Análisis de Algoritmos

Práctica 02: Análisis temporal

Nombres:

Martínez Partida Jair Fabian

Martínez Rodríguez Alejandro

Monteros Cervantes Miguel Angel

Luis Fernando Ramírez Cotonieto

Fecha de entrega:29 de Abril del 2021

Grupo:3CM13

INSTITUTO POLITÉCNICO NACIONAL

ESCOM

Índice

Práctica 02: Análisis Temporal

Análisis de Algoritmos

1. Planteamiento del problema

Con base en el archivo de entrada de la practica 01 que tiene 10,000,000 de números diferentes. Realizar la búsqueda de elementos bajo 5 métodos, realizar el análisis temporal a priori (análisis de casos) y experimental de las complejidades.

- Búsqueda lineal o secuencial
- Búsqueda en un árbol binario de búsqueda
- Búsqueda binaria o dicotómica
- Búsqueda exponencial
- Búsqueda de Fibonacci

Finalmente realizar la adaptación de los cinco métodos de búsqueda empleando hilos (Threads) de manera que se busque mejorar los tiempos de búsqueda de cada método.

2. Actividades y Pruebas

2.1. Busqueda lineal

Un enfoque simple es hacer una búsqueda lineal, es decir:

- Comience desde el elemento más a la izquierda de arr[] y uno por uno compare x con cada elemento de arr[].
- Si x coincide con un elemento, devuelve el índice.
- Si x no coincide con ninguno de los elementos, devuelva -1.

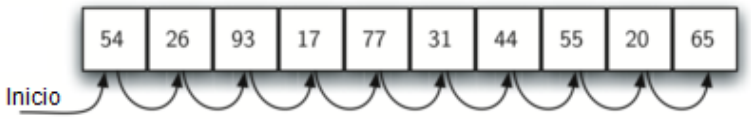

Figura 1: Ejemplo de búsqueda secuencial en una lista de enteros

2.2. Árbol de búsqueda binario

Binary Search Tree es una estructura de datos de árbol binario basada en nodos que tiene las siguientes propiedades

- El subárbol izquierdo de un nodo contiene solo nodos con claves menores que la clave del nodo.
- El subárbol derecho de un nodo contiene solo nodos con claves mayores que la clave del nodo.
- El subárbol izquierdo y derecho también debe ser un árbol de búsqueda binario.
- No debe haber nodos duplicados.

Las propiedades anteriores de Binary Search Tree proporcionan un orden entre claves para que las operaciones como búsqueda, mínimo y máximo se puedan hacer rápidamente. Si no hay orden, entonces podemos tener que comparar cada clave para buscar una clave dada.

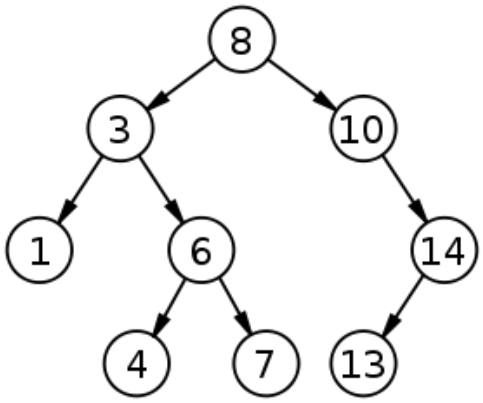

Figura 2: Ejemplo de ordenamiento ABB

2.3. Búsqueda binaria

Se busca en un array ordenado dividiendo repetidamente el intervalo de búsqueda por la mitad. Se comienza con un intervalo que cubra toda la matriz. Si el valor de la clave de búsqueda es menor que el elemento en el medio del intervalo, reduzca el intervalo a la mitad inferior. De lo contrario, estrechelo a la mitad superior. Compruebe repetidamente hasta que se encuentre el valor o el intervalo esté vacío.
La idea de la búsqueda binaria es utilizar la información que el array está ordenado y reducir la complejidad del tiempo a O(Log n).

