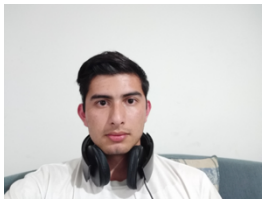
ANALISIS DE ALGORITMOS

Equipo:

Jaime Alejandro Salinas Núñez

Martín Eduardo Barriga Vargas

José Manuel Ramírez Vives



3CM3

Reporte practica 2

"Análisis temporal y notación de orden"

(Algoritmos de búsqueda)

4 de abril de 2019

PLANTEAMIENTO DEL PROBLEMA

La práctica consistió en llevar a cabo la implementación de tres algoritmos de búsqueda para poder realizar una comparación y un análisis de su complejidad temporal, esto midiendo el tiempo que tarda cada algoritmo en encontrar un determinado número en una determinada cantidad de estos.

De igual forma se implementó una variante de cada algoritmo la cual utiliza hilos, para “mejorar” el rendimiento de cada uno de estos, permitiendo realizar un análisis de cómo es que funcionan los hilos en determinados problemas.

También se realizó un análisis teórico con el cual podemos observar que tanto difiere un análisis a otro.

Se implementaron 3 algoritmos de búsqueda para llevar a cabo las pruebas y análisis:

1. Búsqueda lineal o secuencial.
2. Búsqueda binaria o dicotómica.
3. Búsqueda en un árbol binario de búsqueda.

Para cada uno de estos algoritmos, se implementó una variante con hilos, la cual también será analizada.

Se cuenta con un archivo el cual contiene 10 millones de números diferentes, este archivo funciona como la entrada para cada algoritmo, también, se introduce el número a buscar dentro de todo el conjunto de números.
Cabe resaltar que para el algoritmo de búsqueda binaria, ocupa como entrada, que la lista de los 10 millones de números esté ordenada ascendentemente, por lo que el archivo de entrada de este algoritmo, será el archivo de salida de la Práctica01.

Para cada algoritmo se usarán diferentes cantidades de números (parámetro n) y diferentes números, los cuales tendrán que ser buscados por él algoritmo con base en el conjunto de números n.

El tiempo que cada algoritmo tarde en encontrar el número a buscar será medido y reflejado en este reporte.

Al igual basándonos en la medición de los tiempos se realizarán aproximaciones a las formas que dan los algoritmos en tiempo mediante polinomios algebraicos para entender mejor el comportamiento que los algoritmos arrojan.

Se obtendrá una función temporal y una cota de manera teórica que nos servirá de referencia y comparación de lo real a lo teórico.

ACTIVIDADES Y PRUEBAS

I. Análisis teórico de los algoritmos de búsqueda para obtener la función de complejidad de cada uno de ellos.

Se obtendrá la función de complejidad de cada uno de los 3 algoritmos implementados, para poder estimar la cantidad de tiempo que tarda el algoritmo en ejecutarse.

Se tomará únicamente las operaciones básicas que realiza el algoritmo.

Cabe destacar que en este tipo de algoritmos nos encontramos con instancias, por lo que todo dependerá del tamaño del problema y la instancia que se encuentre en ese problema.

Para hacer el análisis teórico, el algoritmo se analizará partiéndolo en 3 casos:

- Mejor caso
- Caso medio
- Peor Caso

Procedemos entonces con el análisis:

- Búsqueda Lineal:

```
int buscarNumero ( int * ar , int tam , int numBuscado ) {
    for ( int i = 0 ; i < tam ; i ++ ) {
        if ( ar [ i ] == numBuscado ) return i ;
    }
    return - 1 ;
}
```

*Operación básica: `if ( ar [ i ] == numBuscado ) return i ;`

Mejor caso:

El mejor caso es cuando el número buscado se encuentra en el primer índice del arreglo. Así, sólo se realiza una comparación para que termine el algoritmo, por lo tanto

$$f_t(n) = 1$$

Peor caso:
El peor caso es cuando el número se encuentra al final del arreglo o cuando no se encuentra. Así, se realizarán comparaciones hasta que llegue al final del arreglo, siendo entonces

$$f_t(n) = n$$

Caso medio:
Se pueden presentar dos casos, donde al realizar la comparación encuentre el número o donde no se encuentre, por lo tanto, suponiendo que la probabilidad para cada caso es la misma en promedio se harían

$$f_t(n) = \frac{1}{2} n \; Operaciones \; básicas$$

- Búsqueda Binaria:

```
1 void BusquedaBinaria(int *A, int num, int n){
2 int superior, inferior;
3 inferior = 0;
4 superior = n;
4 while((inferior <= superior) && (encontrado == 0))
5 {
6       mitad =(inferior+superior)/2;
7       if(A[mitad]==num){
8               encontrado = 1;
9       }
10      else if(A[mitad]>num){
11              superior = mitad - 1;
12      }
13      else{
14              inferior = mitad + 1;
15      }
16 }
17 }
```

Como se mencionó anteriormente, procedemos a encontrar nuestras operaciones básicas, las cuales serán las siguientes:

1. Comparación que determina si se encontró el numero.
2. Primera asignación de la mitad.
3. Asignación de nuevo superior.
4. Asignación de nuevo inferior.
5. Comparación de numero con la mitad del arreglo.

Teniendo entonces:

```
1 void BusquedaBinaria(int *A, int num, int n){
2 int superior, inferior;
3 inferior = 0;
4 superior = n;
4 while((inferior <= superior) && (encontrado == 0))   —> Log2n + 1 comparaciones
5 {
6       mitad =(inferior+superior)/2;   —> 1 Asignación
7       if(A[mitad]==num){              —> 1 Comparación
8               encontrado = 1;
9       }
10      else if(A[mitad]>num){          —> 1 Comparación
11              superior = mitad - 1;   —> 1 Asignación
12      }
13      else{
14              inferior = mitad + 1;   —> 1 Asignación
15      }
16 }
17 }
```

Para el mejor caso de este algoritmo, solo se ejecuta una comparación del while, la asignación de la linea 6 junto con la comparación de la linea 7, por lo que

$$ftm(t) = 3$$

En el peor caso se ejecutarán $Log_2 n + 1$ comparaciones del ciclo while, esto debido a que nuestra variable de control está siendo dividida.
Luego dentro del ciclo, se ejecutará la linea 6,7,10 y 14, por lo que:

$$ftp(t) = (Log_2 n + 1)(4)$$

$$ftp(t) = 4Log_2 n + 4$$

Para el caso medio, analizamos los caminos que puede tomar nuestro algoritmo. Encontramos pues 3 caminos que puede tomar nuestro algoritmo.

Les proporcionaremos la misma probabilidad a cada caso de que este ocurra.

- Caso donde A[mitad]==num

  El algoritmo hace dentro del ciclo, para este caso, $Log_2 n + 1$ comparaciones, una asignación y una comparación. Por lo que se tiene:

$$\frac{(2Log_2 n + 2)}{3}$$

- Caso donde A[mitad]>num

  El algoritmo hace dentro del ciclo, para este caso, $Log_2 n + 1$ comparaciones, y dentro del ciclo dos asignaciones (6 y 11) y dos comparaciones (7 y 10). Por lo que se tiene:

$$\frac{4(Log_2 n + 1)}{3}$$

$$\frac{4Log_2 n + 4)}{3}$$

- Caso donde A[mitad]<num

  El algoritmo hace dentro del ciclo, para este caso, $Log_2 n + 1$ comparaciones, y dentro del ciclo dos asignaciones (6 y 14) y dos comparaciones (7 y 10). Por lo que se tiene:

$$\frac{4(Log_2 n + 1)}{3}$$

$$\frac{4Log_2 n + 4)}{3}$$

Por lo tanto la función de complejidad del caso medio sería:

$$ftme(t) = 2\frac{4(Log_2 n+1)}{3} + \frac{2(Log_2 n+1)}{3}$$

$$ftme(t) = \frac{8(Log_2 n+1)}{3} + \frac{2(Log_2 n+1)}{3}$$

$$ftme(t) = (Log_2 n + 1)(\frac{8}{3} + \frac{2}{3})$$

$$ftme(t) = \frac{10(Log_2 n+1)}{3}$$

- Árbol de Búsqueda Binaria:

```
1.   void busqueda ( struct Parametros * parametros ) {
2.
3.     if ( parametros -> raiz == NULL ) {
4.         //Esta hoja no existe y no se ha encontrado nada
5.     }
6.     else if ( parametros -> raiz -> dato == parametros -> numBusq )
   {
7.         //Si se encontro
8.         parametros -> encontrado = 1 ;
9.     }
10.          else if ( parametros -> numBusq > parametros -> raiz ->
   dato ) {
11.              parametros -> raiz = parametros -> raiz -> der ;
12.              busqueda ( parametros ) ;
13.          }
14.          else if ( parametros -> numBusq < parametros -> raiz ->
   dato ) {
15.              parametros -> raiz = parametros -> raiz -> izq ;
16.              busqueda ( parametros ) ;
17.          }
18.      }
```

*Operaciones básicas:

```
6.    else if ( parametros -> raiz -> dato == parametros -> numBusq )
7.    else if ( parametros -> numBusq > parametros -> raiz -> dato )
8.    else if ( parametros -> numBusq < parametros -> raíz -> dato )
```

Mejor caso:
El mejor caso es cuando el número buscado se encuentra en el nodo raíz del árbol. Así, solo se realiza la primera operación básica, por lo tanto

$$f_t(n) = 1$$

Peor caso:
Cuando el número buscado es el menor de todos por lo que se encuentra en el último hijo izquierdo o cuando no se encuentra en el árbol. Así se recorre todo el árbol realizando las 3 operaciones básicas en cada nodo, resultando

$$f_t(n) = 3\log_2(n)$$

Caso medio:

Se tienen tres posibles casos:

- cuando el número buscado está en el nodo actual (**1 comparación**),
- cuando el número buscado es mayor que el valor en el nodo actual (**2 comparaciones**)
- y cuando el número buscado es menor que el valor en el nodo actual (**3 comparaciones**).

Caso A[i] == numBuscado:

$$\frac{1}{3}(\log_2 n)$$

Caso A[i] > numBuscado:

$$\frac{1}{3}(2\log_2 n)$$

Caso A[i] < numBuscado:

$$\frac{1}{3}(3\log_2 n)$$

Caso medio:

$$f_t(n) = \frac{1}{3}(\log_2 n + 2\log_2 n + 3\log_2 n) = \frac{1}{3}(6\log_2 n) = 2\log_2 n$$

$$f_t(n) = 2\log_2 n \ \ Operaciones\ básicas$$

II. Registros de tiempos promedios de búsqueda para cada algoritmo.

Se tienen los siguientes 20 diferentes números a buscar:

A=[322486, 14700764, 3128036, 6337399, 61396, 10393545, 2147445644, 1295390003, 450057883, 187645041, 1980098116, 152503, 5000, 1493283650, 214826, 1843349527, 1360839354, 2109248666 , 2147470852 y 0].

Cada uno de estos números, se buscó en 20 diferentes tamaños de problemas (n) para cada uno de los 3 diferentes algoritmos de búsqueda.

Siendo los siguientes tamaños de problema (n):

N=[100, 1000, 5000, 10000, 50000, 100000, 200000, 400000, 600000, 800000, 1000000, 2000000, 3000000, 4000000, 5000000, 6000000, 7000000, 8000000, 9000000 y 10000000]

Se obtuvo un promedio de tiempo de búsqueda (real) para cada tamaño de problema y para cada algoritmo de búsqueda, obteniendo los siguientes registros:

Tabla1. Tiempos promedio de búsqueda para búsqueda lineal sin hilos

Búsqueda Lineal (sin hilos)	
Tamaño de n	**Promedio (Sx10000)**
100	6.77037239
1000	6.62398338
5000	6.5677166
10000	6.97040558
50000	9.0816021
100000	10.428071
200000	13.6333704
400000	18.2108879
600000	23.5812664
800000	28.6357403
1000000	33.793211
2000000	56.6482544
3000000	80.1684856
4000000	104.20835
5000000	125.207782
6000000	152.082562
7000000	166.982889
8000000	187.201262
9000000	202.702522
10000000	219.544053

Tabla2. Tiempos promedio de búsqueda para búsqueda lineal con hilos

Búsqueda Lineal (con hilos)	
Tamaño de n	**Promedio (Sx10000)**
100	7.64715672
1000	7.63618946
5000	7.70664215
10000	8.43417645
50000	9.32097435
100000	10.0529194
200000	12.223959
400000	15.4042244
600000	18.4805393
800000	23.3414173
1000000	24.3625641
2000000	40.5329466
3000000	55.675745
4000000	70.7776546
5000000	84.5522881
6000000	103.721738
7000000	133.046269
8000000	155.016899
9000000	170.863628
10000000	178.785443

Tabla3. Tiempo promedio de búsqueda para búsqueda binaria

Búsqueda Binaria (sin hilos)	
Tamaño de n	**Promedio (Sx10000)**
100	6.375074387
1000	6.530165672
5000	6.233811378
10000	6.282567978
50000	6.930947304
100000	7.337450981
200000	8.0524683
400000	7.671833038
600000	7.4249506
800000	8.228063583
1000000	7.99369812
2000000	7.917881012
3000000	8.337855339
4000000	7.655739784
5000000	8.048057556
6000000	8.30411911
7000000	7.959485054
8000000	7.889986038
9000000	7.897019386
10000000	7.740020752

Tabla4. Tiempos promedio de búsqueda para búsqueda binaria con hilos

Búsqueda Binaria (con hilos)	
Tamaño de n	**Promedio (Sx10000)**
100	7.934570313
1000	7.855176926
5000	7.834672928
10000	7.708907127
50000	8.473992348
100000	9.045004845
200000	9.358167648
400000	9.448885918
600000	9.095668793
800000	9.301185608
1000000	9.614348412
2000000	9.294390678
3000000	9.376645088
4000000	9.404063225
5000000	9.568333626
6000000	9.473443031
7000000	9.422898293
8000000	9.315609932
9000000	9.593605995
10000000	9.451627731

