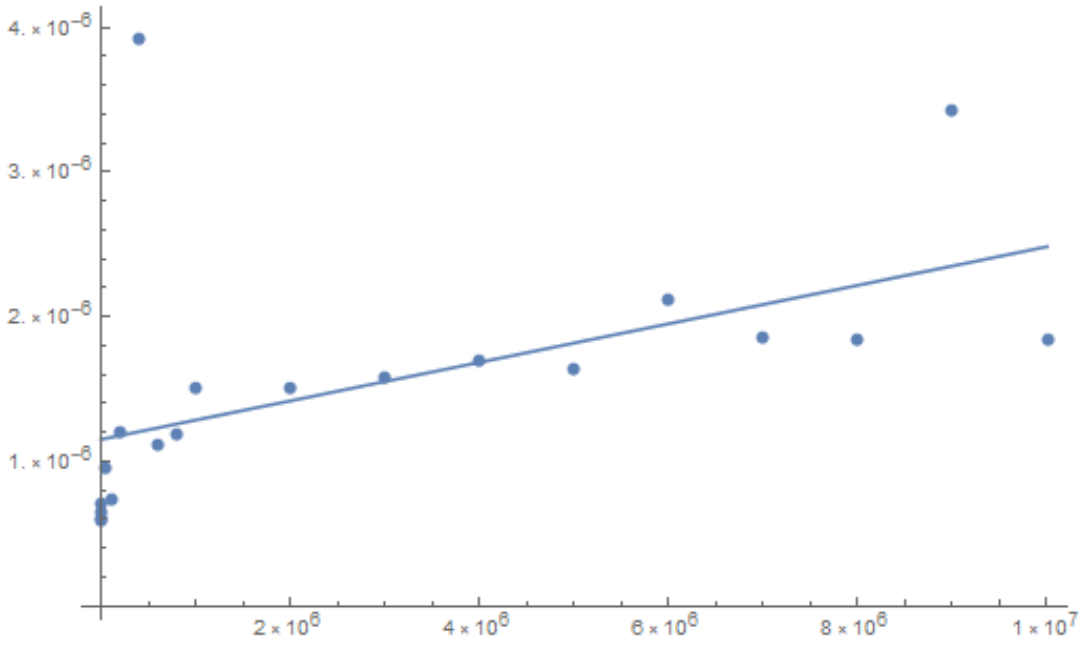
```
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$ gcc tiempo.c -c
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$ gcc ABB.c tiempo.o -o ABB
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$ ./ABB 10000000 <desordenados.txt
Arbol de Busqueda Binaria n = 10000000


2109248666 SI : 0
Total 2.145767211914062e-06
CPU 2.000000000279556e-06
E/S 0.000000000000000e+00
CPU/Wall 93.20675557 %

Promedio Tiempo Total: 0.00000184774398803711 s

-----------------------------------
```

3.5.4. Análisis temporal promedio

Búsqueda Binaria	Numero a buscar	Encontrado	Tiempo Real (seg)	Tiempo CPU (seg)	Tiempo E/S (seg)	% CPU/Wall	Tiempo Real Promedio (seg)
100	2109248666	NO	9.54E-07	1.00E-06	0.00E+00	10485.76%	6.4373E-07
1,000	2109248666	NO	9.54E-07	0.00E+00	1.00E-06	104.8576	6.07967E-07
5,000	2109248666	NO	0.00E+00	0.00E+00	1.00E-06	inf	5.96046E-07
10,000	2109248666	NO	9.54E-07	1.00E-06	0.00E+00	104.8576	7.03335E-07
50,000	2109248666	NO	9.54E-07	0.00E+00	0.00E+00	0	9.53674E-07
100,000	2109248666	NO	9.54E-07	1.00E-06	0.00E+00	104.8576	7.39098E-07
200,000	2109248666	NO	9.54E-07	1.00E-06	1.00E-06	209.7152	1.20401E-06
400,000	2109248666	NO	9.54E-07	1.00E-06	0.00E+00	104.8576	3.92199E-06
600,000	2109248666	NO	9.54E-07	1.00E-06	0.00E+00	104.8576	1.12057E-06
800,000	2109248666	NO	1.91E-06	1.00E-06	1.00E-06	104.8576	1.19209E-06
1,000,000	2109248666	NO	2.15E-06	2.00E-06	1.00E-06	139.810133	1.50204E-06
2,000,000	2109248666	NO	1.91E-06	2.00E-06	0.00E+00	104.8576	1.50204E-06
3,000,000	2109248666	NO	1.91E-06	1.00E-06	0.00E+00	52.4288	1.58548E-06
4,000,000	2109248666	NO	9.54E-07	2.00E-06	0.00E+00	209.7152	1.69277E-06
5,000,000	2109248666	NO	1.91E-06	2.00E-06	1.00E-06	157.2864	1.63317E-06
6,000,000	2109248666	NO	2.15E-06	2.00E-06	0.00E+00	93.2067556	2.12193E-06
7,000,000	2109248666	SI	1.91E-06	2.00E-06	0.00E+00	104.8576	1.85966E-06
8,000,000	2109248666	SI	1.91E-06	2.00E-06	0.00E+00	104.8576	1.83582E-06
9,000,000	2109248666	SI	3.10E-06	2.00E-06	0.00E+00	64.5277539	3.42131E-06
10,000,000	2109248666	SI	2.15E-06	1.00E-06	1.00E-06	93.2067556	1.83582E-06

3.5.5. Gráfica de comportamiento

3.5.6. Aproximación Polinomial



Grado 1: $1.15287 \times 10^{-6} + 1.33212 \times 10^{-13}x$

3.5.7. Tiempo por cada operación básica

Tomamos la función de complejidad temporal del peor caso. Peor Caso $f_{tpc}(n) = n$

Luego, tomamos el polinomio obtenido en la aproximación polinomial en función de n: $P(n) = 1.15287 \times 10^{-6} + 1.33212 \times 10^{-13}x$

Utilizaremos un tamaño de problema n = 10000000 para el cálculo.

Calculamos el número de operaciones para un tamaño de problema n con la función de complejidad temporal del peor caso:

$$f_{tpc}(10000000) = 10000000 = 10000000\ operaciones$$

Ahora, calculamos el tiempo en segundos con el polinomio:

$$P(10000000) = 2.48499 \times 10^{-6}\ segundos$$

El tiempo en que tarda cada operación básica es:

$$Tiempo\ x\ op.\ básica = \frac{2.48499 \times 10^{-6}}{10000000} = 2.48499 \times 10^{-13}\ segundos$$