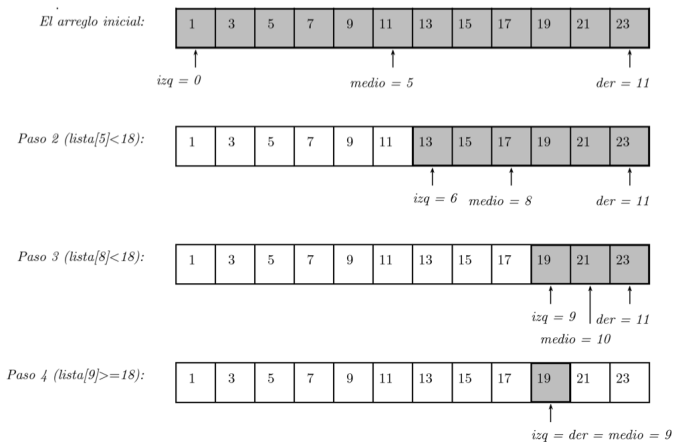

Figura 3: Ejemplo de ordenamiento con búsqueda binaria

2.4. Busqueda Exponencial

El nombre de este algoritmo de búsqueda puede ser engañoso, ya que funciona en tiempo O(Log n). El nombre proviene de la forma en que busca un elemento.
La idea es comenzar con el tamaño 1 de subarray, comparar su último elemento con x, luego probar el tamaño 2, luego 4 y así sucesivamente hasta que el último elemento de un subarray no sea mayor.

2.5. Busqueda de Fibonacci

Dado un array ordenado arr[] de tamaño n y un elemento x para buscar en él. Devuelve el índice de x si está presente en el array, de lo contrario devuelve -1.
Fibonacci Search es una técnica basada en la comparación que utiliza números de Fibonacci para buscar un elemento en una matriz ordenada.
La idea es encontrar primero el número más pequeño de Fibonacci que sea mayor o igual a la longitud del array dado. Deje que el número encontrado de Fibonacci sea fib (m'th Fibonacci number). Utilizamos (m-2)'ésimo número Fibonacci como índice (si es un índice válido). Dejemos que (m-2)'th Fibonacci Number sea i, comparamos arr[i] con x, si x es el mismo, devolvemos i. De lo contrario, si x es mayor, recurrepimos para subarray después de i, de lo contrario recurreamos para subarray antes de i.
El proceso es el siguiente:

- Encuentra el número más pequeño de Fibonacci mayor o igual a n. Deje que este número sea fibM [m'th Fibonacci Number]. Deje que los dos números de Fibonacci que lo preceden sean fibMm1 [(m-1)'ésimo número de Fibonacci] y fibMm2 [(m-2)'ésimo número de Fibonacci].
- Mientras que el array tiene elementos para ser inspeccionados:
  1. Comparar x con el último elemento del rango cubierto por fibMm2
  2. Si x coincide, devuelve el índice
  3. De lo contrario Si x es menor que el elemento, mueva las tres variables de Fibonacci dos Fibonacci hacia abajo, lo que indica la eliminación de aproximadamente dos tercios traseros de la matriz restante.
  4. De lo contrario x es mayor que el elemento, mueva las tres variables de Fibonacci una Fibonacci hacia abajo. Restablecer el desplazamiento al índice. Juntos, estos indican la eliminación de aproximadamente un tercio delantero del array restante.
- Dado que puede haber un solo elemento restante para la comparación, compruebe si fibMm1 es 1. En caso afirmativo, compare x con ese elemento restante. Si coincide, devuelva el índice.

3. Análisis de casos

3.1. Búsqueda en árbol binario de búsqueda

Mejor caso: 1
Peor caso: log2(n)

3.2. Búsqueda binaria

Mejor caso: 1
Peor caso: nlog2(2)

3.3. Búsqueda exponencial

Mejor caso: 1
Peor caso: $\frac{3n^2}{4} + 2$

3.4. Búsqueda Fibonacci

Mejor caso: Contra dominio de Fib(n) piso
Peor caso: $\frac{1}{n+1}$

3.5. Búsqueda lineal

Mejor caso: 1
Peor caso: n

4. Tablas: Busqueda de números

Para cada algoritmo se hizo una prueba y los resultados obtenidos se presentan en la siguiente tabla.