Tabla5. Tiempos promedio de búsqueda para búsqueda en árbol binario sin hilos

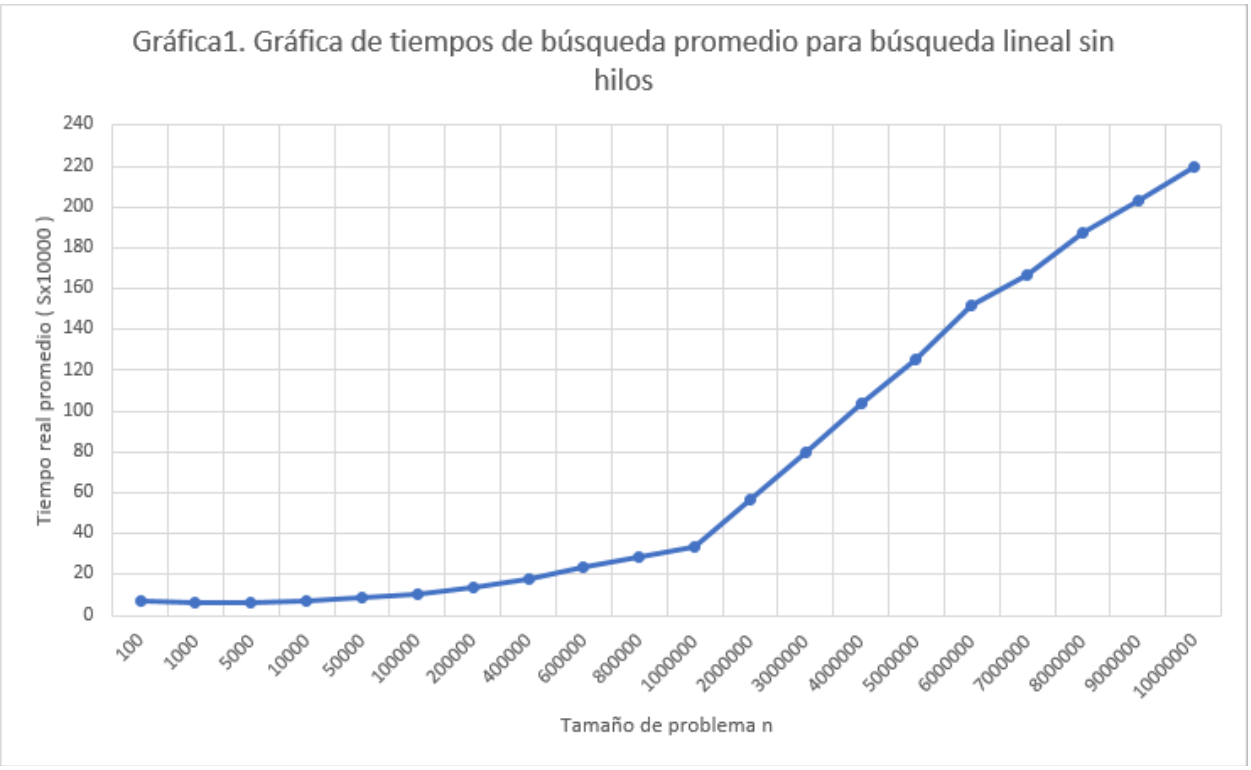
Búsqueda en árbol binario de búsqueda (sin hilos)	
Tamaño de n	**Promedio (Sx10000)**
100	6.68418407
1000	6.45971298
5000	6.37745857
10000	7.05015659
50000	7.34055042
100000	7.88808746
200000	7.43480873
400000	7.3390007
600000	7.49051571
800000	7.62096081
1000000	7.71650028
2000000	7.84850121
3000000	8.25881958
4000000	8.31568241
5000000	8.44378109
6000000	8.44606266
7000000	8.31913948
8000000	8.79454613
9000000	8.39442844
10000000	8.42583179

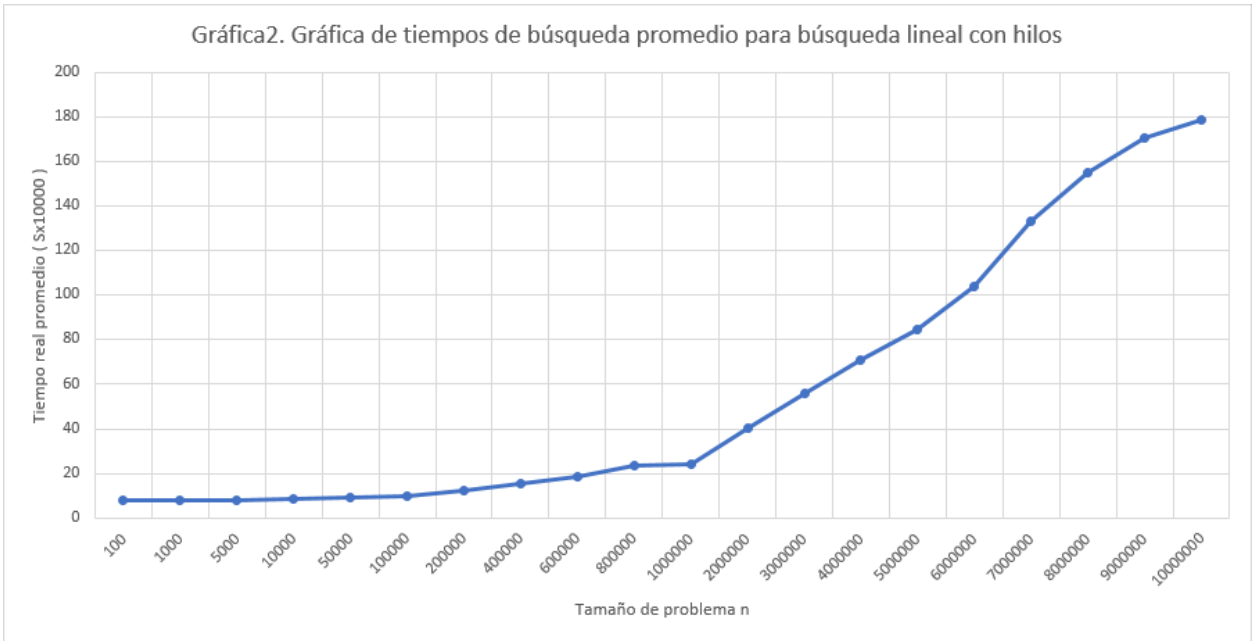
Tabla6. Tiempos promedio de búsqueda para búsqueda en árbol binario con hilos

Búsqueda en árbol binario de búsqueda (con hilos)	
Tamaño de n	**Promedio (Sx10000)**
100	7.3165893
1000	7.5182914
5000	8.4034204
10000	8.5960626
50000	9.054660
100000	9.0178251
200000	9.3928575
400000	9.5489025
600000	9.2351436
800000	9.54234
1000000	9.7953081
2000000	9.4863176
3000000	9.8536014
4000000	9.3638896
5000000	9.5775127
6000000	9.8812580
7000000	9.7949504
8000000	9.6145868
9000000	9.5491409
10000000	9.7616910

III. Graficación de comportamiento temporal de cada algoritmo considerando los tiempos promedios de búsqueda.

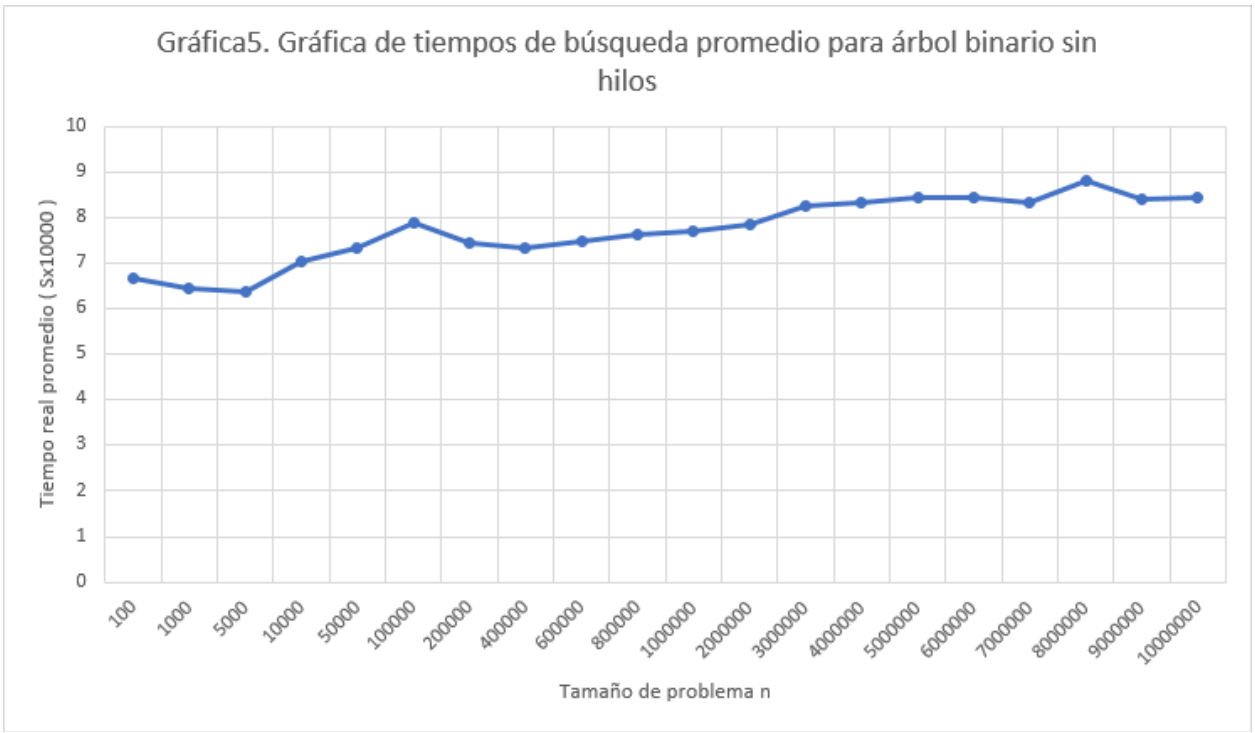
Considerando los tiempos promedios de búsqueda de cada número de A en cada n, representados en las tablas del inciso II, se graficará el comportamiento de cada uno de los algoritmos. Obteniendo las siguientes gráficas:

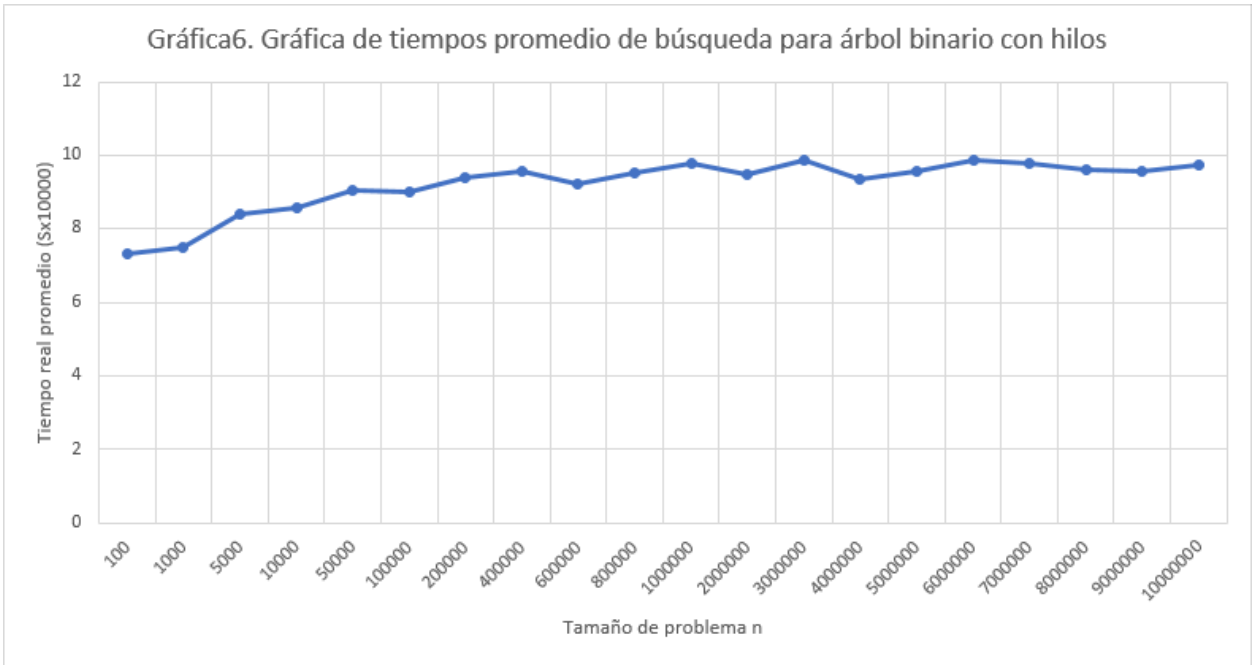



Gráfica2. Gráfica de tiempos de búsqueda promedio para búsqueda lineal con hilos
Tiempo real promedio (Sx10000)
0
20
40
60
80
100
120
140
160
180
200
100
1000
5000
10000
50000
100000
200000
400000
600000
800000
1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000
9000000
10000000
Tamaño de problema n

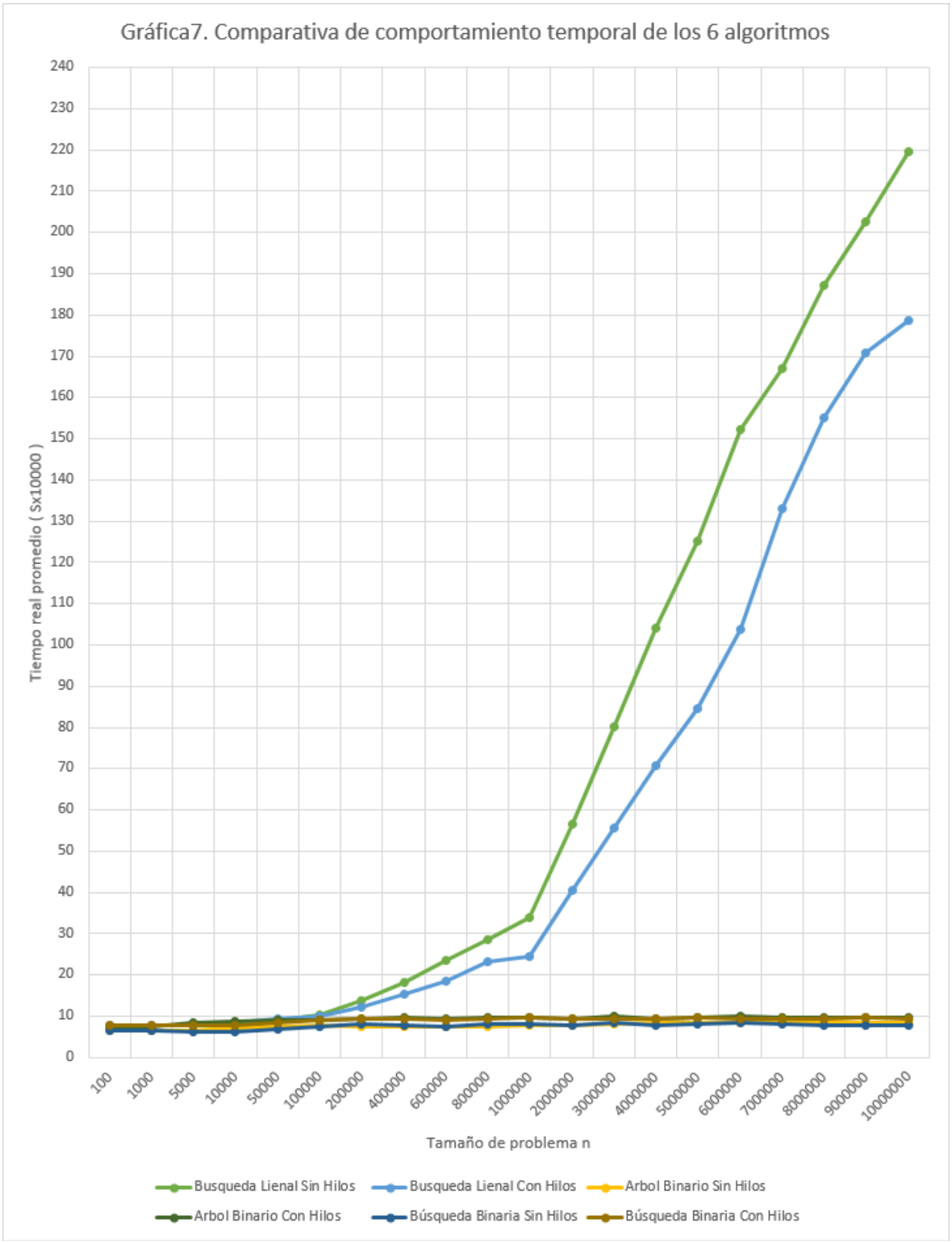

Gráfica3. Gráfica de tiempos de búsqueda promedio para búsqueda binaria sin hilos
Tiempo real promedio (Sx10000)
0
1
2
3
4
5
6
7
8
9
100
1000
5000
10000
50000
100000
200000
400000
600000
800000
1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000
9000000
10000000
Tamaño de problema n