3.5.8. Evaluación de tamaños de problema n's

Para n = 50000000

$P(50000000) = 7.81347 \times 10^{-6}\ segundos$

Para n = 100000000

$P(100000000) = 0.0000144741\ segundos$

Para n = 500000000

$P(500000000) = 0.0000677589\ segundos$

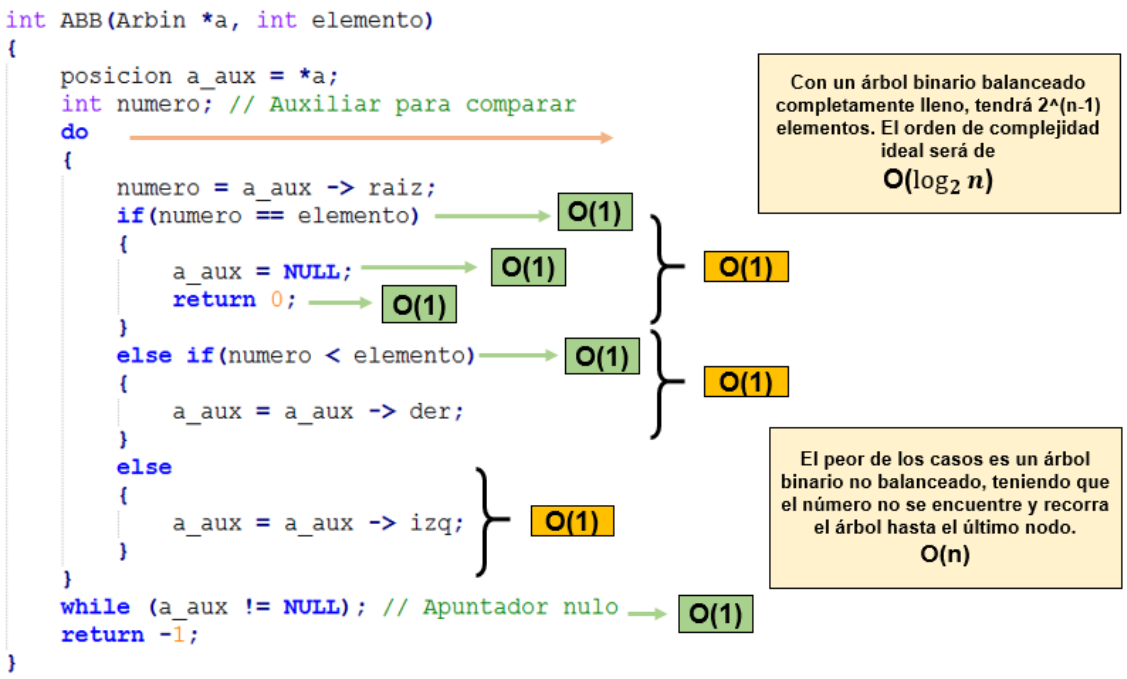
Para n = 1000000000

$P(1000000000) = 0.000134365\ segundos$

Para n = 5000000000

$P(5000000000) = 0.000667213\ segundos$

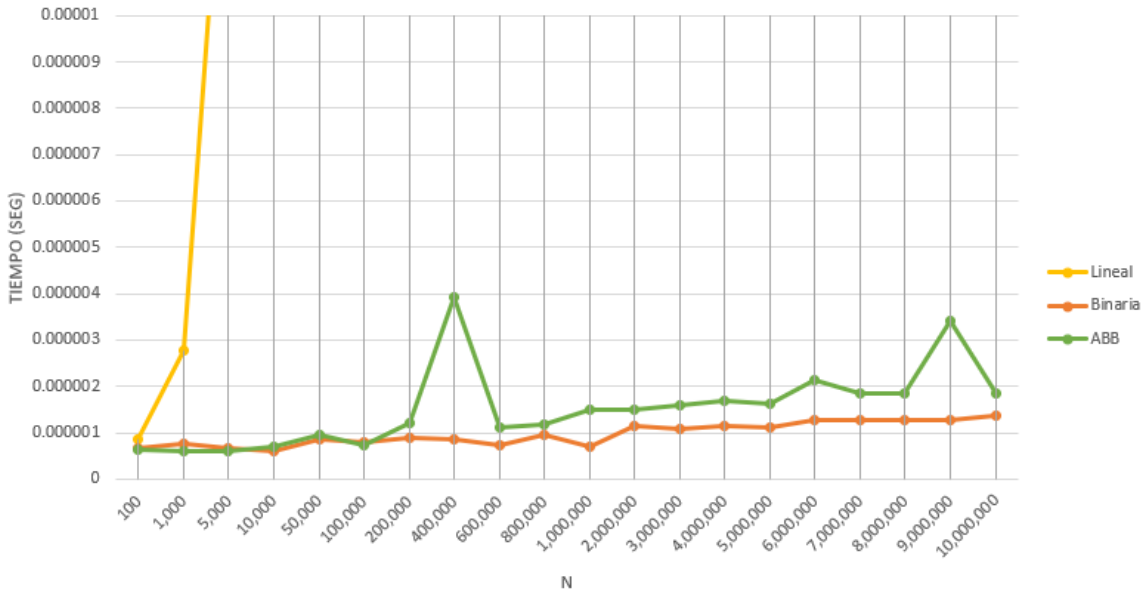
3.5.9. Cota O mayúscula del algoritmo

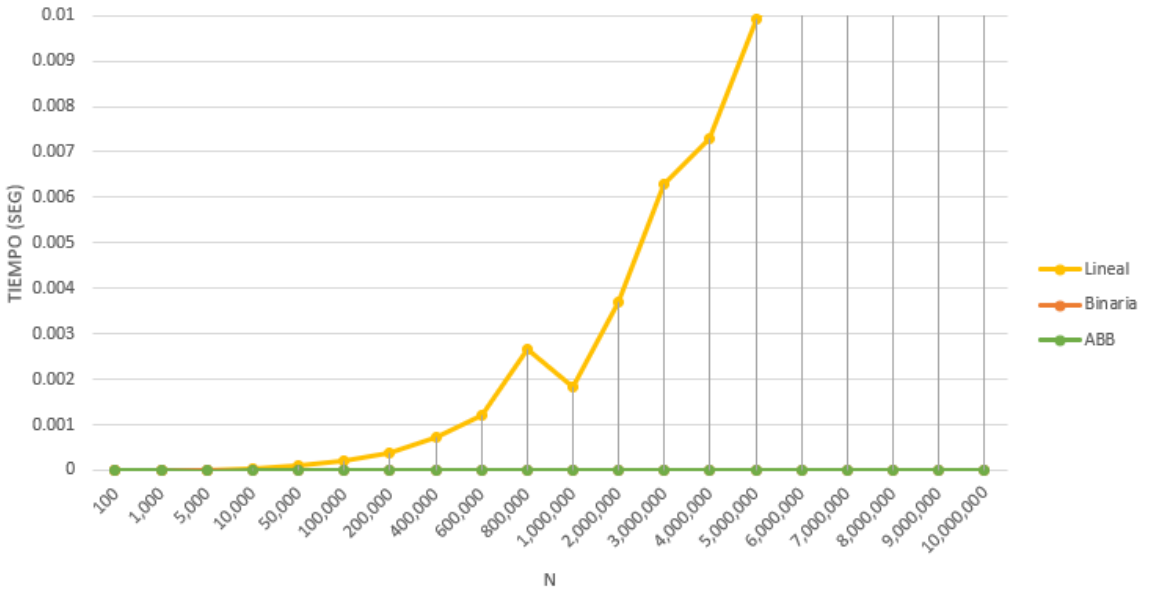


3.5.10. Cota O mayúscula del polinomio

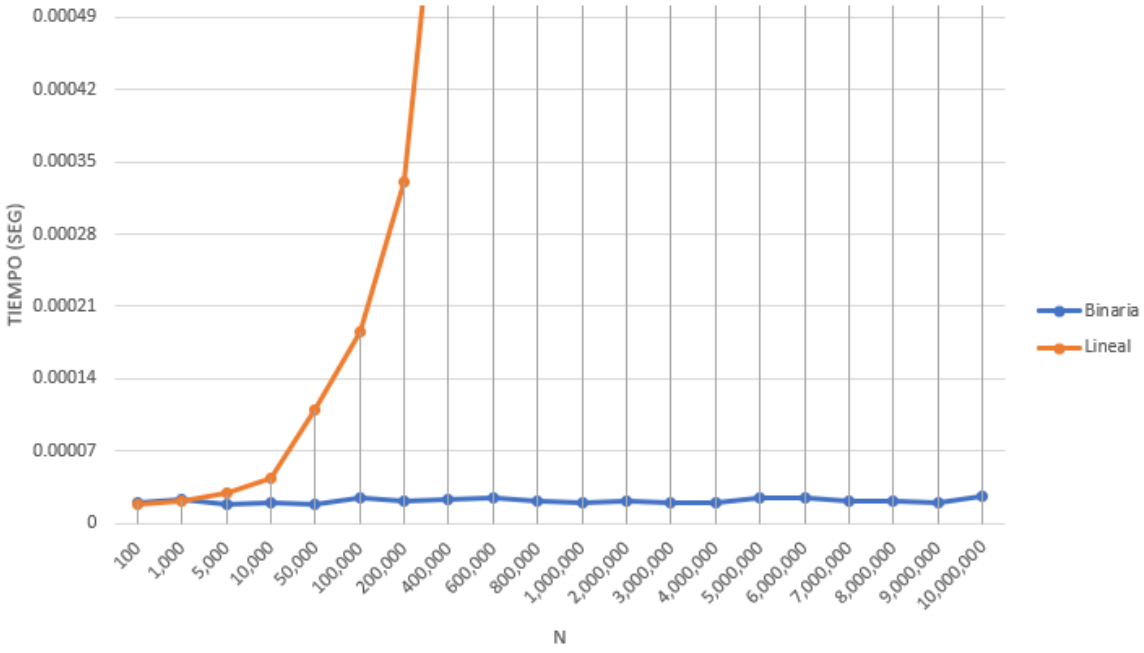
Al tratarse de un polinomio de grado 1, su cota de complejidad O sera $O(n)$.