Búsqueda lineal										
numero a buscar	n									
	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
322486	0.003021002	0.006552935	0.009955883	0.013116121	0.014959812	0.016262054	0.018002987	0.019135952	0.020321131	0.021017075
14700764	0.002932072	0.006732941	0.009206057	0.013607025	0.016370058	0.017773867	0.017457008	0.020143032	0.021364927	0.02226305
3128036	0.00316596	0.006315947	0.010331154	0.011481047	0.015607119	0.016217947	0.0183568	0.017633915	0.021997929	0.021842003
6337399	0.003007889	0.006804943	0.010142088	0.013351917	0.015004158	0.016557932	0.019614935	0.01967001	0.020929098	0.021857023
61396	0.00296998	0.006901026	0.010348082	0.012494087	0.015456915	0.016742945	0.018464088	0.019279957	0.020223856	0.021291971
10393545	0.003015041	0.006743908	0.009469986	0.013475895	0.016129971	0.016353846	0.018690109	0.019140959	0.021602154	0.022701979
2147445644	0.003174067	0.006723166	0.01006794	0.013419867	0.016993046	0.016062021	0.017340183	0.019109011	0.020898104	0.020887137
1295390003	0.003143072	0.006524086	0.009900093	0.013370991	0.015566111	0.018137932	0.018207789	0.019304991	0.020128965	0.021478891
450057883	0.003096104	0.006670952	0.010069132	0.012633085	0.01575613	0.016149044	0.017755032	0.020847797	0.021592856	0.02265501
187645041	0.00314498	0.006122112	0.010209084	0.013938904	0.015498877	0.017822027	0.017481089	0.019425154	0.020112038	0.023694992
1980098116	0.002998114	0.006772995	0.010033131	0.012506962	0.016952992	0.017733097	0.017745018	0.018761873	0.021309853	0.021188021
152503	0.003183126	0.006817102	0.009692907	0.013422966	0.015203953	0.01802206	0.017325878	0.019521952	0.020722151	0.023161173
5000	0.003047943	0.006760836	0.009797096	0.013417959	0.016450167	0.015934944	0.018378973	0.019813776	0.019918919	0.022687912
1493283650	0.002963066	0.006103039	0.009860039	0.012769938	0.005680084	0.016821146	0.018875122	0.020049095	0.022204161	0.023278952
214826	0.003154993	0.006757021	0.010350943	0.013365984	0.016445875	0.017160177	0.019938946	0.021121025	0.020376921	0.021902084
3224862	0.003535986	0.006933928	0.010085821	0.012644053	0.014594793	0.017266989	0.017642975	0.018739939	0.020415068	0.021889925
14932836503	0.003154039	0.004831076	0.009708881	0.011590004	0.014708996	0.016755104	0.01832819	0.018934012	0.020529985	0.022903919
1843349527	0.003140926	0.00676012	0.00928402	0.013243198	0.012972117	0.016900063	0.017986059	0.019196987	0.021027088	0.020359039
1360839354	0.003139973	0.006742954	0.009548903	0.013303042	0.016620159	0.016247988	0.019572973	0.020176172	0.021547079	0.022130013
2109248666	0.003448963	0.006737948	0.00998807	0.012329817	0.01444912	0.018115997	0.017986059	0.019932032	0.020251036	0.02109313
2147470852	0.003126144	0.006918907	0.009738922	0.013445139	0.015142918	0.016350031	0.019165993	0.019849062	0.018801928	0.022404909
0	0.003139019	0.006232023	0.010130882	0.01367712	0.016514063	0.017813921	0.018649817	0.019332886	0.021693945	0.021565914
PROMEDIO	0.003122069	0.006582283	0.009894678	0.012996571	0.015074446	0.016923201	0.018300772	0.019513652	0.020775012	0.022032772

Encontrado:	