Gráfica4. Gráfica de tiempos de búsqueda promedio para búsqueda binaria con hilos
Tiempo real promedio (Sx10000)
0
2
4
6
8
10
12
100
1000
5000
10000
50000
100000
200000
400000
600000
800000
1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000
9000000
10000000
Tamaño de problema n


Gráfica5. Gráfica de tiempos de búsqueda promedio para árbol binario sin hilos
Tiempo real promedio (Sx10000)
0
1
2
3
4
5
6
7
8
9
10
100
1000
5000
10000
50000
100000
200000
400000
600000
800000
1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000
9000000
10000000
Tamaño de problema n


Gráfica6. Gráfica de tiempos promedio de búsqueda para árbol binario con hilos
Tiempo real promedio (Sx10000)
0
2
4
6
8
10
12
100
1000
5000
10000
50000
100000
200000
400000
600000
800000
1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000
9000000
10000000
Tamaño de problema n

IV. Comparación de los 3 algoritmos y sus variantes con hilos.



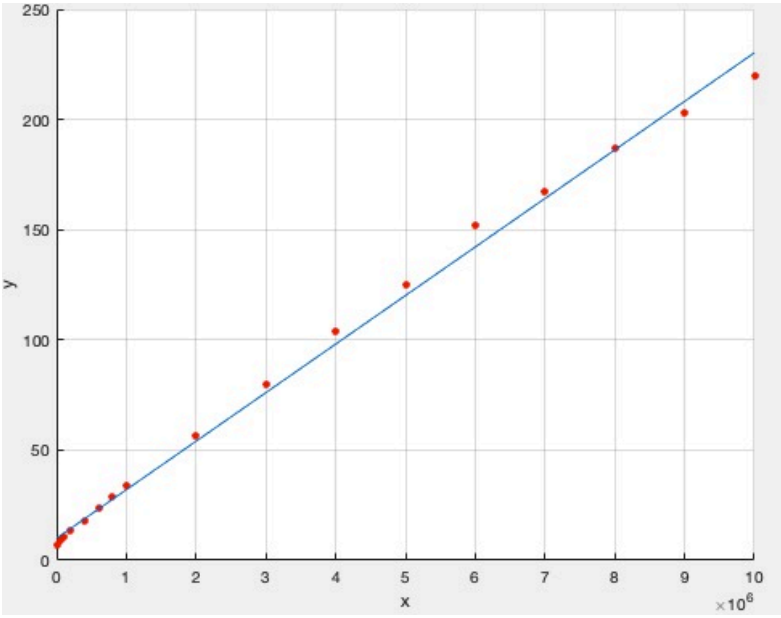
V. Aproximación polinomial del comportamiento temporal (Tiempo real promedio) de cada algoritmo.

A continuación se muestra aproximaciones polinomiales del comportamiento temporal, con base en los datos obtenidos de los tiempos promedios para cada n.

Para esto se realizaron 3 aproximaciones con 3 diferentes grados de polinomios para poder elegir el que más se acercara al comportamiento experimental. Conforme se van proponiendo, se va eligiendo el que mejor se apega a nuestras experimentaciones.

I. Búsqueda lineal sin hilos

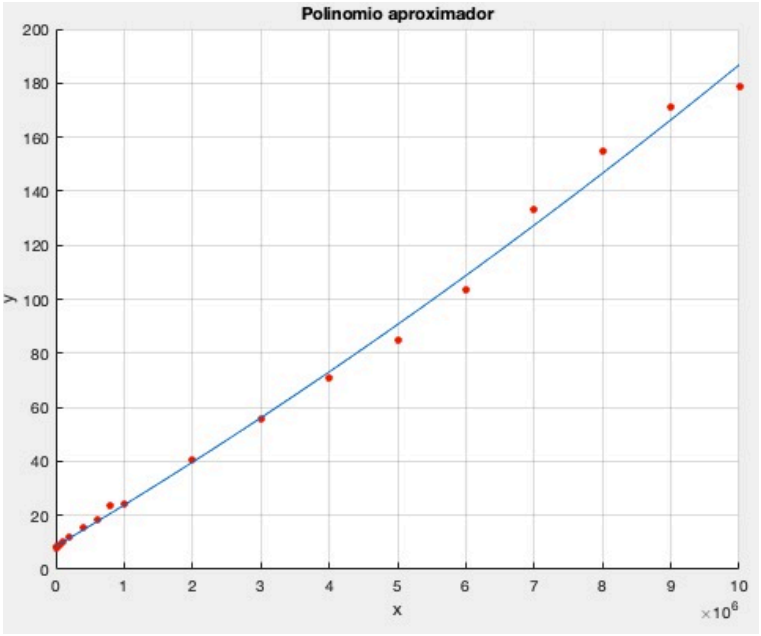
- Polinomio grado 1



$$2.202506005098645e-05x+9.997800109415246$$

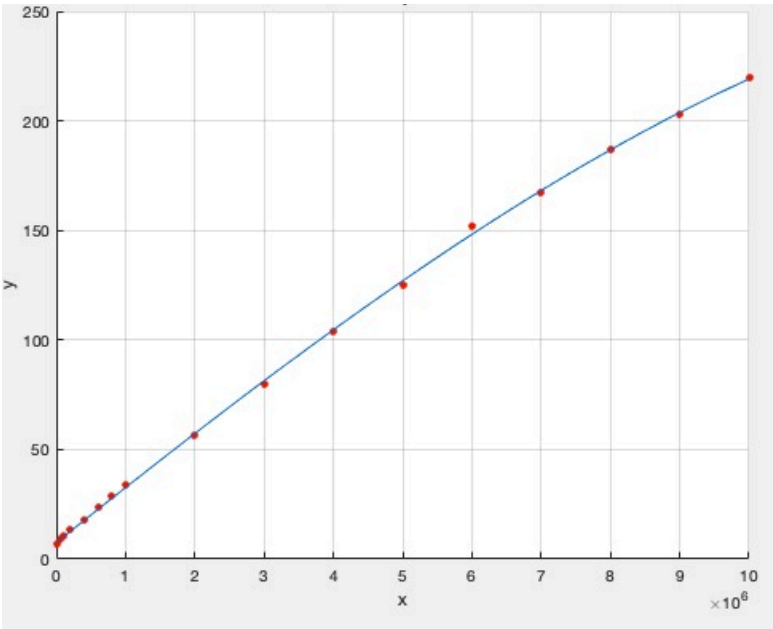
- Polinomio grado 2

$$-5.265104616180772e-13x^2+2.656647929544767e-05x+7.184288176038183$$



- Polinomio grado 3

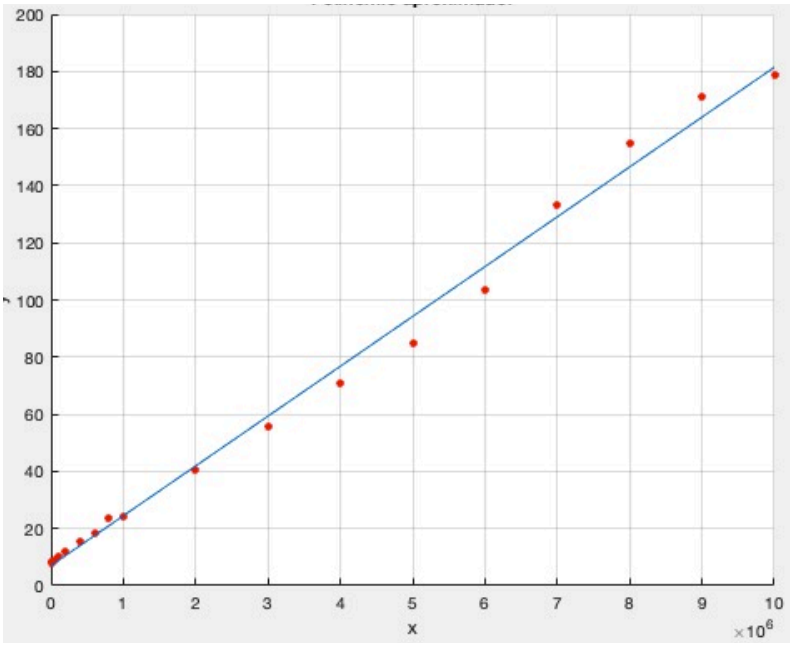
$$-2.635250116295894e-20x^3-1.514191713211754e-13x^2+2.530446809761038e-05x+7.535122866676127$$



II. Búsqueda lineal con hilos

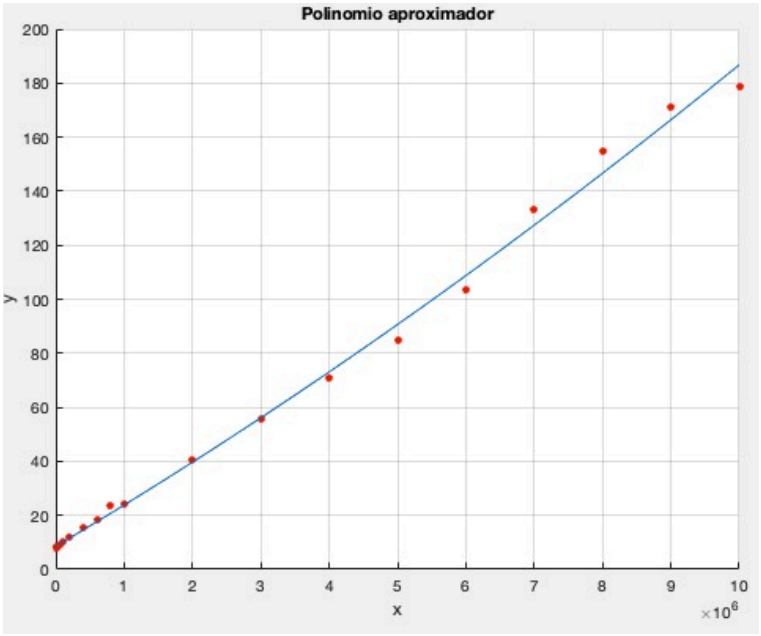
- Polinomio grado 1

$$1.744121258545705e-05x+7.026863611525228$$



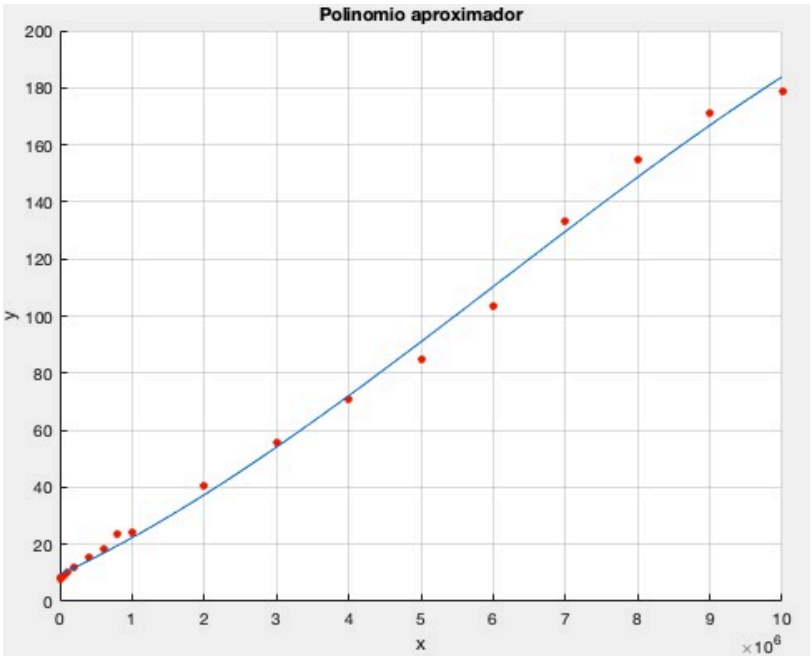
- Polinomio grado 2

$$2.802282075113355e-13x^2+1.502410244766076e-05x+8.524318008687590$$



- Polinomio grado 3

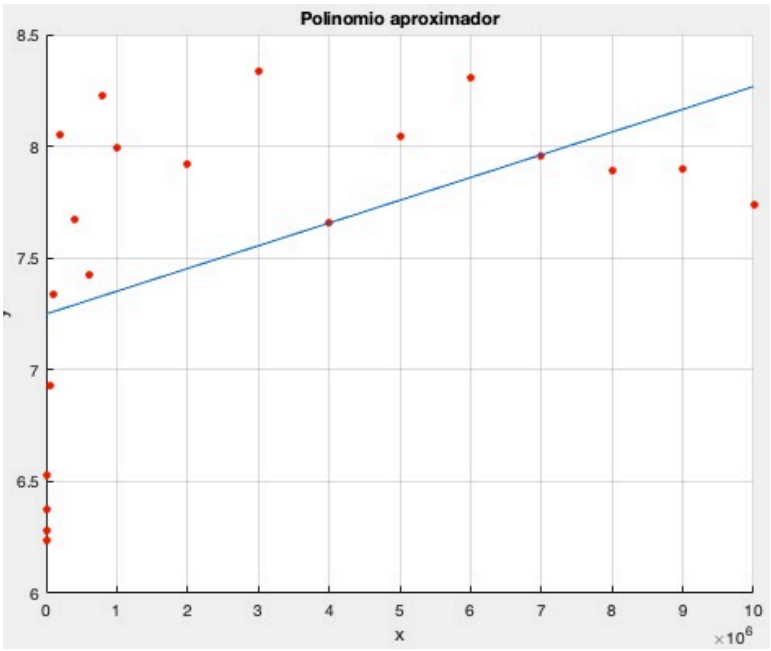
$$-6.936220347917363e-20x^3+1.267502934868502e-12x^2+1.170237306204828e-05x+9.447747137828003$$