3.6. Comparativa gráfica de comportamiento (Tiempo Real promedio)

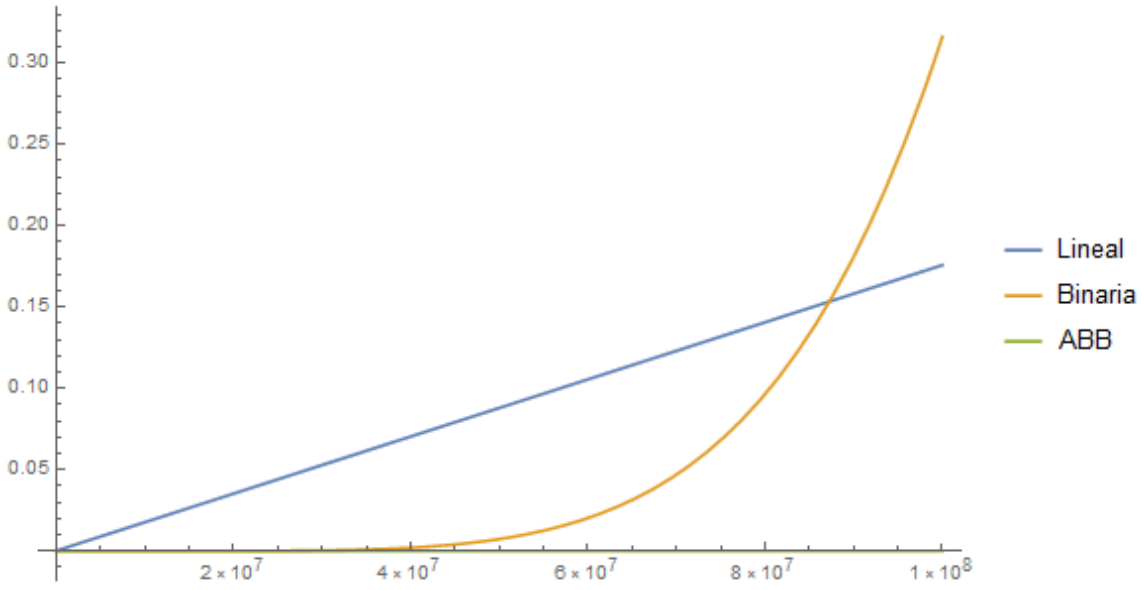


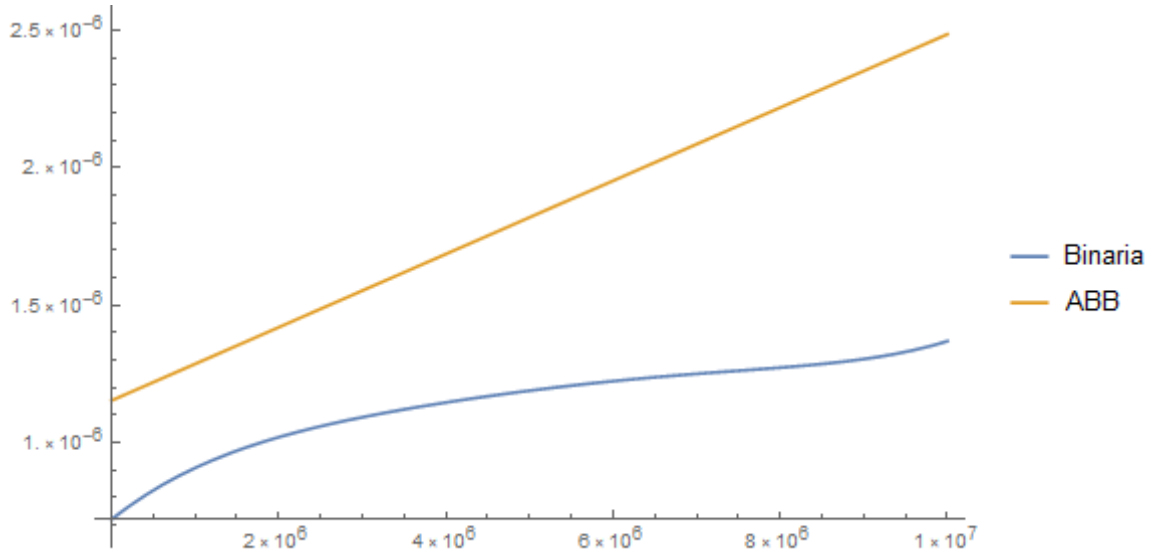


3.7. Comparativa gráfica de comportamiento con Hilos (Tiempo Real promedio)

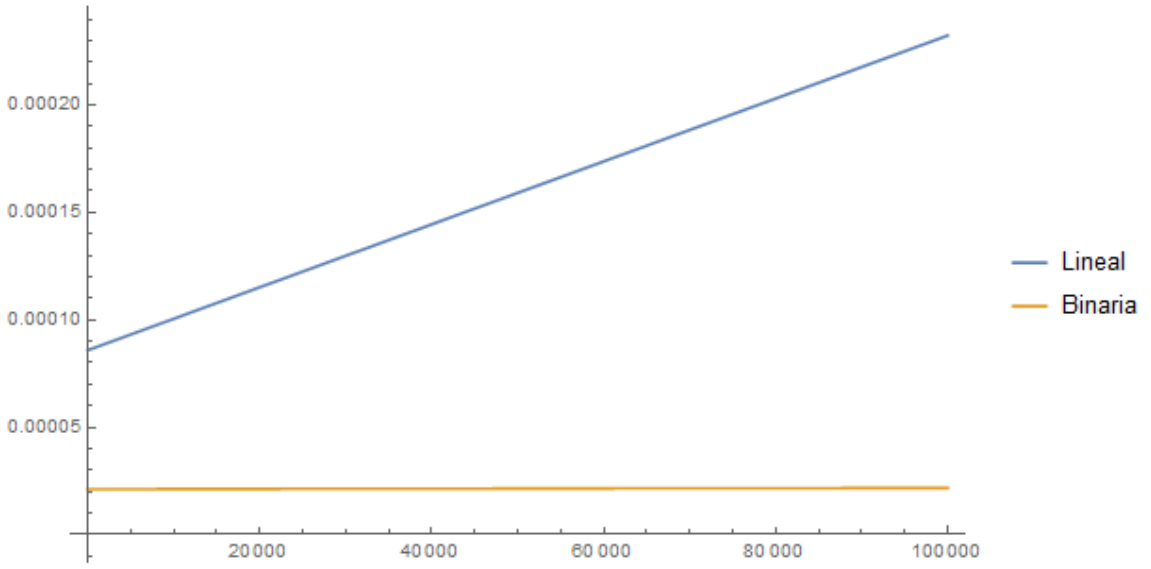


3.8. Comparativa de aproximaciones polinomiales





3.9. Comparativa de aproximaciones polinomiales (Hilos)



3.10. Cuestionario

1. ¿Cuál de los 3 algoritmos es más fácil de implementar?
   El algoritmo de Búsqueda Lineal, ya que las operaciones que utiliza son menores en comparación con los otros algoritmos.

2. ¿Cuál de los 3 algoritmos es más difícil de implementar?
   Podría considerase que el algoritmo del Árbol de Búsqueda Binaria fue el más complejo debido al uso de apuntadores que se realizó, los otros dos no hacen uso de éstos.