Figura 4: Tabla de la búsqueda lineal

Árbol binario de búsqueda										
numero a buscar	n									
	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
322486	0.000002861	5.0068E-06	3.0994E-06	5.0068E-06	5.0068E-06	4.7684E-06	5.0068E-06	4.0531E-06	5.0068E-06	4.0531E-06
14700764	2.1458E-06	1.9073E-06	2.1458E-06	3.0994E-06	3.8147E-06	3.0994E-06	3.0994E-06	4.0531E-06	0.000002861	3.8147E-06
3128036	3.8147E-06	3.8147E-06	3.0994E-06	4.0531E-06	0.000002861	3.0994E-06	4.0531E-06	4.0531E-06	4.0531E-06	0.000002861
6337399	1.9073E-06	3.0994E-06	1.9073E-06	3.0994E-06	3.8147E-06	3.0994E-06	3.8147E-06	4.0531E-06	5.0068E-06	3.0994E-06
61396	2.8611E-06	1.9073E-06	2.8611E-06	4.0531E-06	4.0531E-06	3.0994E-06	4.0531E-06	4.0531E-06	3.0994E-06	4.0531E-06
10393545	1.9073E-06	3.0994E-06	3.0994E-06	3.0994E-06	3.0994E-06	5.0068E-06	4.0531E-06	3.0994E-06	0.000002861	3.0994E-06
2147445644	1.9073E-06	3.0994E-06	1.9073E-06	3.0994E-06	1.9073E-06	1.9073E-06	1.9073E-06	3.0994E-06	2.1458E-06	0.000002861
1295390003	3.0994E-06	3.0994E-06	3.8147E-06	4.0531E-06	3.0994E-06	3.0994E-06	4.0531E-06	4.0531E-06	4.0531E-06	3.8147E-06
450057883	1.9073E-06	0.000002861	4.0531E-06	3.0994E-06	3.0994E-06	3.0994E-06	3.8147E-06	4.0531E-06	4.0531E-06	0.000002861
187645041	1.9073E-06	3.0994E-06	0.000002861	3.0994E-06	4.0531E-06	4.0531E-06	3.0994E-06	0.000002861	3.0994E-06	4.0531E-06
1980098116	0.000002861	2.1458E-06	1.9073E-06	4.0531E-06	1.9073E-06	3.0994E-06	3.0994E-06	3.0994E-06	3.0994E-06	3.0994E-06
152503	9.537E-07	3.0994E-06	3.0994E-06	0.000002861	4.0531E-06	3.0994E-06	3.0994E-06	0.000002861	3.0994E-06	4.0531E-06
5000	3.0994E-06	0.000002861	0.000002861	0.000002861	4.0531E-06	0.000002861	3.0994E-06	3.0994E-06	4.0531E-06	0.000002861
1493283650	1.9073E-06	3.0994E-06	0.000002861	3.0994E-06	3.0994E-06	0.000002861	0.000002861	3.0994E-06	0.000002861	3.0994E-06
214826	2.1458E-06	3.8147E-06	1.9073E-06	4.0531E-06	4.0531E-06	3.8147E-06	4.0531E-06	0.000002861	4.0531E-06	3.0994E-06
3224862	0.000002861	5.0068E-06	0.000002861	5.9605E-06	5.0068E-06	5.0068E-06	5.0068E-06	4.0531E-06	5.0068E-06	4.0531E-06
14932836503	1.9073E-06	0.000002861	1.9073E-06	3.0994E-06	0.000002861	0.000002861	3.0994E-06	0.000002861	0.000002861	0.000002861
1843349527	3.0994E-06	3.0994E-06	3.0994E-06	3.8147E-06	3.0994E-06	4.0531E-06	3.8147E-06	0.000002861	4.0531E-06	3.0994E-06
1360839354	3.0994E-06	0.000002861	3.8147E-06	0.000002861	3.8147E-06	4.0531E-06	5.0068E-06	4.0531E-06	4.0531E-06	4.0531E-06
2109248666	3.0994E-06	3.0994E-06	3.8147E-06	3.0994E-06	4.0531E-06	4.0531E-06	3.8147E-06	5.0068E-06	4.0531E-06	4.0531E-06
2147470852	3.0994E-06	2.1458E-06	3.0994E-06	1.9073E-06	1.9073E-06	1.9073E-06	3.0994E-06	0.000002861	3.0994E-06	3.0994E-06
0	3.0994E-06	4.0531E-06	0.000002861	3.0994E-06	3.0994E-06	0.000002861	0.000002861	0.000002861	3.0994E-06	3.0994E-06
PROMEDIO	2.52505E-06	3.14277E-06	0.000002861	3.47872E-06	3.44621E-06	3.40286E-06	3.63045E-06	3.5004E-06	3.61961E-06	3.4137E-06

Encontrado:	