III. Búsqueda binaria sin hilos

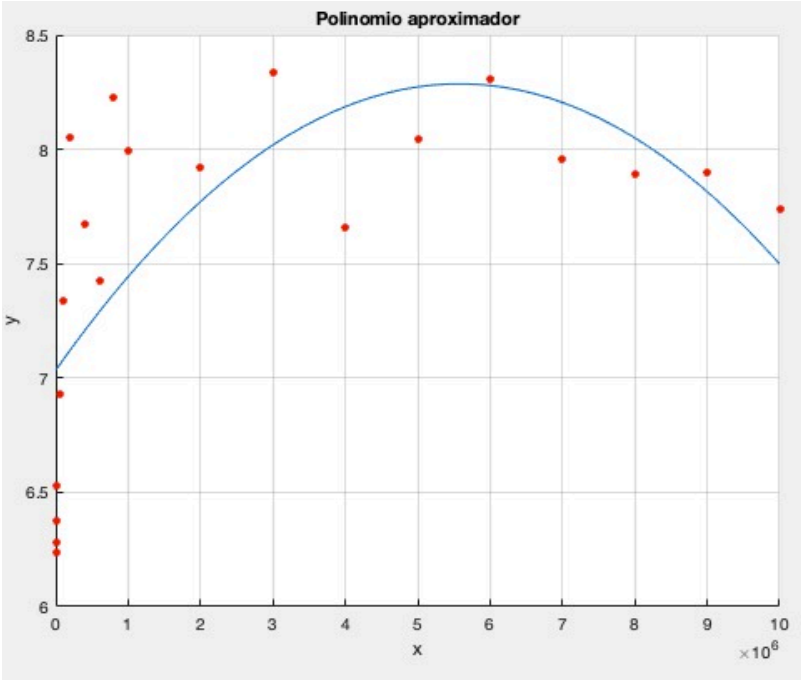
- Polinomio grado 1

$$1.017634174899712e-07x+7.249688883571329$$



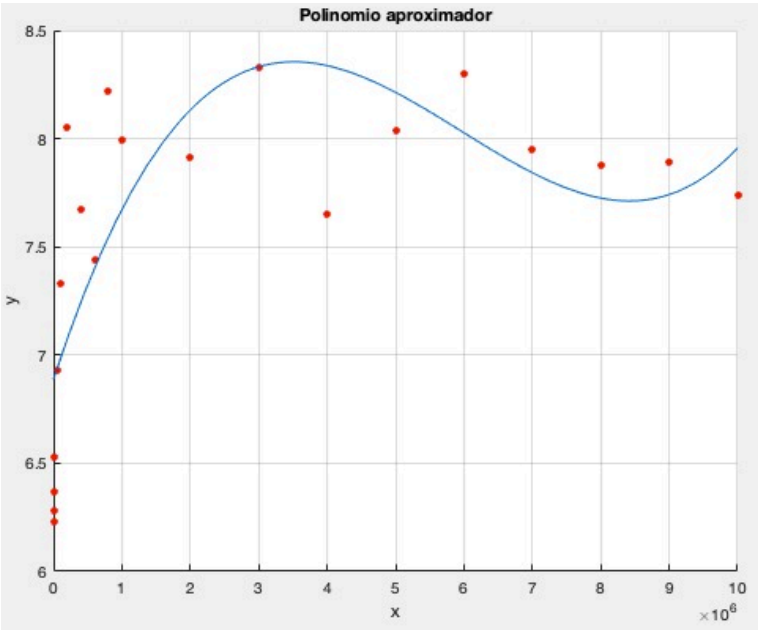
- Polinomio grado 2

$$-4.026426041544096e-14x^2 + 4.490630448377829e-07 + 7.034528909694839$$



- Polinomio grado 3

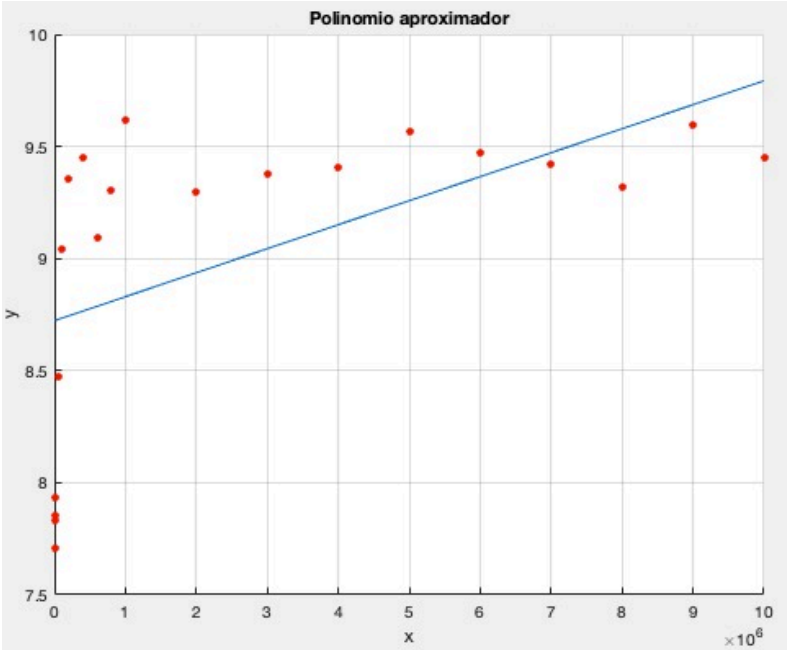
$$1.092525568555496e-20x^3 - 1.955561635795619e-13x^2 + 9.699389380621602e-07x + 6.887549109239086$$



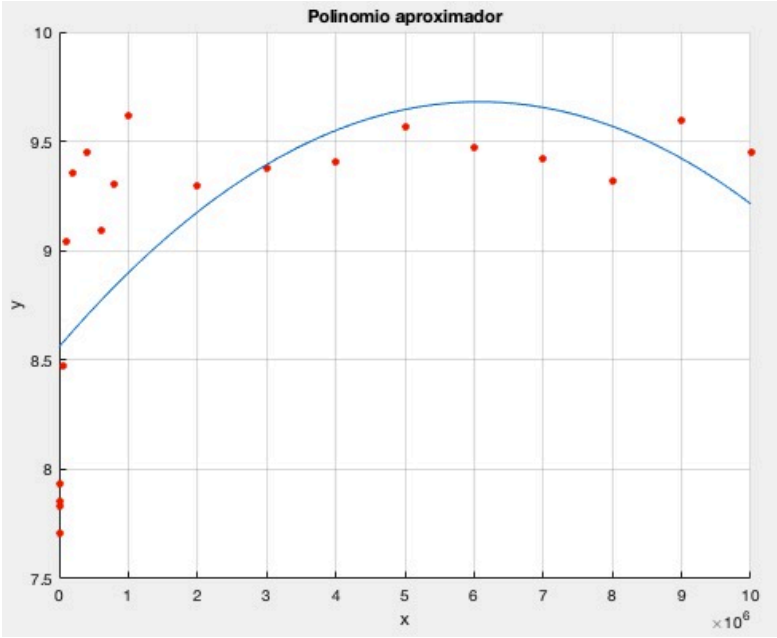
IV. Búsqueda binaria con hilos

- Polinomio grado 1

$$1.070425620077106e-07x+8.722599633050551$$

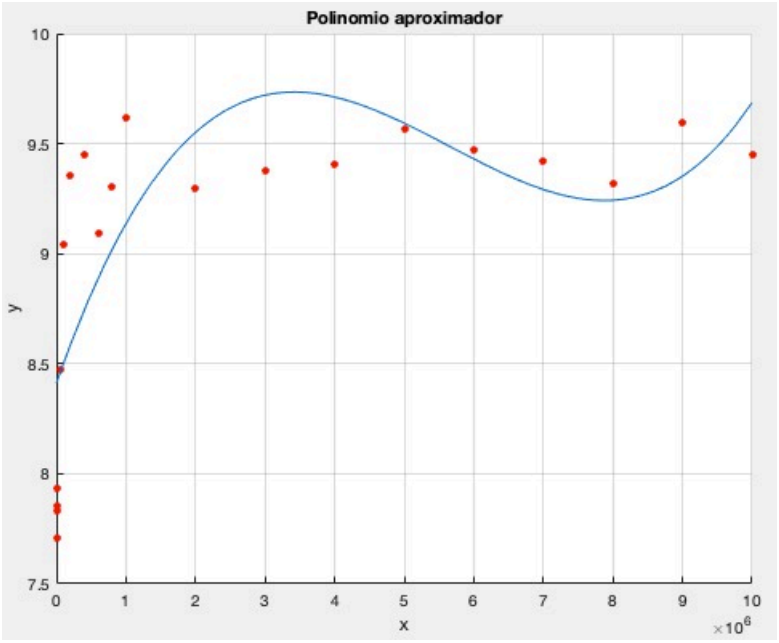


- Polinomio grado 2
  $-3.032937988004987e-14x^2+3.686488158256021e-07x+8.560528642168048$



- Polinomio grado 3

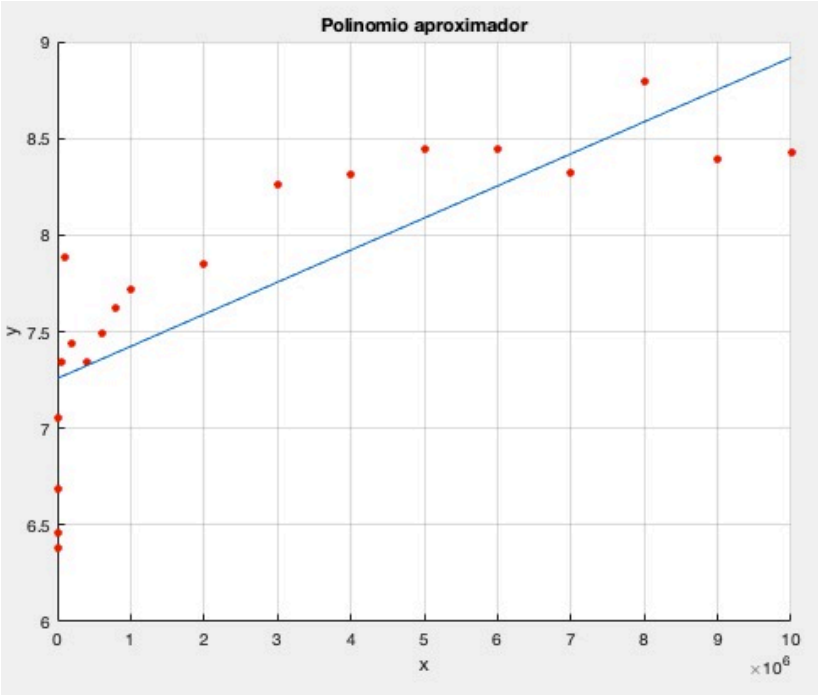
$$1.117740557798178e-20x^3 - 1.894242397755524e-13x^2 + 9.039304923144293e-07x + 8.411722211847186$$



V. Búsqueda por árbol binario de búsqueda sin hilos

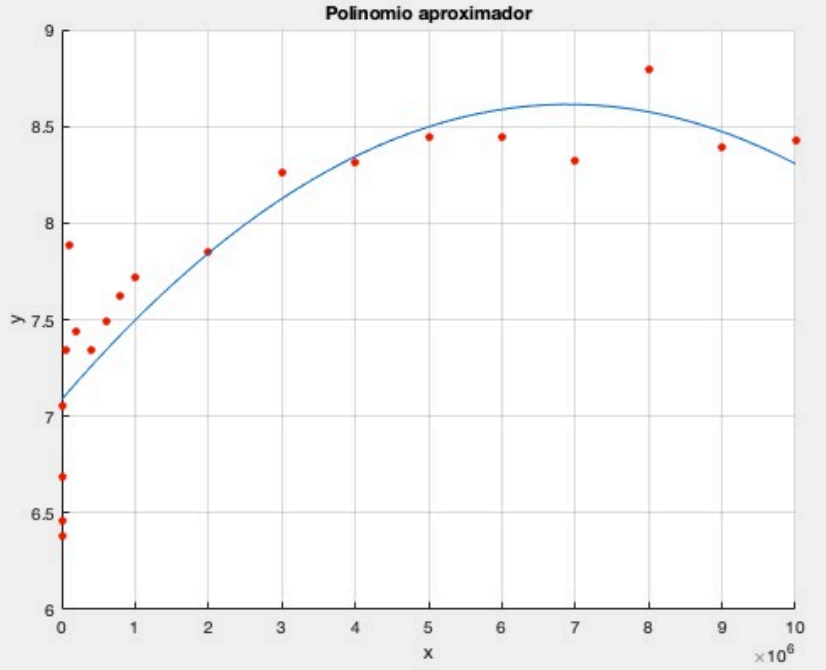
- Polinomio grado 1

$$1.660226481808960e-07x+7.257893091191304$$



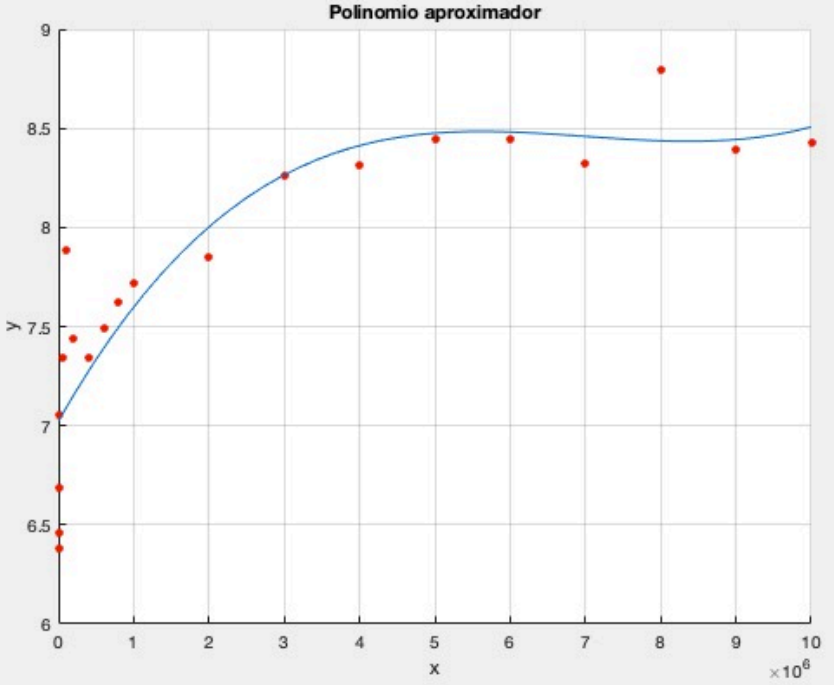
- Polinomio grado 2

$$-3.206760932560426e-14x^2+4.426220108366820e-07x+7.086533530290677$$



- Polinomio grado 3

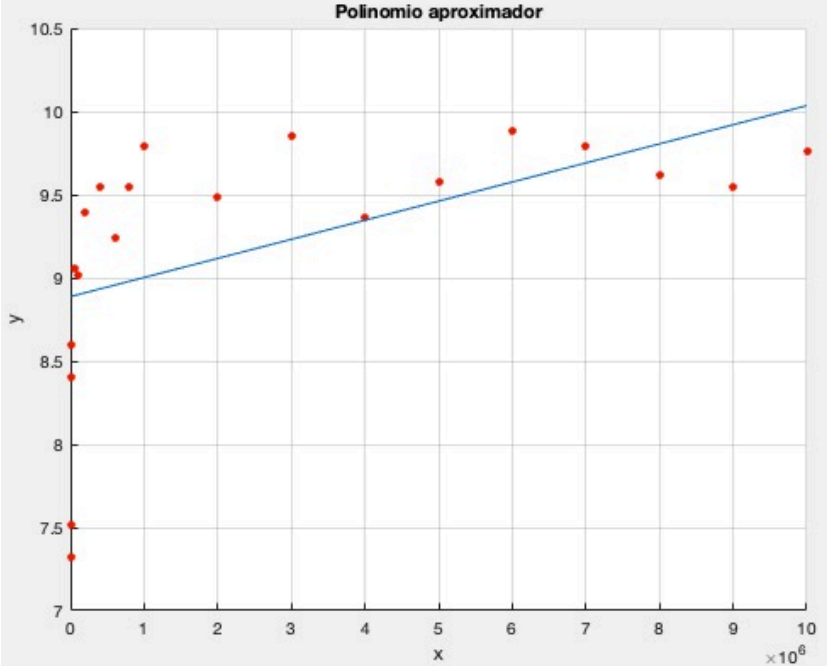
$$4.750775731563285e-21x^3-9.968831022393276e-14x^2+6.701348429041753e-07x+7.023285760230197$$