3. ¿Cuál de los 3 algoritmos es el más difícil de implementar en su variante con hilos?
   El algoritmo de Búsqueda Binaria, pues primero fue necesario identificar en que sección del código se debía de colocar el algoritmo. Además, había que tomar en cuenta el inicio y el fin de cada hilo, y adecuarlo a las variables de control.

4. ¿Cuál de los 3 algoritmos en su variante con hilos resultó ser más rápido?
   El de Búsqueda Lineal, ya que se repartió el trabajo de recorrido y comparación posición por posición entre varios hilos simultáneos.

5. ¿Cuál algoritmo tiene menor complejidad temporal?
   El de Búsqueda Binaria.

6. ¿Cuál algoritmo tiene mayor complejidad temporal?
   El Árbol de Búsqueda Binaria, teniendo en cuenta que no este balanceado, y la Búsqueda Lineal.

7. ¿El comportamiento experimental de los algoritmos era el esperado?¿Por qué?
   En la implementación normal de éstos sí. En cambio, en su implementación con hilos sólo hubo mejora en la Búsqueda Lineal, pues en la Binaria el tiempo incluso aumento. Estos es debido a que se utiliza tiempo para crear los hilos, establecer los rangos del arreglo, inicializarlos y esperar a que terminen. Esto significa un uso de recursos innecesario.

8. ¿Sus resultados experimentales difieren mucho de los análisis teóricos que se analizaron? ¿A que se debe?
   A grandes rasgos no. Ambos análisis se encuentran dentro de la misma cota de complejidad. En algunos tamaños de problema existen tiempos muy altos o muy pequeños que no siguen el comportamiento esperado del algoritmo.

9. ¿Los resultados experimentales de las implementaciones con hilos de los algoritmos realmente tardaron F(t)/hilos de su implementación sin hilos?
   Sí, pues se hicieron pruebas experimentales en las implementaciones con hilos de los algoritmos creando un único hilo, para comparar los resultados con los de su implementación sin hilos, y efectivamente los tiempos coinciden.

10. ¿Cuál es el % de mejora que tiene cada uno de los algoritmos en su variante con hilos?¿Es lo que esperabas?¿Por qué?
    Para la Búsqueda Lineal, utilizando dos hilos, los tiempos reales promedio mejoraron en un 18.65 % en promedio con respecto a los de la implementación sin hilos.

En cambio, para la Búsqueda Binaria, utilizando dos hilos, los tiempos reales promedio empeoraron en un 94.64 % en promedio con respecto a los de la implementación sin hilos.

11. ¿Existió un entorno controlado para realizar las pruebas experimentales?¿Cuál fue? .
Si, se realizaron las búsquedas con distintos tamaños de problema (n's) según el algoritmo, todos al mismo tiempo. Todos los algoritmos fueron probados en la misma máquina al momento de realizar las gráficas comparativas. Solo para el caso del ABB se ingresaron 10 millones de números en desorden, para los otros dos algoritmos estaban ordenados.

12. ¿Si solo se realizara el análisis teórico de un algoritmo antes de implementarlo, podrías asegurar cuál es el mejor?
Sí, pues a pesar de que no todas las veces se obtendrán las mismas funciones de complejidad temporal exactas, éstas siempre estarán contenidas dentro de la cota de complejidad correspondiente al algoritmo.

13. ¿Qué tan difícil fue realizar el análisis teórico de cada algoritmo?

    ✓ **Búsqueda Lineal:**
    Relativamente sencillo, pues es la solución a la que llamamos Bruta.

    ✓ **Búsqueda Binaria:**
    A simple vista pareciera que el orden de complejidad de este algoritmo es lineal, pero hay que prestar mucha atención a la operación que se realiza en la variable *centro*, pues va dividiendo el contador de dos en dos.

    ✓ **ABB:**
    En este algoritmo, el mayor problema fue el ver los diferentes casos que podrían pasar, tenía la posibilidad de ser un árbol balanceado o no balanceado, entonces algunas cosas cambian totalmente.

14. ¿Qué recomendaciones darían a nuevos equipos para realizar esta práctica?

    - **Comprobar los algoritmos con n pequeñas con el objetivo de verificar que el algoritmo esta funcionando de forma correcta, si funciona con n pequeñas también lo hará con las mayores.**
    - **Estudiar y comprender el funcionamiento de hilos en el lenguaje C.**
    - **Probar los hilos en C con programas y ejemplos sencillos, que reflejen el comportamiento real de su ejecución.**
    - **Utilizar algún software matemático como Matlab o Mathematica (Wolfram Alpha) para calcular los polinomios.**
    - **Hacer uso de scripts que faciliten la compilación y ejecución de los algoritmos para muchos tamaños de problema.**
    - **No usar Windows.**

4. Errores detectados

✓ Al compilar los algoritmos de búsqueda Lineal y Binaria en su implementación con hilos aparecen los siguientes warnings en la terminal:

```
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$ gcc tiempo.c -c
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$ gcc -lm LinealHilos.c tiempo.o -lpthread -o LinealHilos
LinealHilos.c: In function ‘LinealHilos’:
LinealHilos.c:44:15: warning: cast from pointer to integer of different size [-W
pointer-to-int-cast]
  int n_thread=(int)id, inicio, fin;
               ^
LinealHilos.c: In function ‘main’:
LinealHilos.c:105:53: warning: cast to pointer from integer of different size [-
Wint-to-pointer-cast]
    if (pthread_create(&thread[i], NULL, LinealHilos,(void*)i) != 0 )
                                                     ^
```

Figura 1: Warning al compilar la Búsqueda Lineal con hilos

```
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$ gcc tiempo.c -c
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$ gcc -lm BinariaHilos.c tiempo.o -lpthread -o BinariaHilos
BinariaHilos.c: In function ‘BinariaHilos’:
BinariaHilos.c:44:15: warning: cast from pointer to integer of different size [-
Wpointer-to-int-cast]
  int n_thread=(int)id, inicio, fin;
               ^
BinariaHilos.c: In function ‘main’:
BinariaHilos.c:112:54: warning: cast to pointer from integer of different size [
-Wint-to-pointer-cast]
    if (pthread_create(&thread[i], NULL, BinariaHilos,(void*)i) != 0 )
                                                      ^
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de Algoritmos/Practicas/P
actica2/Codigo$
```

Figura 2: Warning al compilar la Búsqueda Binaria con hilos

Sin embargo, podemos ignorar estas advertencias pues los ejecutables son generados de manera correcta.

✓ En la implementación de la Búsqueda Binaria con hilos, los tiempos de ejecución son mucho mayores a los de la implementación sin hilos.