Figura 5: Tabla de la búsqueda en arbol binario

Búsqueda binaria										
numero a buscar	n									
	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
322486	0.000002861	3.0994E-06	3.0994E-06	1.9073E-06	0.000002861	3.0994E-06	3.0994E-06	3.0994E-06	3.0994E-06	3.0994E-06
14700764	3.0994E-06	3.0994E-06	3.0994E-06	4.0531E-06	3.0994E-06	4.0531E-06	0.000002861	3.0994E-06	1.9073E-06	4.0531E-06
3128036	2.1458E-06	2.1458E-06	3.0994E-06	4.0531E-06	2.1458E-06	0.000002861	3.0994E-06	4.0531E-06	3.0994E-06	0.000002861
6337399	0.000002861	2.1458E-06	3.0994E-06	3.0994E-06	0.000002861	4.0531E-06	4.0531E-06	3.8147E-06	3.0994E-06	3.0994E-06
61396	0.000002861	0.000002861	2.1458E-06	3.0994E-06	0.000002861	4.0531E-06	1.9073E-06	3.0994E-06	2.1458E-06	1.9073E-06
10393545	1.9073E-06	3.0994E-06	3.0994E-06	3.0994E-06	3.8147E-06	1.9073E-06	4.0531E-06	3.8147E-06	3.0994E-06	3.0994E-06
2147445644	1.9073E-06	1.9073E-06	2.1458E-06	1.9073E-06	3.0994E-06	2.1458E-06	1.9073E-06	3.0994E-06	1.9073E-06	1.9073E-06
1295390003	1.9073E-06	9.537E-07	0.000002861	1.9073E-06	2.1458E-06	2.1458E-06	3.0994E-06	0.000002861	4.0531E-06	5.0068E-06
450057883	1.9073E-06	2.1458E-06	3.0994E-06	0.000002861	3.8147E-06	4.0531E-06	4.0531E-06	4.0531E-06	3.0994E-06	4.0531E-06
187645041	9.537E-07	4.0531E-06	0.000002861	4.0531E-06	4.0531E-06	3.8147E-06	0.000002861	5.0068E-06	0.000002861	3.8147E-06
1980098116	1.1921E-06	3.0994E-06	3.0994E-06	9.537E-07	1.9073E-06	0.000002861	1.9073E-06	3.0994E-06	1.9073E-06	3.0994E-06
152503	1.9073E-06	2.1458E-06	3.0994E-06	3.0994E-06	1.9073E-06	2.1458E-06	3.0994E-06	0.000002861	4.0531E-06	0.000002861
5000	2.1458E-06	3.0994E-06	1.9073E-06	0.000002861	3.0994E-06	3.0994E-06	0.000002861	3.0994E-06	0.000002861	4.0531E-06
1493283650	1.1921E-06	1.9073E-06	0.000002861	0.000002861	1.9073E-06	9.537E-07	2.1458E-06	3.0994E-06	3.0994E-06	5.0068E-06
214826	1.9073E-06	0.000002861	0.000002861	3.0994E-06	0.000002861	4.0531E-06	0.000002861	0.000002861	3.0994E-06	0.000002861
3224862	2.1458E-06	1.9073E-06	1.9073E-06	3.0994E-06	1.9073E-06	4.0531E-06	0.000002861	3.0994E-06	3.0994E-06	0.000002861
14932836503	1.1921E-06	2.1458E-06	0.000002861	2.1458E-06	1.9073E-06	9.537E-07	2.1458E-06	4.0531E-06	5.0068E-06	4.0531E-06
1843349527	1.1921E-06	2.1458E-06	1.9073E-06	2.1458E-06	9.537E-07	0.000002861	1.9073E-06	3.0994E-06	3.0994E-06	3.0994E-06
1360839354	4.0531E-06	9.537E-07	1.9073E-06	0.000002861	1.9073E-06	1.9073E-06	2.1458E-06	4.0531E-06	5.0068E-06	4.0531E-06
2109248666	1.9073E-06	9.537E-07	3.0994E-06	1.9073E-06	1.9073E-06	1.9073E-06	1.9073E-06	1.9073E-06	1.9073E-06	1.9073E-06
2147470852	1.9073E-06	3.0994E-06	1.9073E-06	1.9073E-06	1.9073E-06	1.9073E-06	1.9073E-06	1.9073E-06	2.1458E-06	3.0994E-06
0	1.1921E-06	1.9073E-06	0.000002861	1.9073E-06	0.000002861	3.0994E-06	2.1458E-06	3.0994E-06	1.9073E-06	1.9073E-06
PROMEDIO	2.0157E-06	2.35166E-06	2.67676E-06	2.67676E-06	2.53588E-06	2.81766E-06	2.67677E-06	3.28365E-06	2.9802E-06	3.26197E-06