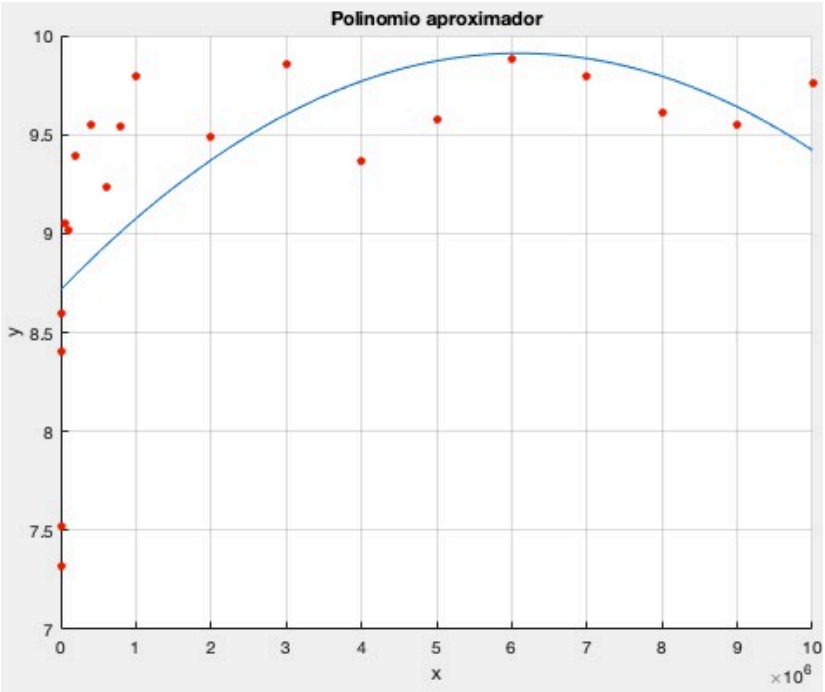
VI. Búsqueda por árbol binario de búsqueda con hilos

- Polinomio grado 1

$$1.147399435342051e-07x+8.887256074446464$$



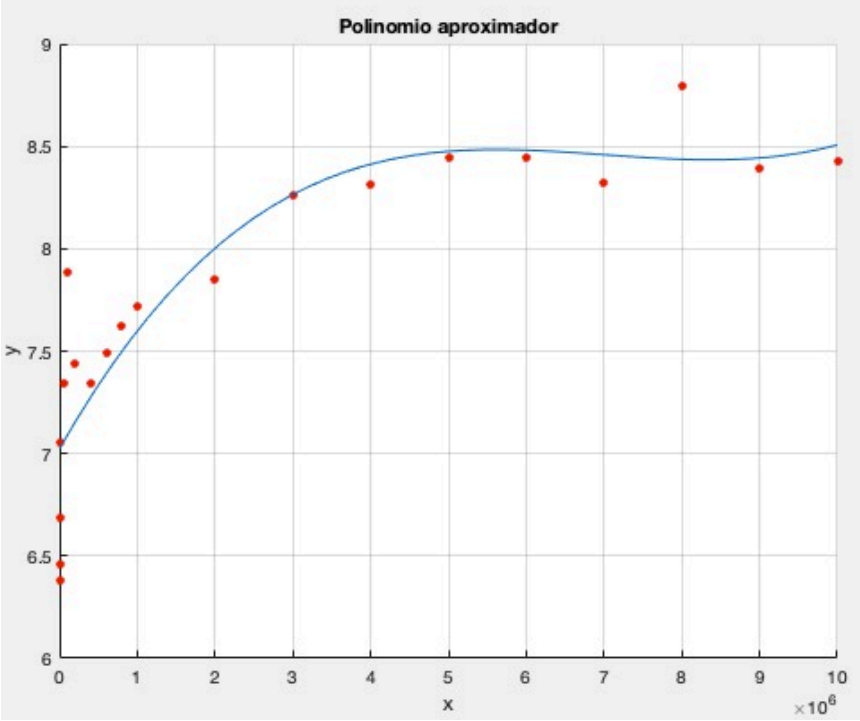
- Polinomio grado 2



$$-3.216320102657428e-14x^2+3.921638329981801e-07x+8.715385700540175$$

- Polinomio grado 3

$$1.122935346616971e-20x^3 - 1.919974670334024e-13x^2 + 9.299332739762958e-07x + 8.565887680408958$$

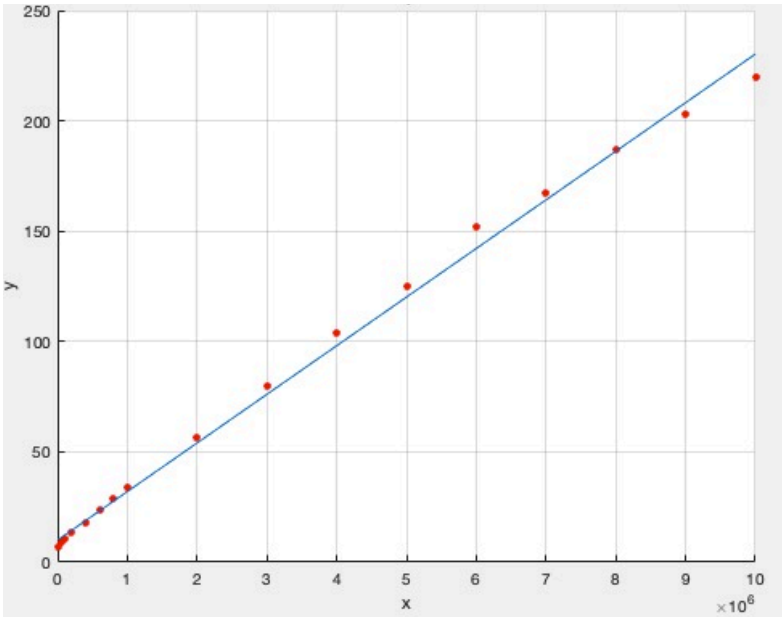


VI. Mostrar gráficamente la comparativa de las aproximaciones de cada algoritmo. Determinar si el comportamiento experimental se aproxima a lo esperado teóricamente.

Búsqueda lineal sin hilos

- Polinomio grado 1

$$2.202506005098645e-05x+9.997800109415246$$

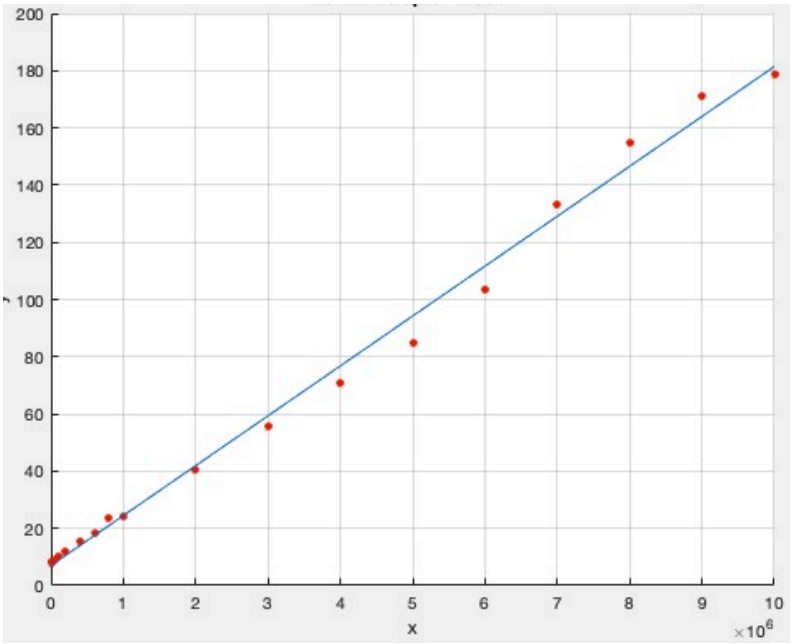


La aproximación más acertada fue esta, debido a que es la que mejor refleja el comportamiento de nuestros datos arrojados por la computadora, a diferencia de la cuadrática y la cúbica. Esta aproximación es la más certera de todas las analisadas.

Búsqueda lineal con hilos

- Polinomio grado 1

$$1.744121258545705e-05x+7.026863611525228$$

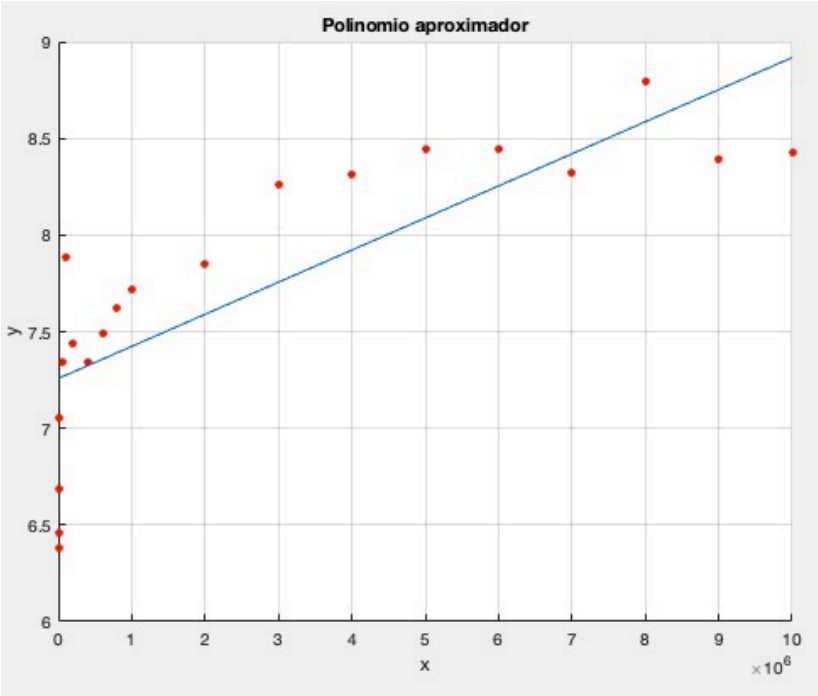


Esta fue la aproximación que mejor reflejó nuestros puntos, aunque no toca con varios puntos, es la más acercada a nuestros datos que se pudo encontrar.

Búsqueda por árbol binario de búsqueda sin hilos

- Polinomio grado 1

$$1.660226481808960e-07x+7.257893091191304$$

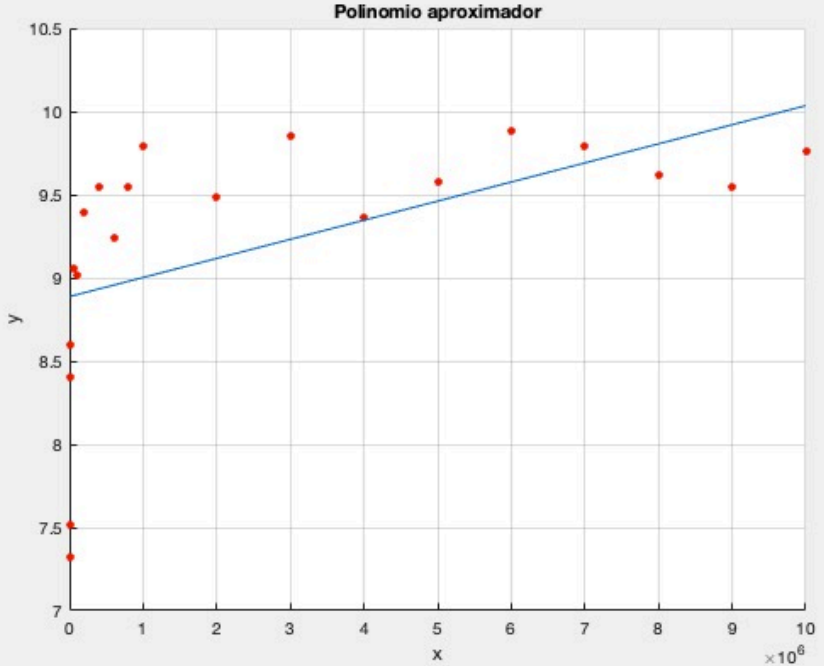


Una vez más la aproximación de grado uno es la elegida, pues aunque falla en varios puntos, no es tan grave que esto suceda, pues en realidad los puntos tienen valores muy pequeños respecto a nuestro dominio.

Búsqueda por árbol binario de búsqueda con hilos

- Polinomio grado 1

$$1.147399435342051e-07x+8.887256074446464$$

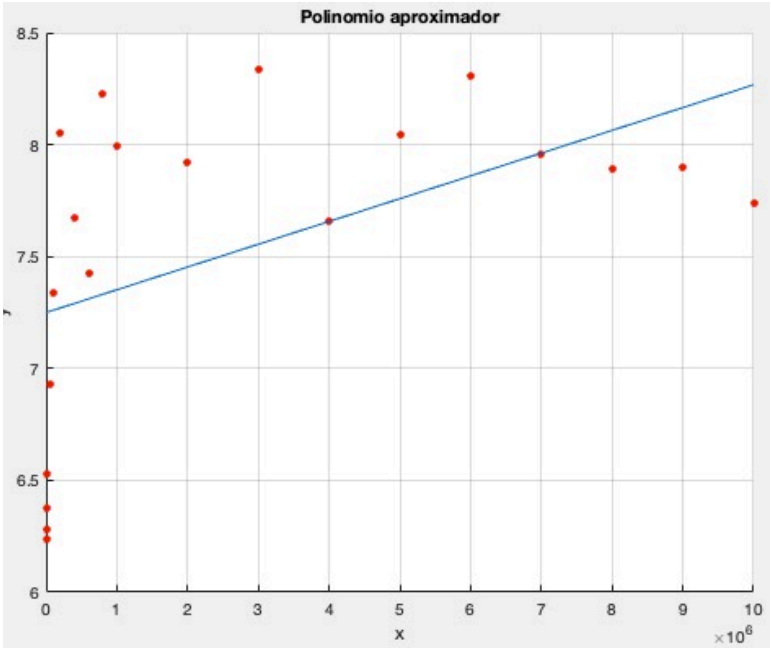


La diferencia que existe entre los puntos y la aproximación no es tan grande cuando los números son más grandes que 1000, por lo que aunque no es precisa se elige como la mejor aproximación. Pues los comportamientos de la cuadrática y la cúbica demuestran varios problemas.

Búsqueda binaria sin hilos

- Polinomio grado 1

$$1.017634174899712e-07x+7.249688883571329$$

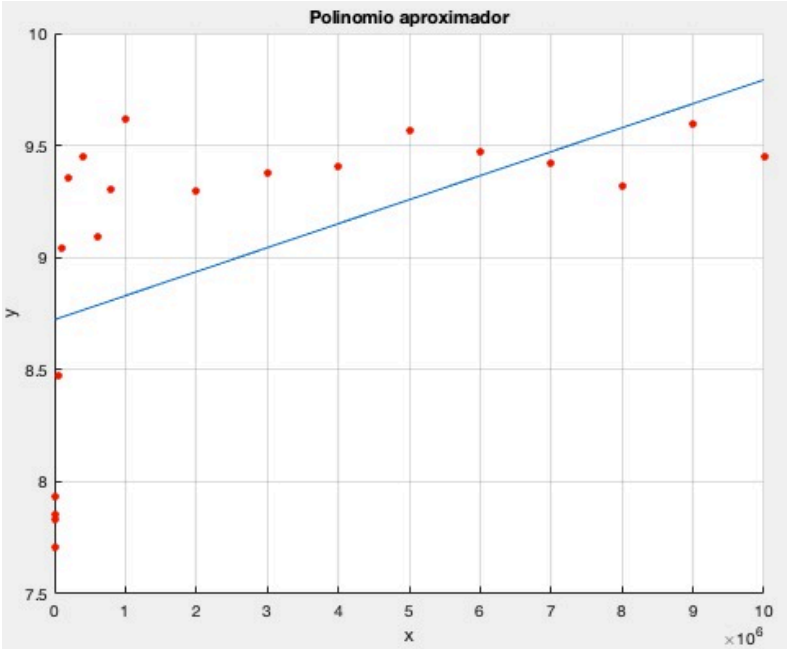


La diferencia que existe entre los puntos y la aproximación, nuevamente, no es tan grande cuando los números son más grandes que 1000, así que se elige como la más adecuada para representar el comportamiento de nuestros datos, al contrario que la cuadrática y la cúbica.