5. Anexos

5.1. Búsqueda Lineal

```c
//*****************************************************************
//AUTORES:
// Nicolas Sayago Abigail
// Parra Garcilazo Cinthya Dolores
// Ramos Diaz Enrique
//
//Practica 2: Análisis temporal y notación de orden (Algoritmos de búsqueda)
//Compilación:
//   gcc tiempo.c -c
//   gcc Lineal.c tiempo.o -o Lineal
//
//Ejecución: "./Lineal 10000000 <ordenados.txt" (Linux)
//*****************************************************************

//*****************************************************************
//Librerias incluidas
//*****************************************************************
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "tiempo.h"

//*****************************************************************
//Busqueda Lineal
//*****************************************************************
//Descripción: Función que implementa el algoritmo de busqueda Lineal
//Recibe: Un arreglo de enteros, el tamaño del arreglo y un entero a buscar
//Devuelve: -1 si no encontro el dato, o la posicion en el arreglo del dato
//*****************************************************************
int Lineal(int A[], int n, int dato)
{
   int posicion = 0;
   while(posicion < n)
   {
      if(dato == A[posicion])
         return posicion;
      posicion++;
   }
   return -1;
}

int main(int argc, char *argv[])
{
   //Obtenemos n como parametro del main y creamos una arreglo dinamico
   int n = atoi(argv[1]), i = 0, j = 0;
   float suma = 0, promedio = 0;
   int *arreglo = (int*)calloc(n,sizeof(int));
   int datos[20] = {322486, 14700764, 3128036, 6337399, 61396,
   10393545, 2147445644, 1295390003, 450057883, 187645041,
   1980098116, 152503, 5000, 1493283650, 214826, 1843349527,
   1360839354, 2109248666 , 2147470852, 0};

   //Agregamos los n valores del txt al arreglo
   for(i = 0; i < n; i++)
      fscanf(stdin, "%d", &arreglo[i]);
   printf("\nBusqueda Lineal n = %d", n);
   //Buscamos los valores solicitados en la lista de 10 millones
   for(j = 0; j < 20; j++)
   {
      double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para medición de tiempos
      uswtime(&utime0, &stime0, &wtime0);

      int s = Lineal(arreglo, n, datos[j]);
```

```
64
65          uswtime(&utime1, &stime1, &wtime1);
66
67          if(j == 17)// para 2109248666
68          {
69             if(s != -1)
70                printf("\n\n%d SI : %d ", datos[j], s);
71             else
72                printf("\n\n%d NO : --- ", datos[j]);
73
74             //Cálculo del tiempo de ejecución del programa
75             printf("\n");
76             printf("Total %.15e \n",  wtime1 - wtime0); //Tiempo Real
77             printf("CPU %.15e \n",  utime1 - utime0); //Tiempo CPU
78             printf("E/S %.15e \n",  stime1 - stime0); //Tiempo E/S
79             printf("CPU/Wall %.8f %% ",100.0 * (utime1 - utime0 + stime1 - stime0) / (wtime1 -
                 ↪  wtime0)); //CPU Wall
80             printf("\n");
81          }
82          suma = suma + wtime1 - wtime0;
83       }
84       printf("\nPromedio Tiempo Total: %.20f s\n\n", suma/20);
85
86       printf("-------------------------------------\n");
87    }
```

5.2. Búsqueda Lineal (Hilos)

```
1  //*****************************************************************
2  //AUTORES:
3  // Nicolas Sayago Abigail
4  // Parra Garcilazo Cinthya Dolores
5  // Ramos Diaz Enrique
6  //
7  //Practica 2: Análisis temporal y notación de orden (Algoritmos de búsqueda)
8  //Compilación:
9  //   gcc tiempo.c -c
10 //   gcc -lm LinealHilos.c tiempo.o -lpthread -o LinealHilos
11 //Ejecución: "./LinealHilos 4 10000000 <ordenados.txt"
12 //*****************************************************************
13
14 //*****************************************************************
15 //LIBRERIAS INCLUIDAS
16 //*****************************************************************
17 #include <pthread.h>
18 #include <stdio.h>
19 #include <stdlib.h>
20 #include "tiempo.h"
21
22 //*****************************************************************
23 //VARIABLES GLOBALES
24 //*****************************************************************
25 int NumThreads, N, indice = 0, encontrado = -1, s = 0;
26 //Variables del tamaño de problema, numeros de hilos, y auxiliares para
27 //recorrer nuestro arreglo de numeros a buscar
28 int *arreglo;//Arreglo de los numeros de entrada del TXT
29 int datos[20] = {322486, 14700764, 3128036, 6337399, 61396,
30 10393545, 2147445644, 1295390003, 450057883, 187645041,
31 1980098116, 152503, 5000, 1493283650, 214826, 1843349527,
32 1360839354, 2109248666 , 2147470852, 0};//Arreglo de numeros a buscar
33
34 //*****************************************************************
35 //LinealHilos
36 //*****************************************************************
37 //Descripción: Hilo que procesa el algoritmo de busqueda lineal e
38 //indica si el dato a buscar está o no en el arreglo.
```

```
39  //Recibe: Un indice de tipo void que indica el numero de hilo
40  //Devuelve: Nada
41  //*****************************************************************
42  void* LinealHilos(void* id)
43  {
44     int n_thread=(int)id, inicio, fin;
45
46     //Revisar la parte de los datos a procesar
47     inicio=(n_thread*N)/NumThreads;
48     if(n_thread==NumThreads-1)
49        fin=N-1;
50     else
51        fin=((n_thread+1)*N)/NumThreads-1;
52
53     //*****************************************************************
54     //Implementacion del algoritmo Busqueda lineal
55     //*****************************************************************
56     int posicion = inicio;
57     if(datos[indice] > arreglo[inicio])
58        while(posicion <= fin)
59        {
60           if(datos[indice] == arreglo[posicion])
61              encontrado = 1;
62           posicion++;
63        }
64  }
65
66  //*****************************************************************
67  //PROGRAMA PRINCIPAL
68  //*****************************************************************
69  int main (int argc, char *argv[])
70  {
71     int i, k;     //Variables auxiliares para loops
72     float suma = 0, promedio = 0;
73     NumThreads = atoi(argv[1]);
74     N = atoi(argv[2]);
75     arreglo = (int*)calloc(N,sizeof(int));
76
77     //Con este for vamos agregando los n valores del txt al arreglo
78     for(k = 0; k < N; k++)
79        fscanf(stdin, "%d", &arreglo[k]);
80     printf("\nBusqueda Lineal Hilos n = %d\n\n", N);
81
82     //Con este for vamos buscando cada numero en el arreglo
83     for(indice = 0; indice < 20; indice++)
84     {
85        //*****************************************************************
86        //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
87        //*****************************************************************
88        double utime0, stime0, wtime0,utime1, stime1, wtime1;
89        uswtime(&utime0, &stime0, &wtime0);
90        encontrado = -1;
91        s = 0;
92
93        //*************************************************************************
94        //Obtener el número de threads a usar y el arreglo para la identificacion de los mismos
95        //*************************************************************************
96        pthread_t *thread;
97        thread = calloc(NumThreads,sizeof(pthread_t));
98
99        //*************************************************************************
100       //Procesar desde cada hilo "LinealHilos"
101       //*************************************************************************
102       //Crear los threads con el comportamiento "segmentar"
103       for (i = 1; i < NumThreads; i++)
104       {
105          if (pthread_create(&thread[i], NULL, LinealHilos,(void*)i) != 0 )
106          {
107             perror("El hilo no pudo crearse");
```

```
108             exit(-1);
109          }
110       }
111
112       //El main ejecuta el hilo 0
113       LinealHilos(0);
114
115       //Esperar a que terminen los hilos
116       for (i = 1; i < NumThreads; i++)
117          pthread_join (thread[i], NULL);
118
119       free(thread);
120       //*****************************************************************
121       //Evaluar los tiempos de ejecución
122       //*****************************************************************
123       uswtime(&utime1, &stime1, &wtime1);
124
125       //Cálculo del tiempo de ejecución del programa
126       if(indice == 17)// para 2109248666
127       {
128          if(encontrado != -1)
129             printf("\n%d SI : %d ", datos[indice], s);
130          else
131             printf("\n%d NO : --- ", datos[indice]);
132
133          printf("\n");
134          printf("Total  %.15e\n",  wtime1 - wtime0);
135          printf("CPU's %.15e\n",  utime1 - utime0);
136          printf("Hilos %.15e\n", (utime1 - utime0)/NumThreads);
137          printf("E/S %.15e\n",  stime1 - stime0);
138          printf("CPU/Wall %.8f %%\n",100.0 * (utime1 - utime0 + stime1 - stime0) / (wtime1 -
               ↪  wtime0));
139          printf("\n");
140       }
141       suma = suma + wtime1 - wtime0;
142
143       //*****************************************************************
144    }
145    printf("\nPromedio Tiempo Total: %.20f s\n\n", suma/20);
146
147    printf("------------------------------------\n");
148
149    //Terminar programa normalmente
150    exit (0);
151 }
```