Encontrado:	

Figura 6: Tabla de la búsqueda binaria

Búsqueda exponencial										
numero a buscar	n									
	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
322486	1.9073E-06	3.0994E-06	1.9073E-06	1.9073E-06	3.0994E-06	0.000002861	0.000002861	1.9073E-06	2.1458E-06	0.000002861
14700764	0.000002861	3.0994E-06	3.8147E-06	3.0994E-06	3.0994E-06	0.000002861	3.0994E-06	0.000002861	0.000002861	4.0531E-06
3128036	1.9073E-06	0.000002861	3.0994E-06	3.8147E-06	4.0531E-06	1.9073E-06	3.0994E-06	0.000002861	3.0994E-06	0.000002861
6337399	1.9073E-06	3.0994E-06	2.1458E-06	3.0994E-06	3.0994E-06	0.000002861	3.0994E-06	3.0994E-06	3.0994E-06	3.0994E-06
61396	1.9073E-06	1.9073E-06	2.1458E-06	1.9073E-06	3.0994E-06	9.537E-07	0.000002861	2.1458E-06	1.9073E-06	2.1458E-06
10393545	3.0994E-06	3.0994E-06	3.0994E-06	3.0994E-06	3.0994E-06	2.1458E-06	0.000002861	3.0994E-06	1.9073E-06	3.8147E-06
2147445644	2.1458E-06	2.1458E-06	3.0994E-06	3.0994E-06	0.000002861	0.000002861	0.000002861	3.0994E-06	3.0994E-06	3.0994E-06
1295390003	0.000002861	4.0531E-06	1.9073E-06	1.9073E-06	1.9073E-06	3.0994E-06	4.0531E-06	4.0531E-06	5.0068E-06	5.0068E-06
450057883	1.9073E-06	1.9073E-06	3.0994E-06	3.8147E-06	4.0531E-06	3.0994E-06	3.8147E-06	4.0531E-06	4.0531E-06	3.0994E-06
187645041	0.000002861	3.8147E-06	4.0531E-06	4.0531E-06	5.0068E-06	4.0531E-06	4.0531E-06	5.0068E-06	5.0068E-06	0.000002861
1980098116	2.1458E-06	2.1458E-06	3.0994E-06	3.0994E-06	1.9073E-06	0.000002861	1.9073E-06	3.0994E-06	0.000002861	0.000002861
152503	1.9073E-06	1.9073E-06	1.9073E-06	3.0994E-06	3.0994E-06	1.9073E-06	3.0994E-06	3.0994E-06	3.0994E-06	2.1458E-06
5000	1.9073E-06	2.1458E-06	1.9073E-06	1.9073E-06	1.9073E-06	1.9073E-06	1.9073E-06	1.9073E-06	1.9073E-06	3.0994E-06
1493283650	1.9073E-06	0.000002861	0.000002861	0.000002861	0.000002861	1.9073E-06	2.1458E-06	4.0531E-06	4.0531E-06	5.0068E-06
214826	2.1458E-06	2.1458E-06	3.0994E-06	0.000002861	0.000002861	3.0994E-06	2.1458E-06	0.000002861	3.0994E-06	2.1458E-06
3224862	1.9073E-06	2.1458E-06	3.0994E-06	2.1458E-06	2.1458E-06	2.1458E-06	3.0994E-06	0.000002861	1.9073E-06	3.0994E-06
14932836503	0.000002861	0.000002861	3.0994E-06	2.1458E-06	2.1458E-06	3.0994E-06	1.9073E-06	4.0531E-06	5.9605E-06	4.0531E-06
1843349527	2.1458E-06	1.9073E-06	0.000002861	1.9073E-06	2.1458E-06	0.000002861	2.1458E-06	0.000002861	1.9073E-06	4.0531E-06
1360839354	9.537E-07	1.9073E-06	1.9073E-06	1.9073E-06	0.000002861	0.000002861	0.000002861	5.0068E-06	0.000002861	5.0068E-06
2109248666	1.9073E-06	0.000002861	0.000002861	0.000002861	0.000002861	3.0994E-06	3.8147E-06	0.000002861	3.0994E-06	0.000002861
2147470852	3.0994E-06	3.0994E-06	3.0994E-06	1.9073E-06	2.1458E-06	3.0994E-06	3.0994E-06	3.0994E-06	3.0994E-06	3.0994E-06
0	2.1458E-06	1.1921E-06	1.9073E-06	1.9073E-06	3.0994E-06	1.9073E-06	1.9073E-06	1.9073E-06	1.9073E-06	1.9073E-06
PROMEDIO	2.19993E-06	2.55756E-06	2.73095E-06	2.65509E-06	2.88268E-06	2.61174E-06	2.85016E-06	3.17528E-06	3.08858E-06	3.28366E-06