Búsqueda binaria con hilos

- Polinomio grado 1

$$1.070425620077106e-07x+8.722599633050551$$



Aunque claramente se ve que en algunos puntos nuestra aproximación falla, la diferencia que existe entre los puntos y la aproximación no es tan grande cuando los números son más grandes que 1000, así que se elige como la más adecuada para representar el comportamiento de nuestros datos, al contrario que la cuadrática y la cúbica.

VII. Determine cuánto le lleva a la computadora ejecutar cada paso básico del algoritmo.

Para encontrar el tiempo que le toma a la computadora ejecutar cada paso básico, tomamos la función de complejidad temporal del peor caso del algoritmo y sustituimos sus valores con los dados en algún peor caso de nuestros resultados, es decir, cuando no se encontró el número buscado.

Así, los resultados son los siguientes:

- Búsqueda Lineal.
  Fórmula del peor caso:

$$f_t(n) = n$$

  Peor caso tomado:

Número a buscar	Tamaño de n	Tiempo real	Encontrado
5000	10000000	307.269096s	No

  Sustituyendo:

$$307.269096s = 10{,}000{,}000\ Operaciones$$

  Para una operación:

$$\frac{307.269096s}{10{,}000{,}000} = 1\ Operación$$

$$0.0000307s = 1\ Operación$$

- Búsqueda Binaria.

  Fórmula del peor caso:

$$ftp(t) = 4Log_2 n + 4$$

Peor caso tomado:

Número a buscar	Tamaño de n	Tiempo real	Encontrado
2147470852	10000000	12.3906136	Si

Sustituyendo:

$$12.3906136s = 4Log_2(10{,}000{,}000) + 4\ Operaciones$$

Para una operación:

$$\frac{12.3906136s}{97.01398666} = 1\ Operación$$

$$0.1277198s = 1\ Operación$$

- Árbol Binario de Búsqueda

Fórmula del peor caso:

$$f_t(n) = 3\log_2(n)$$

Peor caso tomado:

Número a buscar	Tamaño de n	Tiempo real	Encontrado
187645041	10000000	9.54866409	No

Sustituyendo:

$$9.54866409s = 3Log_2(10{,}000{,}000)\ Operaciones$$

Para una operación:

$$\frac{12.3906136s}{69.7604899} = 1\ Operación$$

$$0.1776164s = 1\ Operación$$

VIII. Determine cuál será el tiempo de búsqueda para de cada algoritmo para un tamaño n de:

- **50,000,000**

  Búsqueda Lineal:
  Operaciones a realizar = n = 50,000,000
  Tiempo total = (50,000,000)(0.0000307s) = 1,535s

  Búsqueda Binaria:
  Operaciones a realizar = $4Log_2(n) + 4$ = 106.301699
  Tiempo total = (106.301699)(0.1277198s) = 13.5768317s

  Árbol Binario de Búsqueda:
  Operaciones a realizar = $3Log_2(n)$ = 76.72627428
  Tiempo total = (76.72627428)(0.1776164s) = 13.6278446s

- **100,000,000**

  Búsqueda Lineal:
  Operaciones a realizar = n = 100,000,000
  Tiempo total = (100,000,000)(0.0000307s) = 3,070s

  Búsqueda Binaria:
  Operaciones a realizar = $4Log_2(n) + 4$ = 110.301699
  Tiempo total = (110.301699)(0.1277198s) = 14.0877109s

  Árbol Binario de Búsqueda:
  Operaciones a realizar = $3Log_2(n)$ = 79.72627428
  Tiempo total = (79.72627428)(0.1776164s) = 14.1606938s

- **500,000,000**

  Búsqueda Lineal:

Operaciones a realizar = n = 500,000,000
Tiempo total = (500,000,000)(0.0000307s) = 15,350s

Búsqueda Binaria:
Operaciones a realizar = $4Log_2(n) + 4$ = 119.5894114
Tiempo total = (119.5894114)(0.1277198s) = 15.2739357s

Árbol Binario de Búsqueda:
Operaciones a realizar = $3Log_2(n)$ = 86.69205856
Tiempo total = (86.69205856)(0.1776164s) = 15.3979313s

- **1,000,000,000**

  Búsqueda Lineal:
  Operaciones a realizar = n = 1,000,000,000
  Tiempo total = (1,000,000,000)(0.0000307s) = 30,700s

  Búsqueda Binaria:
  Operaciones a realizar = $4Log_2(n) + 4$ = 123.5894114
  Tiempo total = (123.5894114)(0.1277198s) = 15.7848149s

  Árbol Binario de Búsqueda:
  Operaciones a realizar = $3Log_2(n)$ = 89.69205856
  Tiempo total = (89.69205856)(0.1776164s) = 15.9307805s

- **5,000,000,000**

  Búsqueda Lineal:
  Operaciones a realizar = n = 5,000,000,000
  Tiempo total = (5,000,000,000)(0.0000307s) = 153,500s

  Búsqueda Binaria:
  Operaciones a realizar = $4Log_2(n) + 4$ = 132.8771238
  Tiempo total = (132.8771238)(0.1277198s) = 16.9710396s

  Árbol Binario de Búsqueda:

Operaciones a realizar = $3Log_2(n)$ = 96.65784285
Tiempo total = (96.65784285)(0.1776164s) = 17.1680180s

IX. Indique las cotas O mayúscula para cada algoritmo con base en el análisis teórico del peor caso.

Búsqueda Binaria(Con hilos y sin hilos): **O(logn)**

Búsqueda Lineal(Con hilos y sin hilos): **O(n)**

Búsqueda Árbol Binario(Con hilos y sin hilos): **O(logn)**

X. Indique las cotas O mayúscula para cada aproximación polinomial seleccionada y justifique.

Búsqueda Binaria(Con hilos y sin hilos): **O(n)**

Búsqueda Lineal(Con hilos y sin hilos): **O(n)**

Búsqueda Árbol Binario(Con hilos y sin hilos): **O(n)**

Se determinaron estas cotas debido a que la aproximación polinomial era de grado 1, por lo cual fue fácil determinar que el grado de nuestra cota también debía serlo. Además, sabiendo nuestras cotas por el análisis teórico se intuye que la cota más próxima que domina a O(logn) es O(n)

XI. Responda a las siguientes preguntas.

1. ¿Cuál de los 3 algoritmos es más fácil de implementar?
   La Búsqueda Lineal. Su funcionamiento es bastante fácil de entender y sólo requiere de 4 líneas de código.

2. ¿Cuál de los 3 algoritmos es más difícil de implementar?
   El Árbol Binario de Búsqueda, debido a las diferentes funciones que se necesitan para inicializar el árbol con todos los números y las comparaciones entre los datos de cada nodo y el número a buscar.

3. ¿Cuál de los 3 algoritmos es el más difícil de implementar en su variante con hilos?
   Creemos que ninguno se dificultó más que otro, si acaso en la Búsqueda Binaria y en el Árbol Binario de Búsqueda al intentar encontrar una implementación que realmente mejorara el rendimiento,

4. ¿Cuál de los 3 algoritmos en su variante con hilos resultó ser más rápido?
   En la Búsqueda Lineal es en la única que realmente hay una diferencia en los resultados.

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

6. ¿Cuál algoritmo tiene mayor complejidad temporal?
   La búsqueda Lineal.

7. ¿El comportamiento experimental de los algoritmos era el esperado? ¿Por qué?
   Mayormente sí, obviamente hay muchos factores que no podemos tomar en cuenta durante el análisis teórico pero haciéndolo bien para las operaciones básicas de cada algoritmo podemos ver que los resultados concuerdan con lo esperado en éste.

8. ¿Sus resultados experimentales difieren mucho de los análisis teóricos que realizó? ¿A qué se debe?
   Podemos decir que no. Si comparamos las gráficas que resultarían de las funciones obtenidas en el análisis teórico y las obtenidas de la experimentación, podemos ver que se asemejan bastante. Esto se debe a que hicimos un análisis acorde al comportamiento del algoritmo tomando en cuenta sus operaciones básicas.

9. ¿Los resultados experimentales de la implementación con hilos de los algoritmos realmente tardaron F(t)/#hilos de su implementación sin hilos?
   No, principalmente en los de Búsqueda Binaria y Árbol Binario de Búsqueda en los cuáles la implementación con hilos prácticamente no implica una mejora.
   En cuanto a la Búsqueda Lineal que es en la que sí encontramos una variación en los resultados, pero no se cumple que sea F(t)/#hilos.

10. ¿Cuál es el porcentaje de mejora que tiene cada uno de los algoritmos en su variante con hilos? ¿Es lo que esperabas? ¿Por qué?
    En los algoritmos de Búsqueda Binaria y de Árbol Binario de Búsqueda no hay mejora alguna, por el contrario, el tiempo incrementa un poco debido a que tiene que esperar a que los hilos se creen y se unan.
    En la Búsqueda Binara sí hay una mejora en las búsquedas para arreglos de mayor longitud. Al principio, de hecho tarda más el algoritmo con hilos, pero conforme el tamaño del arreglo crece, se puede encontrar una mejora de hasta 32% en algunos casos.

11. ¿Existió un entorno controlado para realizar las pruebas experimentales? ¿Cuál fue?

Sí, todas las pruebas se realizaron en una misma computadora, teniendo especial cuidado de no contar operaciones fuera de las de la búsqueda y de no ejecutar otras tareas que pudieran interferir con un registro correcto del tiempo.

12. ¿Si solo se realizara el análisis teórico de un algoritmo antes de implementarlo, podrías asegurar cuál es el mejor?
    Siendo un análisis bien realizado, por supuesto que se puede asegurar cuál es mejor, sólo es de conocer o ver cómo se comporta cada una de las funciones que definen la complejidad de los algoritmos.

13. ¿Qué tan difícil fue realizar el análisis teórico de cada algoritmo?
    Obtener las funciones para el mejor y el peor caso resulta sencillo, pero la del caso se complica un poco al definir correctamente la función complejidad y realizar las operaciones para adaptarla a la sumatoria.

14. ¿Qué recomendaciones darían a nuevos equipos para realizar esta práctica?
    El uso de scripts para registrar los resultados de cada algoritmo y posteriormente utilizar un software como Excel que facilite el manejo de estos datos y su graficación.
    Utilizar una misma computadora para realizar todas las pruebas y cuidar que no haya otras tareas ejecutándose que puedan alterar los resultados.
    Realizar juntos el análisis teórico para así corregirse mutuamente y encontrar más cosas a tomar en cuenta para llegar a la solución correcta de cada algoritmo.

Códigos Comentados

Búsqueda Lineal Sin Hilos

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include "tiempo.h"
5. 
6. /*
7.     Búsqueda Lineal
8.     Versión 1.0
9.     Autores: Barriga Vargas Martín Eduardo, Ramírez Vives José
   Manuel, Salinas Núñez Jaime Alejandro
10.     */
11. 
12. 
13.     //Compilación: "gcc main.c tiempo.x  -o main(teimpo.c si se
   tiene la implementación de la libreria o tiempo.o si solo se tiene
   el codigo objeto)"
14.     //Ejecución: "./main n" (Linux y MAC OS)
15.     //************************************************************
   *****
16. 
17. 
18.     int buscarNumero(int *ar, int tam, int numBuscado){ //Se
   encarga de buscar el número, recibe el arreglo, su tamaño y el
   numero a buscar, devuelve un entero indicando si se encontró o no el
   número
19.         for(int i = 0; i < tam; i++){//Se recorre el arreglo
20.             if(ar[i] == numBuscado) return i; //Si el número del
   arreglo corresponde con el número buscado se regresa un número
   indicando la posición del arreglo
21.         }
22.         return -1;//Si se llega hasta esta línea, se regresa un -1
   indicando que no se encontró el número
23.     }
24. 
25.     //************************************************************
   *****
26.     //PROGRAMA PRINCIPAL
27.     //************************************************************
   *****
28.     int main (int argc, char* argv[])
29.     {
30.         double utime0, stime0, wtime0,utime1, stime1, wtime1; //Va
   riables para medición de tiempos
31. 
32.         //********************************************************
   **********
33.         //Algoritmo
```