5.3. Búsqueda Binaria

```
1  //*****************************************************************
2  //AUTORES:
3  // Nicolas Sayago Abigail
4  //   Parra Garcilazo Cinthya Dolores
5  // Ramos Diaz Enrique
6  //
7  //Practica 2: Análisis temporal y notación de orden (Algoritmos de búsqueda)
8  //Compilación:
9  //   gcc tiempo.c -c
10 //   gcc Binaria.c tiempo.o -o Binaria
11 //
12 //Ejecución: "./Binaria 10000000 <ordenados.txt" (Linux)
13 //*****************************************************************
14
15 //*****************************************************************
16 //Librerias incluidas
17 //*****************************************************************
18 #include <stdio.h>
```

```c
#include <stdlib.h>
#include <string.h>
#include "tiempo.h"

//*****************************************************************
//Binaria
//*****************************************************************
//Descripción: Función que implementa el algoritmo de busqueda Binaria
//Recibe: Un arreglo de enteros, el tamaño del arreglo y un entero a buscar
//Devuelve: -1 si no encontro el dato, o la posicion en el arreglo del dato
//*****************************************************************
int Binaria(int A[], int n, int dato)
{
   //centro: subíndice central del intervalo
     //inf: límite inferior del intervalo
     //sup: límite superior del intervalo
   int centro, inf = 0, sup = n-1;
   while(inf <= sup)
   {
      centro = ((sup + inf)/2);
      if(A[centro] == dato)
      {
         /*Para imprimir la posicion y el dato dentro del arreglo
         printf("SI %d : %d", A[centro], centro);*/
         return centro;
      }
      else if (dato < A[centro])
         sup = centro - 1;
      else
         inf = centro + 1;
   }
   return -1;
}

int main(int argc, char *argv[])
{
   //Obtenemos n como parametro del main y creamos una arreglo dinamico
   int n = atoi(argv[1]), i = 0, j = 0;
   float suma = 0, promedio = 0;
   int *arreglo = (int*)calloc(n,sizeof(int));
   int datos[20] = {322486, 14700764, 3128036, 6337399, 61396,
   10393545, 2147445644, 1295390003, 450057883, 187645041,
   1980098116, 152503, 5000, 1493283650, 214826, 1843349527,
   1360839354, 2109248666 , 2147470852, 0};

   //Con este for vamos agregando los n valores del txt al arreglo
   for(i = 0; i < n; i++)
      fscanf(stdin, "%d", &arreglo[i]);
   printf("\nBusqueda Binaria n = %d", n);
   //Con este for vamos buscando cada numero en el arreglo
   for(j = 0; j < 20; j++)
   {
      double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para medición de tiempos
      uswtime(&utime0, &stime0, &wtime0);

      int s = Binaria(arreglo, n, datos[j]);

      uswtime(&utime1, &stime1, &wtime1);

      if(j == 17)// para 2109248666
      {
         if(s != -1)
            printf("\n\n%d SI : %d ", datos[j], s);
         else
            printf("\n\n%d NO : --- ", datos[j]);

         //Cálculo del tiempo de ejecución del programa
         printf("\n");
         printf("Total %.15e \n",  wtime1 - wtime0); //Tiempo Real
```

```
88          printf("CPU %.15e \n",  utime1 - utime0); //Tiempo CPU
89          printf("E/S %.15e \n",  stime1 - stime0); //Tiempo E/S
90          printf("CPU/Wall %.8f %% ",100.0 * (utime1 - utime0 + stime1 - stime0) / (wtime1 -
              ↪  wtime0)); //CPU Wall
91          printf("\n");
92       }
93       suma = suma + wtime1 - wtime0;
94    }
95    printf("\nPromedio Tiempo Total: %.20f s\n\n", suma/20);
96
97    printf("-----------------------------------\n");
98    exit(0);
99 }
```

5.4. Búsqueda Binaria (Hilos)

```
1  //*****************************************************************
2  //AUTORES:
3  // Nicolas Sayago Abigail
4  //   Parra Garcilazo Cinthya Dolores
5  // Ramos Diaz Enrique
6  //
7  //Practica 2: Análisis temporal y notación de orden (Algoritmos de búsqueda)
8  //Compilación:
9  //   gcc tiempo.c -c
10 //   gcc -lm BinariaHilos.c tiempo.o -lpthread -o BinariaHilos
11 //Ejecución: "./BinariaHilos 4 10000000 <ordenados.txt"
12 //*****************************************************************
13
14 //*****************************************************************
15 //LIBRERIAS INCLUIDAS
16 //*****************************************************************
17 #include <pthread.h>
18 #include <stdio.h>
19 #include <stdlib.h>
20 #include "tiempo.h"
21
22 //*****************************************************************
23 //VARIABLES GLOBALES
24 //*****************************************************************
25 int NumThreads, N, indice = 0, encontrado = -1, s = 0;
26 //Variables del tamaño de problema, numeros de hilos, y auxiliares para
27 //recorrer nuestro arreglo de numeros a buscar
28 int *arreglo;//Arreglo de los numeros de entrada del TXT
29 int datos[20] = {322486, 14700764, 3128036, 6337399, 61396,
30 10393545, 2147445644, 1295390003, 450057883, 187645041,
31 1980098116, 152503, 5000, 1493283650, 214826, 1843349527,
32 1360839354, 2109248666 , 2147470852, 0};//Arreglo de numeros a buscar
33
34 //*****************************************************************
35 //BinariaHilos
36 //*****************************************************************
37 //Descripción: Hilo que procesa el algoritmo de busqueda binaria
38 //Recibe: Un indice de tipo void que indica el numero de hilo
39 //Devuelve: Nada, pero indica si el dato a buscar esta o no en el
40 //arreglo de numeros ordenados
41 //*****************************************************************
42 void* BinariaHilos(void* id)
43 {
44    int n_thread=(int)id, inicio, fin;
45
46    //Revisar la parte de los datos a procesar
47    inicio=(n_thread*N)/NumThreads;
48    if(n_thread==NumThreads-1)
49       fin=N-1;
50    else
```

```
51          fin=((n_thread+1)*N)/NumThreads-1;
52
53       //*************************************************************
54       //Implementacion del algoritmo Busqueda binaria
55       //*************************************************************
56       int centro, inf = inicio, sup = fin;
57       while(inf <= sup)
58       {
59          centro = ((sup + inf)/2);
60          if(arreglo[centro] == datos[indice])
61          {
62             encontrado = 1;
63             s = centro;
64             break;
65          }
66          else if (datos[indice] < arreglo[centro])
67             sup = centro - 1;
68          else
69             inf = centro + 1;
70       }
71    }
72
73    //*************************************************************
74    //PROGRAMA PRINCIPAL
75    //*************************************************************
76    int main (int argc, char *argv[])
77    {
78       int i, k;     //Variables auxiliares para loops
79       float suma = 0, promedio = 0;
80       NumThreads = atoi(argv[1]);
81       N = atoi(argv[2]);
82       arreglo = (int*)calloc(N,sizeof(int));
83
84       //Con este for vamos agregando los n valores del txt al arreglo
85       for(k = 0; k < N; k++)
86          fscanf(stdin, "%d", &arreglo[k]);
87       printf("\nBusqueda Binaria Hilos n = %d\n\n", N);
88
89       //Con este for vamos buscando cada numero en el arreglo
90       for(indice = 0; indice < 20; indice++)
91       {
92          //*****************************************************************
93          //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
94          //*****************************************************************
95          double utime0, stime0, wtime0,utime1, stime1, wtime1;
96          uswtime(&utime0, &stime0, &wtime0);
97          encontrado = -1;
98          s = 0;
99
100         //*************************************************************************
101         //Obtener el número de threads a usar y el arreglo para la identificacion de los mismos
102         //*************************************************************************
103         pthread_t *thread;
104         thread = calloc(NumThreads,sizeof(pthread_t));
105
106         //*************************************************************************
107         //Procesar desde cada hilo "BinariaHilos"
108         //*************************************************************************
109         //Crear los threads con el comportamiento "segmentar"
110         for (i = 1; i < NumThreads; i++)
111         {
112            if (pthread_create(&thread[i], NULL, BinariaHilos,(void*)i) != 0 )
113            {
114               perror("El hilo no pudo crearse");
115               exit(-1);
116            }
117         }
118
119         //El main ejecuta el hilo 0
```

```
120         BinariaHilos(0);
121
122         //Esperar a que terminen los hilos
123         for (i = 1; i < NumThreads; i++)
124            pthread_join (thread[i], NULL);
125
126         free(thread);
127         //*****************************************************************
128         //Evaluar los tiempos de ejecución
129         //*****************************************************************
130         uswtime(&utime1, &stime1, &wtime1);
131
132         //Cálculo del tiempo de ejecución del programa
133         if(indice == 17)// para 2109248666
134         {
135            if(encontrado != -1)
136               printf("\n%d SI : %d ", datos[indice], s);
137            else
138               printf("\n%d NO : --- ", datos[indice]);
139
140            printf("\n");
141            printf("Total  %.15e\n",  wtime1 - wtime0);
142            printf("CPU's %.15e\n",  utime1 - utime0);
143            printf("Hilo %.15e\n", (utime1 - utime0)/NumThreads);
144            printf("E/S %.15e\n",  stime1 - stime0);
145            printf("CPU/Wall %.8f %%\n",100.0 * (utime1 - utime0 + stime1 - stime0) / (wtime1 -
                 ↪  wtime0));
146            printf("\n");
147         }
148         suma = suma + wtime1 - wtime0;
149
150         //*****************************************************************
151      }
152      printf("\nPromedio Tiempo Total: %.20f s\n\n", suma/20);
153
154      printf("-----------------------------------\n");
155
156      //Terminar programa normalmente
157      exit (0);
158   }
```