Encontrado:	

Figura 7: Tabla de la búsqueda exponencial

Búsqueda fibonacci										
numero a buscar	n									
	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
322486	0.000002861	1.9073E-06	3.8147E-06	4.0531E-06	5.0068E-06	4.0531E-06	3.8147E-06	0.000002861	0.000002861	3.8147E-06
14700764	0.000002861	4.0531E-06	0.000002861	4.0531E-06	4.0531E-06	5.0068E-06	5.0068E-06	4.0531E-06	5.0068E-06	4.0531E-06
3128036	3.0994E-06	0.000002861	4.0531E-06	5.0068E-06	4.0531E-06	4.0531E-06	0.000002861	3.8147E-06	0.000002861	4.0531E-06
6337399	2.1458E-06	3.0994E-06	4.0531E-06	4.0531E-06	4.0531E-06	3.8147E-06	3.0994E-06	3.0994E-06	4.0531E-06	4.0531E-06
61396	1.9073E-06	2.1458E-06	4.0531E-06	3.0994E-06	0.000002861	0.000002861	3.8147E-06	4.0531E-06	4.0531E-06	4.0531E-06
10393545	1.9073E-06	1.9073E-06	3.8147E-06	0.000002861	3.0994E-06	4.0531E-06	5.0068E-06	3.8147E-06	3.0994E-06	4.0531E-06
2147445644	1.9073E-06	3.0994E-06	2.1458E-06	1.9073E-06	3.0994E-06	3.0994E-06	2.1458E-06	2.1458E-06	1.9073E-06	3.0994E-06
1295390003	9.537E-07	2.1458E-06	0.000002861	0.000002861	1.9073E-06	0.000002861	4.0531E-06	4.0531E-06	5.0068E-06	5.0068E-06
450057883	2.1458E-06	3.0994E-06	3.0994E-06	4.0531E-06	4.0531E-06	4.0531E-06	0.000002861	4.0531E-06	4.0531E-06	4.0531E-06
187645041	1.9073E-06	0.000002861	4.0531E-06	3.0994E-06	3.0994E-06	4.0531E-06	4.0531E-06	3.0994E-06	2.1458E-06	4.0531E-06
1980098116	1.9073E-06	2.1458E-06	0.000002861	1.9073E-06	2.1458E-06	0.000002861	3.0994E-06	2.1458E-06	3.0994E-06	4.0531E-06
152503	1.9073E-06	4.0531E-06	0.000002861	5.0068E-06	4.0531E-06	3.0994E-06	4.0531E-06	0.000002861	4.0531E-06	0.000002861
5000	1.9073E-06	3.0994E-06	3.0994E-06	3.0994E-06	2.1458E-06	3.0994E-06	0.000002861	3.0994E-06	0.000002861	4.0531E-06
1493283650	1.9073E-06	1.9073E-06	3.0994E-06	2.1458E-06	0.000002861	3.0994E-06	3.8147E-06	4.0531E-06	4.0531E-06	4.0531E-06
214826	1.9073E-06	3.0994E-06	4.0531E-06	3.0994E-06	4.0531E-06	3.8147E-06	0.000002861	4.0531E-06	4.0531E-06	5.0068E-06
3224862	1.9073E-06	3.0994E-06	3.8147E-06	4.0531E-06	0.000002861	0.000002861	3.0994E-06	3.0994E-06	4.0531E-06	5.0068E-06
14932836503	2.1458E-06	3.0994E-06	2.1458E-06	1.9073E-06	2.1458E-06	3.0994E-06	0.000002861	4.0531E-06	5.0068E-06	5.0068E-06
1843349527	1.9073E-06	9.537E-07	2.1458E-06	1.9073E-06	1.9073E-06	1.9073E-06	1.9073E-06	1.9073E-06	0.000002861	5.0068E-06
1360839354	1.9073E-06	1.9073E-06	0.000002861	1.9073E-06	1.9073E-06	3.0994E-06	3.0994E-06	3.8147E-06	4.0531E-06	3.0994E-06
2109248666	9.537E-07	2.1458E-06	1.9073E-06	1.9073E-06	2.1458E-06	1.9073E-06	3.0994E-06	1.9073E-06	3.0994E-06	1.9073E-06
2147470852	1.1921E-06	3.0994E-06	9.537E-07	0.000002861	0.000002861	1.9073E-06	1.9073E-06	0.000002861	2.1458E-06	0.000002861
0	1.9073E-06	3.0994E-06	4.0531E-06	3.0994E-06	4.0531E-06	3.0994E-06	4.0531E-06	3.8147E-06	1.9073E-06	4.0531E-06
PROMEDIO	1.96151E-06	2.67677E-06	3.1211E-06	3.08858E-06	3.11026E-06	3.26197E-06	3.33784E-06	3.30533E-06	3.46789E-06	3.9664E-06