```
34.         //*********************************************************
   **********
35.         FILE *Numeros;
36.         Numeros = fopen("numeros10millones.txt", "r");
37.         int i,tamano, *ar, resultado,num,numBusq;
38.
39.         tamano=atoi(argv[1]); //lee el tamaño
40.         numBusq = atoi(argv[2]);//lee el número a buscar
41.
42.
43.         ar = (int *) malloc(sizeof(int)*tamano); //crea el arreglo
   dinámico
44.         for(i=0; i < tamano; i++){//Recorre el arreglo
45.             fscanf(Numeros,"%d",&num);   //lee el numero desde el
   archivo de texto y lo guarda en el arreglo
46.             ar[i] = num;
47.         }
48.         fclose(Numeros);
49.
50.         uswtime(&utime0, &stime0, &wtime0);//Comienza a medir
   tiempos
51.         resultado = buscarNumero(ar, tamano, numBusq); //Llama a
   la función de búsqueda, mandando como parámetros el arreglo, su
   tamaño y el número a buscar, recibe un entero indicando si se
   encontró o no el número
52.         uswtime(&utime1, &stime1, &wtime1);//Finaliza la medición
   de tiempos
53.
54.         //Cálculo del tiempo de ejecución del programa
55.         printf("%.10f\n",  ((wtime1 - wtime0))*10000);
56.         //printf("%.10f\n",  utime1 - utime0);
57.         /*if(resultado>=0){
58.             printf("Si\n");
59.         }
60.         else{
61.             printf("No\n");
62.         }*/
63.         //printf("%.10f s\n",  stime1 - stime0);
64.         //printf("%.10f %% \n",100.0 * (utime1 - utime0 + stime1 -
   stime0) / (wtime1 - wtime0));
65.         //printf("\n");
66.
67.         //*********************************************************
   **********
68.         //Terminar programa normalmente
69.         exit (0);
70.
71.     }
```

Búsqueda Lineal Con Hilos

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

/*
    Búsqueda Lineal Con Hilos
    Lee de un archivo de texto una cantidad de números desordenados, los guarda en un arreglo y procede a realizar una búsqueda utilizando hilos, indicándole a cada hilo con ayuda de estructuras qué porción del arreglo buscará
          04/04/2019
          Versión 1.0
          Autores: Barriga Vargas Martín Eduardo, Ramírez Vives José Manuel, Salinas Núñez Jaime Alejandro
     */


     //Compilación: "gcc main.c tiempo.x  -o main(teimpo.c si se tiene la implementación de la libreria o tiempo.o si solo se tiene el codigo objeto)"
     //Ejecución: "./main n" (Linux y MAC OS)
     //*************************************************************
*****

     struct Parametros{ //Estructura elaborada para guardar el contenido de los datos necesarios para hacer la búsqueda del número, los cuales son: El inicio de donde debemos empezar a buscar, el final, el número buscado, el arreglo en el que se encuentran los números y una variable que representa si ya se encontró el dato
          int inicio;
          int fin;
          int numBuscado;
          int *ar;
          int encontrado;
     };


     void buscarNumero(struct Parametros *parametros){ //Se encarga de buscar el número en un rango, recibe una estructura tipo parámetro y no regresa nada
          int i,tam = parametros->fin; //se le asigna a la variable tam el valor fin, que dice hasta donde debe de dejar de buscar
          for(i = parametros->inicio; i < tam; i++){ //Hacemos un recorrido del valor inicio que viene en parámetros hasta tam
               if(parametros->ar[i] == parametros->numBuscado) { //Si encontramos el número buscado en el arreglo entonces modificamos el valor de encontrado y rompemos el ciclo
```

```
33.                     parametros->encontrado = 1;
34.                     break;
35.                 }
36.             }
37.             //en caso de que no se haya encontrado el número, el valor encontrado se queda en 0
38.         }
39.
40.         //*************************************************************
41.         //PROGRAMA PRINCIPAL
42.         //*************************************************************
43.         int main (int argc, char* argv[])
44.         {
45.             double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para medición de tiempos
46.
47.             //*************************************************************
48.             //Algoritmo
49.             //*************************************************************
50.             FILE *Numeros;
51.             pthread_t Hilo1,Hilo2; //Creamos nuestros hilos
52.             Numeros = fopen("numeros10millones.txt", "r");
53.             int i,tamano, *ar, resultado,num,numBusq;
54.
55.             tamano=atoi(argv[1]); //leemos el tamaño
56.             numBusq = atoi(argv[2]);//leemos el número a buscar
57.
58.
59.             ar = (int *) malloc(sizeof(int)*tamano);//Creamos un arreglo dinámico en el que vamos a guardar los números
60.             for(i=0; i < tamano; i++){ //Se recorre el archivo e introducimos los números en el arreglo
61.                 fscanf(Numeros,"%d",&num);
62.                 ar[i] = num;
63.             }
64.             fclose(Numeros);
65.
66.             struct Parametros param1 = {0,tamano/2,numBusq,ar,0};//Creamos una instancia tipo parámetros en el que indicamos que el punto de inicio es 0, el fin de búsqueda es el tamaño entre dos, le indicamos el número a buscar y le asignamos el valor 0 a la variable "encontrado"
67.             struct Parametros param2 = {tamano/2,tamano,numBusq,ar,0};//Creamos una instancia tipo parámetros en el que indicamos que el punto de inicio es tamaño entre dos, el fin de búsqueda es el tamaño, le indicamos el número a buscar y le asignamos el valor 0 a la variable "encontrado"
68.
```

```
69.          uswtime(&utime0, &stime0, &wtime0); //Comenzamos a medir
   el tiempo
70.              pthread_create(&Hilo1,NULL, (void*)buscarNumero,(void*
   )&param1); //creamos el hilo1, indicándole la instancia, null, la
   función que ejecutará y la estructura que le enviará a dicha función
71.              pthread_create(&Hilo2,NULL, (void*)buscarNumero,(void*
   )&param2); //creamos el hilo2, indicándole la instancia, null, la
   función que ejecutará y la estructura que le enviará a dicha función
72.              pthread_join(Hilo1, NULL);//Ejecutamos el hilo1
73.              pthread_join(Hilo2, NULL);//Ejecutamos el hilo2
74.          uswtime(&utime1, &stime1, &wtime1);//Finalizamos la
   medición de tiempo
75.
76.          //Cálculo del tiempo de ejecución del programa
77.          printf("%.10f\n",  ((wtime1 - wtime0))*10000);
78.          //printf("%.10f\n",  utime1 - utime0);
79.          /*if(param1.encontrado == 1 || param2.encontrado==1){
80.              printf("Si\n");
81.          }
82.          else{
83.              printf("No\n");
84.          }*/
85.          //printf("%.10f s\n",  stime1 - stime0);
86.          //printf("%.10f %% \n",100.0 * (utime1 - utime0 + stime1 -
   stime0) / (wtime1 - wtime0));
87.          //printf("\n");
88.
89.          //**********************************************************
   **********
90.          //Terminar programa normalmente
91.          exit (0);
92.
93.      }
```

Búsqueda en Árbol Binario Sin Hilos

```
1. /*
2.     Búsqueda con Árbol Binario
3.     Lee de un archivo de texto una cantidad de úmeros desordenados y
   procede a elaborar el árbol, donde el hijo izquierdo siempre será
   menor que el padre y el hijo derecho será mayor que el padre..
4.     Después se toma el tiempo que tarda en buscar un número deseado
   en los nodos del árbol.
5.     04/04/2019
6.     Versión 1.0
7.     Autores: Barriga Vargas Martín Eduardo, Ramírez Vives José
   Manuel, Salinas Núñez Jaime Alejandro
8. */
9.
10.
11.     //Compilación: "gcc main.c tiempo.x  -o main(teimpo.c si se
   tiene la implementación de la libreria o tiempo.o si solo se tiene
   el codigo objeto)"
12.     //Ejecución: "./main n" (Linux y MAC OS)
13.     //**********************************************************
   *****
14.
15.     #include <stdio.h>
16.     #include <stdlib.h>
17.     #include <string.h>
18.     #include "tiempo.h"
19.     #include <iostream>
20.     using namespace std;
21.
22.     struct nodo { //Estructura encargada de almacenar un valor
   entero, un nodo izquierdo y un nodo derecho, los cuales son del
   mismo tipo que la estructura
23.         int dato;
24.         nodo* izq;
25.         nodo* der;
26.     };
27.
28.     nodo* crearNodo(int dato) { //Recibe el dato que se va a
   introducir en el nodo que está por crearse, devuelve un valor tipo
   nodo (el nodo que ha sido creado)
29.         nodo* nuevoNodo = new nodo(); // se crea una instancia
30.         nuevoNodo->dato = dato; // se le asigna al nodo el
   parametro recibido
31.         nuevoNodo->izq = nuevoNodo->der = NULL; //tanto a los
   nodos izquierdo y derecho se les asigna null, pues actualmente no
   sabemos si deben crearse
32.         return nuevoNodo; //regresamos el nodo creado
33.     }
34.
```

```
35.      nodo* insertar(nodo* raiz,int dato) { //Recibimos el nodo raiz
   y el dato que le insertaremos, duevelve el nodo raiz
36.          if(raiz == NULL) {  // si el nodo recibido es nulo
   entonces significa que estamos en un nodo izquierdo o derecho
37.              raiz = crearNodo(dato); //por lo tanto creamos un nodo
   y se lo asignamos al nodo actual
38.          }
39.          else if(dato <= raiz->dato) { //si el dato es menor o
   igual al valor del nodo en el que estamos parados entonces nos
   dirijimos a su hijo izquierdo
40.              raiz->izq = insertar(raiz->izq,dato); // llamamos
   recursivamente a la función para seguir buscando, y nos regresará la
   parte izquierda del nodo ya editado
41.          }
42.          else { //si el dato es mayor entonces pertenece a la rama
   derecha del nodo en el que nos encontramos
43.              raiz->der = insertar(raiz->der,dato);// llamamos
   recursivamente a la función para seguir buscando, y nos regresará la
   parte derecha del nodo ya editado
44.          }
45.          return raiz; //regresamos el nodo editado
46.      }
47.
48.      nodo* recorridoInsertar(int *arES, int tamano, nodo *raiz){ //
   Recibimos el nodo raiz, un arreglo y el tamaño del arreglo,
   devolvemos un nodo, el cuál es la raiz del arbol
49.          int i, indice =0;
50.          for(i = 0; i < tamano; i++)  //recorremos el arreglo
51.          {
52.              raiz = insertar(raiz, arES[i]); //por cada elemento en
   el arreglo mandamos a llamar la función insertar con el valor que
   tiene el arreglo en el indice indicado y la raiz del árbol,
   guardamos la raiz del árbol con el nodo ya añadido
53.          }
54.          return raiz; //regresamos la raiz del árbol
55.      }
56.
57.      int busqueda(nodo *raiz, int numBusq){ //Recibimos el nodo
   raiz de nuestro árbol y nuestro número a buscar, regresamos un
   entero indicando si se encontró o no el número
58.          if(raiz == NULL) return 0;  //Si el nodo que acabo de
   recibir es nulo, entonces significa que el número no está y
   regresamos un 0. Si el número estuviera ya hubiera aparecido antes
59.          if(raiz->dato == numBusq) return 1; //Si el dato del nodo
   en el que estoy parado es igual al que estoy buscando entonces
   regresamos un 1, indicando que sí se encuentra
60.          if(numBusq > raiz->dato) {//Si el número buscando es más
   grande que el valor del nodo en el que estoy parado entonces me
   dirijo a su nodo derecho
61.              return busqueda(raiz->der, numBusq); //regreso el
   valor que obtenga de haber buscado en el nodo derecho
62.          }
63.          else {
```

```
64.             return busqueda(raiz->izq,numBusq);//regreso el valor
    que obtenga de haber buscado en el nodo izquierdo
65.         }
66.     }
67.
68.
69.     //************************************************************
    *****
70.     //PROGRAMA PRINCIPAL
71.     //************************************************************
    *****
72.     int main (int argc, char* argv[])
73.     {
74.         double utime0, stime0, wtime0,utime1, stime1, wtime1; //Va
    riables para medición de tiempos
75.
76.         //************************************************************
    **********
77.         //Algoritmo
78.         //************************************************************
    **********
79.         FILE *Numeros;
80.         Numeros = fopen("numeros10millones.txt", "r");
81.
82.         int tamano,num, *arES,res,numBusq;
83.
84.         tamano=atoi(argv[1]); //leemos el tamaño
85.         numBusq = atoi(argv[2]);//leemos el numero que vamos a
    buscar
86.         arES = (int *) malloc(sizeof(int)*tamano); //creamos un
    arreglo dinámico
87.
88.         nodo* raiz = NULL; //creamos un nodo raiz con un valor
    nulo
89.
90.         for(int i=0; i< tamano; i++) // leemos todos los números
    del archivo de texto y se guardan en un arreglo
91.         {
92.             fscanf(Numeros,"%d",&num);
93.             //cin >> num;
94.             arES[i] = num;
95.
96.         }
97.         fclose(Numeros);
98.
99.         raiz = recorridoInsertar(arES, tamano, raiz); //llamamos a
    una función para insertar nuestros valores del arreglo en nuestra
    raíz, mandando la raiz, el arreglo y su tamaño
100.        uswtime(&utime0, &stime0, &wtime0);
101.        res =  busqueda(raiz, numBusq); //mandamos a llamar a la
    función búsqueda con la raiz de nuestro árbol y el número a buscar.
    Guardamos el valor que regresa la función, representando si se
    encuentra o no el valor en el árbol
```

```
102.         uswtime(&utime1, &stime1, &wtime1);
103.
104.
105.         //Cálculo del tiempo de ejecución del programa
106.         printf("%.10f\n",  ((wtime1 - wtime0))*10000);
107.         //printf("%.10f\n",  utime1 - utime0);
108.         //if(res == 0) cout << "No" << endl;
109.         //else cout << "Si" << endl;
110.         //printf("%.10f s\n",  stime1 - stime0);
111.         //printf("%.10f %% \n",100.0 * (utime1 - utime0 + stime1 -
   stime0) / (wtime1 - wtime0));
112.         //printf("\n");
113.
114.         //*********************************************************
   **********
115.         //Terminar programa normalmente
116.         exit (0);
117.     }
```