5.5. Arbin.h

```
1   //*****************************************************************
2   //   AUTORES:
3   //    Nicolas Sayago Abigail
4   //   Parra Garcilazo Cinthya Dolores
5   //    Ramos Diaz Enrique
6   // ****************************************************************
7   //   Practica 2: Análisis temporal y notación de orden (Algoritmos de búsqueda)
8   //   ****************************************************************
9
10  //   ****************************************************************
11  //         Librerias
12  //   ****************************************************************
13
14  #include "stdlib.h"
15
16  //Defimos la estructura de un arbol binario
17  //Consiste en una raiz, un nodo izquierdo y un bodo derecho
18  typedef struct NodoA
19  {
20     struct NodoA *izq;
21     struct NodoA *der;
22     int raiz;
23  } NodoA;
24
```

```
25 typedef NodoA *posicion;   // La posición será la dirección hacia un NodoA específico
26 typedef posicion Arbin; // El árbol binario será simplemente una posición de un NodoA
27
28 //   ************************************************************
29 //                   consA
30 //   ************************************************************
31 //   Descripción: Construye un arbol binario
32 //   Recibe: Un apuntador al arbol binario
33 //   Devuelve:
34 //   ************************************************************
35 void consA(Arbin *a)
36 {
37    *a = NULL; // El apuntador enviado por el usuario se coloca en un valor NULL
38 }
39
40 //   ************************************************************
41 //                   destruir
42 //   ************************************************************
43 //   Descripción: Elimina de la memoria un arbol binario
44 //   Recibe: Un apuntador al arbol binario
45 //   Devuelve:
46 //   ************************************************************
47 void destruir(Arbin *a)
48 {
49    if (*a != NULL) // Veirficamos no estar apuntando a un valor nulo en el árbol enviado
50    {
51       if ((*a) -> izq != NULL)      // Verificamos si el árbol izquierdo existe, para eliminarlo
         ↪  primero
52          destruir(&((*a) -> izq));   // Llamamos recursivamente la función por el lado izquierdo
53       if ((*a) -> der != NULL)      // Posteriormente eliminamos el lado derecho del árbol
         ↪  verificando que existe
54          destruir(&((*a) -> der)); // Llamamos recursivamente la función por el lado derecho
55       free(*a);                           // Liberamos el nodo una vez que ya no tiene hijos
56    }
57 }
```

5.6. Árbol de Búsqueda Binaria

```
1  //************************************************************
2  //   AUTORES:
3  //    Nicolas Sayago Abigail
4  //   Parra Garcilazo Cinthya Dolores
5  //    Ramos Diaz Enrique
6  // ***********************************************************
7  //   Practica 2: Análisis temporal y notación de orden (Algoritmos de búsqueda)
8  //   Compilación:
9  //   gcc tiempo.c -c
10 //   gcc ABB.c tiempo.o -o ABB
11 //
12 //   Ejecución: "./ABB 10000000 <desordenados.txt" (Linux)
13 //   ************************************************************
14
15 //   ************************************************************
16 //                   Librerias incluidas
17 //   ************************************************************
18 #include <stdio.h>
19 #include <stdlib.h>
20 #include "Arbin.h"
21 #include "tiempo.h"
22
23 //   ************************************************************
24 //                   Insertar
25 //   ************************************************************
26 //   Descripción: Inserta un arreglo de números en un ABB
27 //   Recibe: Un ABB vacio y un entero
28 //   Devuelve: Nada, pero construye el arbol binario
```

```
29 //   ************************************************************
30 void Insertar(Arbin *a, int e)
31 {
32    Arbin *apu_a = a; // Declaramos un apuntador para recorrer el árbol
33    while (*apu_a != NULL)
34    {
35       if (e > ((*apu_a) -> raiz))
36          apu_a = &((*apu_a) -> der);
37       else
38          apu_a = &((*apu_a) -> izq);
39    }
40    *apu_a = (NodoA *)malloc(sizeof(NodoA));
41    (*apu_a) -> raiz = e;          // En el nodo colocaremos el elemento a introducir en el árbol
42    (*apu_a) -> izq = NULL;       // Nos aseguramos de que ambos hijos estén apuntando a un valor NULL
43    (*apu_a) -> der = NULL;
44 }
45
46 //   ************************************************************
47 //                   ABB
48 //   ************************************************************
49 //   Descripción: Busca si existe un elemento en el arbol
50 //   Recibe: Un arbol binario y el elemento a buscar
51 //   Devuelve: 0 si existe el elemento, -1 si no existe
52 //   ************************************************************
53
54 int ABB(Arbin *a, int elemento)
55 {
56    // Declaramos un apuntador auxiliar para viajar por el árbol
57    posicion a_aux = *a;
58    int numero; // Auxiliar para comparar
59    do
60    {
61       numero = a_aux -> raiz;
62       // Ese nodo dira cual es el numero en ese momento, posteriormente moveremos el índice un lugar
         ↪  más
63       //printf("Compara: Arbol-%d , elemento-%d \n", numero, elemento);
64       if(numero == elemento)
65       {
66          a_aux = NULL;
67          return 0;
68       }
69       else if(numero < elemento)
70       {
71          // Si el elemento es mayor a la raiz, vamos al lado derecho del subarbol
72          a_aux = a_aux -> der;
73       }
74       else
75       {
76          // Si el elemento es menor a la raiz, vamos al lado izquierdo del subarbol
77          a_aux = a_aux -> izq;
78       }
79    }
80    while (a_aux != NULL); // Apuntador nulo
81
82    return -1;
83 }
84
85 int main(int argc, char *argv[])
86 {
87    //Obtenemos n como parametro del main y creamos una arreglo dinamico
88    int n = atoi(argv[1]), i = 0, j = 0, k = 0;
89    int *arreglo = (int*)calloc(n,sizeof(int));
90    float suma = 0, promedio = 0;
91    // DECLARO UN ARREGLO PARA LOS DATOS QUE VAMOS A BUSCAR EN EL ARBOL
92    int datos[20] = {322486, 14700764, 3128036, 6337399, 61396,
93    10393545, 2147445644, 1295390003, 450057883, 187645041,
94    1980098116, 152503, 5000, 1493283650, 214826, 1843349527,
95    1360839354, 2109248666 , 2147470852, 0};
96
```

```
97    printf("Arbol de Busqueda Binaria n = %d\n", n);
98    //Con este for vamos agregando los n valores del txt al arreglo
99    for(k = 0; k < n; k++)
100   {
101      fscanf(stdin, "%d", &arreglo[k]);
102   }
103
104   /*AGREGA AL ARBOL LOS NUMEROS DEL TXT*/
105   Arbin ArbolBinBusqueda;
106   consA(&ArbolBinBusqueda); //Asignamos el valor NULL al apuntador del ABB
107
108   for(i = 0; i < n; i++)
109   {
110      Insertar(&ArbolBinBusqueda, arreglo[i]);
111   }
112
113   /*
114      BUSCA EN EL ARBOL CADA NUMERO DEL ARREGLO DATOS
115   */
116   for(j = 0; j < 20; j++)
117   {
118      double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para medición de tiempos
119      uswtime(&utime0, &stime0, &wtime0);
120
121      int s = ABB(&ArbolBinBusqueda, datos[j]);
122      // La funcion recibe el arbol, numero de datos y el dato
123
124      uswtime(&utime1, &stime1, &wtime1);
125
126      if(j == 17)// para 2109248666
127      {
128         if(s != -1)
129            printf("\n\n%d SI : %d ", datos[j], s);
130         else
131            printf("\n\n%d NO : --- ", datos[j]);
132
133         //Cálculo del tiempo de ejecución del programa
134         printf("\n");
135         printf("Total %.15e \n",  wtime1 - wtime0); //Tiempo Real
136         printf("CPU %.15e \n",  utime1 - utime0); //Tiempo CPU
137         printf("E/S %.15e \n",  stime1 - stime0); //Tiempo E/S
138         printf("CPU/Wall %.8f %% ",100.0 * (utime1 - utime0 + stime1 - stime0) / (wtime1 -
            ↪  wtime0)); //CPU Wall
139         printf("\n");
140      }
141      suma = suma + wtime1 - wtime0;
142   }
143
144   destruir(&ArbolBinBusqueda);
145
146   printf("\nPromedio Tiempo Total: %.20f s\n\n", suma/20);
147
148   printf("-------------------------------------\n");
149
150   return 0;
151 }
```