Búsqueda en Árbol Binario Con Hilos

```
1. /*
2.     Búsqueda con Árbol Binario Con Hilos
3.     Lee de un archivo de texto una cantidad de úmeros desordenados y
   procede a elaborar el árbol, donde el hijo izquierdo siempre será
   menor que el padre y el hijo derecho será mayor que el padre..
4.     Después se toma el tiempo que tarda en buscar un número deseado
   en los nodos del árbol utilizando dos hilos, uno que se encarga de
   analizar lo que está del lado izquierdo del árbol y otro que se
   encarga de analizar todo el lado derecho del árbol
5.     Versión 1.0
6.     Autores: Barriga Vargas Martín Eduardo, Ramírez Vives José
   Manuel, Salinas Núñez Jaime Alejandro
7. */
8.
9.
10.     //Compilación: "gcc main.c tiempo.x  -o main(teimpo.c si se
   tiene la implementación de la libreria o tiempo.o si solo se tiene
   el codigo objeto)"
11.     //Ejecución: "./main n" (Linux y MAC OS)
12.     //**************************************************************
   *****
13.
14.     #include <stdio.h>
15.     #include <stdlib.h>
16.     #include <string.h>
17.
18.     #include "tiempo.h"
19.     #include <pthread.h>
20.
21.     struct nodo {  //Estructura encargada de almacenar un valor
   entero, un nodo izquierdo y un nodo derecho, los cuales son del
   mismo tipo que la estructura
22.         int dato;
23.         struct nodo* izq;
24.         struct nodo* der;
25.     };
26.     struct Parametros{ //Estructura encargada de guardar la raiz
   de nuestro árbol, el número a bucar y un indicador de si se encontró
   o no el número
27.         struct nodo* raiz;
28.         int numBusq;
29.         int encontrado;
30.     };
31.
32.     struct nodo* crearNodo(int dato) {//Recibe el dato que se va a
   introducir en el nodo que está por crearse, devuelve un valor tipo
   nodo (el nodo que ha sido creado)
33.         struct nodo* nuevoNodo = malloc(sizeof(struct nodo)); //
   se crea una instancia
```

```
34.          nuevoNodo->dato = dato;// se le asigna al nodo el
   parametro recibido
35.          nuevoNodo->izq = nuevoNodo->der = NULL; //tanto a los
   nodos izquierdo y derecho se les asigna null, pues actualmente no
   sabemos si deben crearse
36.          return nuevoNodo;//regresamos el nodo creado
37.      }
38.
39.      struct nodo* insertar(struct nodo* raiz,int dato) {//Recibimos
   el nodo raiz y el dato que le insertaremos, duevelve el nodo raiz
40.          if(raiz == NULL) { // si el nodo recibido es nulo entonces
   significa que estamos en un nodo izquierdo o derecho
41.              raiz = crearNodo(dato);//por lo tanto creamos un nodo
   y se lo asignamos al nodo actual
42.          }
43.          else if(dato <= raiz->dato) { //si el dato es menor o
   igual al valor del nodo en el que estamos parados entonces nos
   dirijimos a su hijo izquierdo
44.              raiz->izq = insertar(raiz->izq,dato); // llamamos
   recursivamente a la función para seguir buscando, y nos regresará la
   parte izquierda del nodo ya editado
45.          }
46.          else {//si el dato es mayor entonces pertenece a la rama
   derecha del nodo en el que nos encontramos
47.              raiz->der = insertar(raiz->der,dato); // llamamos
   recursivamente a la función para seguir buscando, y nos regresará la
   parte derecha del nodo ya editado
48.          }
49.          return raiz;//regresamos el nodo editado
50.      }
51.
52.      struct nodo* recorridoInsertar(int *arES, int tamano, struct n
   odo *raiz){ //Recibimos el nodo raiz, un arreglo y el tamaño del
   arreglo, devolvemos un nodo, el cuál es la raiz del arbol
53.          int i, indice =0;
54.          struct nodo *nuevoNodo = NULL;//recorremos el arreglo
55.          for(i = 0; i < tamano; i++)
56.          {
57.              raiz = insertar(raiz, arES[i]);//por cada elemento en
   el arreglo mandamos a llamar la función insertar con el valor que
   tiene el arreglo en el indice indicado y la raiz del árbol,
   guardamos la raiz del árbol con el nodo ya añadido
58.          }
59.          return raiz; //regresamos la raiz del árbol
60.      }
61.
62.
63.      void busqueda( struct Parametros *parametros){ //Se encarga de
   buscar el número y recibe como parámetro una estructura tipo
   parámetros con toda la información dentro
64.
```

```
65.          if(parametros->raiz == NULL){ //Si el nodo que acabo de
   recibir es nulo, entonces significa que el número no está y
   regresamos un 0. Si el número estuviera ya hubiera aparecido antes
66.
67.          }
68.          else if(parametros->raiz->dato == parametros-
   >numBusq){ //Si el dato del nodo actual es igual al valor que
   estamos buscando entonces actualizamos el valor de "encontrado" a 1
69.              parametros->encontrado = 1;
70.          }
71.          else if(parametros->numBusq > parametros->raiz-
   >dato) {//En caso de que el número buscado sea mayor que el dato que
   guarda el nodo en el que estamos parados le asignamos al nodo el
   valor de nuestra raiz derecha y mandamos a buscar nuevamente
72.              parametros->raiz = parametros->raiz->der;
73.              busqueda(parametros);
74.          }
75.          else if(parametros->numBusq < parametros->raiz->dato){//En
   caso de que el número buscado sea menor que el dato que guarda el
   nodo en el que estamos parados le asignamos al nodo el valor de
   nuestra raiz izquierda y mandamos a buscar nuevamente
76.              parametros->raiz = parametros->raiz->izq;
77.              busqueda(parametros);
78.          }
79.      }
80.
81.
82.      //*************************************************************
   *****
83.      //PROGRAMA PRINCIPAL
84.      //*************************************************************
   *****
85.      int main (int argc, char* argv[])
86.      {
87.          double utime0, stime0, wtime0,utime1, stime1, wtime1; //Va
   riables para medición de tiempos
88.
89.          //*********************************************************
   **********
90.          //Algoritmo
91.          //*********************************************************
   **********
92.          pthread_t Hilo1,Hilo2; //Declaramos los hilos
93.          FILE *Numeros;
94.
95.          Numeros = fopen("numeros10millones.txt", "r");
96.
97.          int tamano,num, *arES,res,numBusq;
98.
99.          tamano=atoi(argv[1]);//leemos el tamaño
100.         numBusq = atoi(argv[2]);//leemos el numero que vamos a
   buscar
```

```
101.        arES = (int *) malloc(sizeof(int)*tamano);//creamos un
    arreglo dinámico
102.
103.        struct nodo* raiz = NULL;//creamos un nodo raiz con un
    valor nulo
104.
105.        for(int i=0; i< tamano; i++) // leemos todos los números
    del archivo de texto y se guardan en un arreglo
106.        {
107.            fscanf(Numeros,"%d",&num);
108.            //cin >> num;
109.            arES[i] = num;
110.
111.        }
112.        fclose(Numeros);
113.        raiz = recorridoInsertar(arES, tamano, raiz);  //llamamos
    a una función para insertar nuestros valores del arreglo en nuestra
    raíz, mandando la raiz, el arreglo y su tamaño
114.        struct Parametros param1 = {raiz-
    >izq, numBusq,0}; //Instanciamos una estructura de tipo Parametros,
    le mandamos el nodo iquierdo del árbol, el numero a buscar y un
    valor 0 indicando que aún no se ha encontrado el número
115.        struct Parametros param2= {raiz-
    >der, numBusq,0};//Instanciamos una estructura de tipo Parametros,
    le mandamos el nodo derecho del árbol, el numero a buscar y un valor
    0 indicando que aún no se ha encontrado el número
116.        uswtime(&utime0, &stime0, &wtime0);
117.        if(raiz->dato != numBusq){ //En caso de que el número que
    nos piden buscar no sea la raiz utiliamos un hilo para buscar del
    lado del nodo izquierdo del árbol y otro hilo para bucar del lado
    derecho
118.            pthread_create(&Hilo1,NULL, (void*)busqueda, (void*)&p
    aram1);//Creamos los hilos, mandando como parámetros la función y
    sus estructuras parámetros correspondientes
119.            pthread_create(&Hilo2,NULL, (void*)busqueda, (void*)&p
    aram2);
120.            pthread_join(Hilo1, NULL);//Ejecutamos los hilos
121.            pthread_join(Hilo2, NULL);
122.        }
123.        uswtime(&utime1, &stime1, &wtime1);
124.
125.
126.        //Cálculo del tiempo de ejecución del programa
127.        printf("%.10f\n",  ((wtime1 - wtime0))*10000);
128.        /*printf("%.10f\n",  utime1 - utime0);
129.        if(res == 0) cout << "No" << endl;
130.        else cout << "Si" << endl;
131.        printf("%.10f s\n",  stime1 - stime0);
132.        printf("%.10f %% \n",100.0 * (utime1 - utime0 + stime1 -
    stime0) / (wtime1 - wtime0));
133.        printf("\n");*/
134.
```

```
135.         //**********************************************************
   **********
136.         //Terminar programa normalmente
137.         exit (0);
138.     }
```

Búsqueda Binaria Sin Hilos

```
1. /*
2.     Búsqueda Binaria
3.     Algoritmo que busca un numero específico en un archivo de 10
   millones de numeros diferentes ordenados aleatoriamente
4.     Versión 1.0
5.     Autores: Barriga Vargas Martín Eduardo, Ramírez Vives José
   Manuel, Salinas Núñez Jaime Alejandro
6.
7.     Compilación: "gcc main.c tiempo.x  -o main(teimpo.c si se tiene
   la implementación de la libreria o tiempo.o si solo se tiene el
   codigo objeto)"
8.     Ejecución: "./main n NúmeroaBuscar" (Linux y MAC OS)
9. */
10.
11.
12.      /**** Librerias para el funcionamiento del programa ****/
13.      #include <stdio.h>
14.      #include <stdlib.h>
15.      #include <string.h>
16.      #include "tiempo.h"
17.
18.
19.      /**** Prototipo de función de busqueda binaria ****/
20.      /* Recibe un arreglo de enteros, un tamaño de problema n y el
   numero a buscar */
21.      void BusquedaBinaria(int *, int, int);
22.
23.      double utime0=0, stime0=0, wtime0=0,utime1=0, stime1=0, wtime1
   =0;//Declaración de variables para medición de tiempo
24.      int mitad, encontrado=0;
25.
26.
27.      /************************/
28.      /** PROGRAMA PRINCIPAL **/
29.      /************************/
30.
31.      int main (int argc, char* argv[])
32.      {
33.
34.          int n,i, numero; //Variables para tamaño de problema,
   ciclo para leer datos, y numero a buscar
35.          int *arr;
36.
37.          n = atoi(argv[1]);
38.          numero = atoi(argv[2]);
39.
40.          arr = (int *) malloc(sizeof(int)*n);
41.
```

```
42.          //Se lee y obtienen los numeros del archivo texto con 10
   millones de números
43.          FILE *Numeros;
44.          Numeros = fopen("10MOrdenados.txt", "r");
45.
46.          for (i = 0; i < n; i++)
47.            fscanf(Numeros,"%d",&arr[i]);
48.          fclose(Numeros);
49.
50.
51.          BusquedaBinaria(arr,numero,n); //Se manda a llamar la
   función de ordenamiento binaria
52.
53.          printf("\n");
54.          printf("%d\n",n);
55.          printf("%d\n",numero);
56.          printf("%d\n",mitad+1);
57.          if(encontrado==0)
58.              printf("No\n");
59.          else
60.              printf("Si\n");
61.          printf("Tiempo real: %.10f\n",wtime1 - wtime0);
62.          printf("Tiempo CPU: %.5f\n",   (utime1 - utime0)*100000);
63.
64.          //Terminar programa normalmente
65.          exit (0);
66.      }
67.
68.
69.      /** Función de busqueda binaria **/
70.      /** La función recibe un arreglo de numeros enteros, un tamaño
   de problema n y un numero a buscar **/
71.      /** No posee un dato de retorno **/
72.      /*****************************************************************
   ******************************/
73.      /** El algoritmo se encarga de ir diviendo en mitades el
   tamaño del arreglo y comparando el **/
74.      /** número que queda en la mitad del arreglo con el numero que
   estamos buscando, en caso de **/
75.      /** se encuentre, cambia la bandera encontrado, en caso de no
   encontrarlo, seguirá diviendo **/
76.      /** entre dos, creando partes más pequeñas de arreglo en las
   cuales irá buscando el número  **/
77.      /*****************************************************************
   ******************************/
78.
79.      void BusquedaBinaria(int *A, int num, int n){
80.          int superior, inferior;
81.
82.          inferior = 0;
83.          superior = n;
84.
```

```
85.      uswtime(&utime0, &stime0, &wtime0); //Empezamos a medir
   tiempos de busqueda
86.          while((inferior <= superior) && (encontrado == 0))
87.          {
88.              mitad =(inferior+superior)/2;
89.
90.              if(A[mitad]==num){
91.                  encontrado = 1;
92.              }
93.              else if(A[mitad]>num){
94.                  superior = mitad - 1;
95.              }
96.              else{
97.                  inferior = mitad + 1;
98.              }
99.          }
100.         uswtime(&utime1, &stime1, &wtime1); //Terminamos de medir
   tiempos de busqueda
101.     }
```

Búsqueda Binaria Con Hilos

```
1. /*
2.     Búsqueda Binaria
3.     Algoritmo que busca un numero específico en un archivo de 10
   millones de numeros diferentes ordenados aleatoriamente
4.     Versión 1.0
5.     Autores: Barriga Vargas Martín Eduardo, Ramírez Vives José
   Manuel, Salinas Núñez Jaime Alejandro
6.
7.     Compilación: "gcc main.c tiempo.x  -o main(teimpo.c si se tiene
   la implementación de la libreria o tiempo.o si solo se tiene el
   codigo objeto)"
8.     Ejecución: "./main n NúmeroaBuscar" (Linux y MAC OS)
9. */
10.
11.
12.      /**** Librerias para el funcionamiento del programa ****/
13.      #include <stdio.h>
14.      #include <stdlib.h>
15.      #include <string.h>
16.      #include "tiempo.h"
17.      #include <pthread.h>
18.
19.      /** Definición de estructura para apoyo en el pase de
   parametros a las funcion por medio de hilos **/
20.      struct Parametros{
21.          int ni;
22.          int ns;
23.          int num;
24.          int *A;
25.      };
26.
27.      double utime0=0, stime0=0, wtime0=0,utime1=0, stime1=0, wtime1
   =0;//Declaración de variables para medición de tiempo
28.      int encontrado=0;
29.
30.      /** Función de busqueda binaria adaptada para el su uso con
   hilos **/
31.      /** La función recibe una estructura de tipo Parametros la
   cual contiene un arreglo de numeros enteros, un tamaño de problema n
   y un numero a buscar **/
32.      /** No posee un dato de retorno **/
33.      /*****************************************************************
   *******************************/
34.      /** El algoritmo se encarga de ir diviendo en mitades el
   tamaño del arreglo y comparando el **/
35.      /** número que queda en la mitad del arreglo con el numero que
   estamos buscando, en caso de **/
36.      /** se encuentre, cambia la bandera encontrado, en caso de no
   encontrarlo, seguirá diviendo **/
```

```
37.      /** entre dos, creando partes más pequeñas de arreglo en las
   cuales irá buscando el número  **/
38.      /*****************************************************************
   ******************************/
39.      void BusquedaBinariaHilos(struct Parametros *parametros){
40.          int superior, inferior;
41.          int mitad;
42.
43.          inferior = parametros->ni;
44.          superior = parametros->ns;
45.
46.              while((inferior <= superior) && (encontrado == 0))
47.              {
48.                  mitad =(inferior+superior)/2;
49.
50.                  if(parametros->A[mitad]==parametros->num){
51.                      encontrado = 1;
52.                  }
53.                  else if(parametros->A[mitad]>parametros->num){
54.                      superior = mitad - 1;
55.                  }
56.                  else{
57.                      inferior = mitad + 1;
58.                  }
59.              }
60.
61.      }
62.
63.      /************************/
64.      /** PROGRAMA PRINCIPAL **/
65.      /************************/
66.      int main (int argc, char* argv[])
67.      {
68.
69.          int n, i, numero, *arr;  //Variables para tamaño de
   problema, ciclo para leer datos, y numero a buscar
70.          pthread_t Hilo1,Hilo2,Hilo3; //Variables de tipo Pthread
   con las cuales declaramos nuestros hilos a utilizar
71.
72.          n = atoi(argv[1]);
73.          numero = atoi(argv[2]);
74.          arr = (int *) malloc(sizeof(int)*n);
75.
76.          //Se lee y obtienen los numeros del archivo texto con 10
   millones de números
77.          FILE *Numeros;
78.          Numeros = fopen("10MOrdenados.txt", "r");
79.
80.          for (i = 0; i < n; i++)
81.            fscanf(Numeros,"%d",&arr[i]);
82.          fclose(Numeros);
83.
```

```
84.         // Se definen las estructuras con los parametros que
   pasaremos a nuestra función de busqueda //
85.         // Se dividió en tres el arreglo inicial para que cada uno
   de los hilos tenga un tercio del //
86.         // arerglo corriendo.
87.         struct Parametros param1 = {n*(2/3),n,numero,arr};
88.         struct Parametros param2 = {n*(1/3),n*(2/3),numero,arr};
89.         struct Parametros param3 = {0,n*(1/3),numero,arr};
90.
91.         uswtime(&utime0, &stime0, &wtime0); //Empezamos la
   medición de tiempo de busqueda
92.
93.         pthread_create(&Hilo1,NULL, (void*)BusquedaBinariaHilos,(v
   oid*)&param1); //Se manda a correr el hilo uno con la ultima parte
   del arreglo
94.         pthread_create(&Hilo2,NULL, (void*)BusquedaBinariaHilos,(v
   oid*)&param2); //Se manada a correr el hilo dos con la segunda
   tercia del arreglo
95.         pthread_create(&Hilo3,NULL, (void*)BusquedaBinariaHilos,(v
   oid*)&param3); //Se manda a correr el hilo con el primer tercio del
   arreglo
96.
97.         pthread_join(Hilo1, NULL); // Se espera que termine su
   proceso el hilo 1
98.         pthread_join(Hilo2, NULL); // Se espera que termine su
   proceso el hilo 2
99.         pthread_join(Hilo3, NULL); // Se espera que termine su
   proceso el hilo 3
100.
101.        uswtime(&utime1, &stime1, &wtime1); //Terminamos la
   medición de los tiempos de busqueda
102.
103.        //Cálculo del tiempo de ejecución del programa
104.        printf("\n");
105.        printf("%d\n",n);
106.        printf("%d\n",numero);
107.        printf("%d\n",mitad+1);
108.        if(encontrado==0)
109.            printf("No\n");
110.        else
111.            printf("Si\n");
112.
113.        printf("Tiempo real:%.10f\n",wtime1 - wtime0);
114.        printf("Tiempo CPU%.5f\n", utime1 - utime0);
115.
116.        //Terminar programa normalmente
117.        exit (0);
118.    }
```