5.7. Script de Compilación

```
1  gcc tiempo.c -c
2
3  gcc Lineal.c tiempo.o -o Lineal
4  ./Lineal 100 <ordenados.txt >>lineal.txt
5  ./Lineal 1000 <ordenados.txt >>lineal.txt
6  ./Lineal 5000 <ordenados.txt >>lineal.txt
7  ./Lineal 10000 <ordenados.txt >>lineal.txt
```

```
./Lineal 50000 <ordenados.txt >>lineal.txt
./Lineal 100000 <ordenados.txt >>lineal.txt
./Lineal 200000 <ordenados.txt >>lineal.txt
./Lineal 400000 <ordenados.txt >>lineal.txt
./Lineal 600000 <ordenados.txt >>lineal.txt
./Lineal 800000 <ordenados.txt >>lineal.txt
./Lineal 1000000 <ordenados.txt >>lineal.txt
./Lineal 2000000 <ordenados.txt >>lineal.txt
./Lineal 3000000 <ordenados.txt >>lineal.txt
./Lineal 4000000 <ordenados.txt >>lineal.txt
./Lineal 5000000 <ordenados.txt >>lineal.txt
./Lineal 6000000 <ordenados.txt >>lineal.txt
./Lineal 7000000 <ordenados.txt >>lineal.txt
./Lineal 8000000 <ordenados.txt >>lineal.txt
./Lineal 9000000 <ordenados.txt >>lineal.txt
./Lineal 10000000 <ordenados.txt >>lineal.txt

./LinealHilos 2 100 <ordenados.txt >>lineal.txt
./LinealHilos 2 1000 <ordenados.txt >>lineal.txt
./LinealHilos 2 5000 <ordenados.txt >>lineal.txt
./LinealHilos 2 10000 <ordenados.txt >>lineal.txt
./LinealHilos 2 50000 <ordenados.txt >>lineal.txt
./LinealHilos 2 100000 <ordenados.txt >>lineal.txt
./LinealHilos 2 200000 <ordenados.txt >>lineal.txt
./LinealHilos 2 400000 <ordenados.txt >>lineal.txt
./LinealHilos 2 600000 <ordenados.txt >>lineal.txt
./LinealHilos 2 800000 <ordenados.txt >>lineal.txt
./LinealHilos 2 1000000 <ordenados.txt >>lineal.txt
./LinealHilos 2 2000000 <ordenados.txt >>lineal.txt
./LinealHilos 2 3000000 <ordenados.txt >>lineal.txt
./LinealHilos 2 4000000 <ordenados.txt >>lineal.txt
./LinealHilos 2 5000000 <ordenados.txt >>lineal.txt
./LinealHilos 2 6000000 <ordenados.txt >>lineal.txt
./LinealHilos 2 7000000 <ordenados.txt >>lineal.txt
./LinealHilos 2 8000000 <ordenados.txt >>lineal.txt
./LinealHilos 2 9000000 <ordenados.txt >>lineal.txt
./LinealHilos 2 10000000 <ordenados.txt >>lineal.txt

gcc Binaria.c tiempo.o -o Binaria
./Binaria 100 <ordenados.txt >>Binaria.txt
./Binaria 1000 <ordenados.txt >>Binaria.txt
./Binaria 5000 <ordenados.txt >>Binaria.txt
./Binaria 10000 <ordenados.txt >>Binaria.txt
./Binaria 50000 <ordenados.txt >>Binaria.txt
./Binaria 100000 <ordenados.txt >>Binaria.txt
./Binaria 200000 <ordenados.txt >>Binaria.txt
./Binaria 400000 <ordenados.txt >>Binaria.txt
./Binaria 600000 <ordenados.txt >>Binaria.txt
./Binaria 800000 <ordenados.txt >>Binaria.txt
./Binaria 1000000 <ordenados.txt >>Binaria.txt
./Binaria 2000000 <ordenados.txt >>Binaria.txt
./Binaria 3000000 <ordenados.txt >>Binaria.txt
./Binaria 4000000 <ordenados.txt >>Binaria.txt
./Binaria 5000000 <ordenados.txt >>Binaria.txt
./Binaria 6000000 <ordenados.txt >>Binaria.txt
./Binaria 7000000 <ordenados.txt >>Binaria.txt
./Binaria 8000000 <ordenados.txt >>Binaria.txt
./Binaria 9000000 <ordenados.txt >>Binaria.txt
./Binaria 10000000 <ordenados.txt >>Binaria.txt

./BinariaHilos 2 100 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 1000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 5000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 10000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 50000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 100000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 200000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 400000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 600000 <ordenados.txt >>Binaria.txt
```

```
./BinariaHilos 2 800000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 1000000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 2000000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 3000000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 4000000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 5000000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 6000000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 7000000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 8000000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 9000000 <ordenados.txt >>Binaria.txt
./BinariaHilos 2 10000000 <ordenados.txt >>Binaria.txt

gcc ABB.c tiempo.o -o ABB
./ABB 100 <desordenados.txt >>ABB.txt
./ABB 1000 <desordenados.txt >>ABB.txt
./ABB 5000 <desordenados.txt >>ABB.txt
./ABB 10000 <desordenados.txt >>ABB.txt
./ABB 50000 <desordenados.txt >>ABB.txt
./ABB 100000 <desordenados.txt >>ABB.txt
./ABB 200000 <desordenados.txt >>ABB.txt
./ABB 400000 <desordenados.txt >>ABB.txt
./ABB 600000 <desordenados.txt >>ABB.txt
./ABB 800000 <desordenados.txt >>ABB.txt
./ABB 1000000 <desordenados.txt >>ABB.txt
./ABB 2000000 <desordenados.txt >>ABB.txt
./ABB 3000000 <desordenados.txt >>ABB.txt
./ABB 4000000 <desordenados.txt >>ABB.txt
./ABB 5000000 <desordenados.txt >>ABB.txt
./ABB 6000000 <desordenados.txt >>ABB.txt
./ABB 7000000 <desordenados.txt >>ABB.txt
./ABB 8000000 <desordenados.txt >>ABB.txt
./ABB 9000000 <desordenados.txt >>ABB.txt
./ABB 10000000 <desordenados.txt >>ABB.txt
```

6. Bibliografía

[1] E. A. Franco Martínez, “Análisis temporal y notación de orden (Algoritmos de búsqueda)” Análisis de Algoritmos, Escuela Superior de Computación, Instituto Politécnico Nacional, Ciudad de México, México, Practica02.pdf, Sep. 2018

[2] (2018) WolframAlpha. Accessed september 2018. [Online].
Available: http://www.wolframalpha.com/