Encontrado:	

Figura 8: Tabla de la búsqueda fibonacci

5. Gráficas del comportamiento temoral de cada algoritmo

Todas las gráficas fueron desarrolladas en el software MATLAB 2020b.

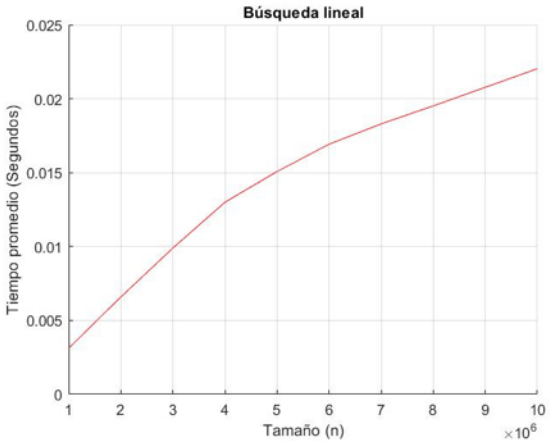

Figura 9: Gráfica de la búsqueda lineal

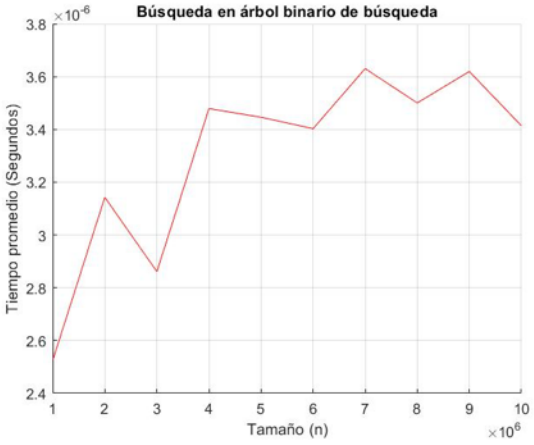

Figura 10: Gráfica de la búsqueda en arbol binario

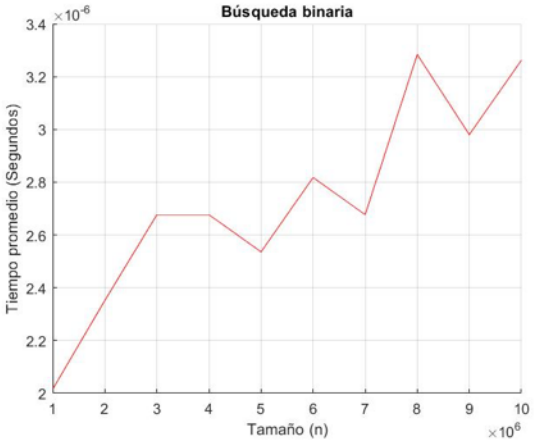

Figura 11: Gráfica de la búsqueda binaria

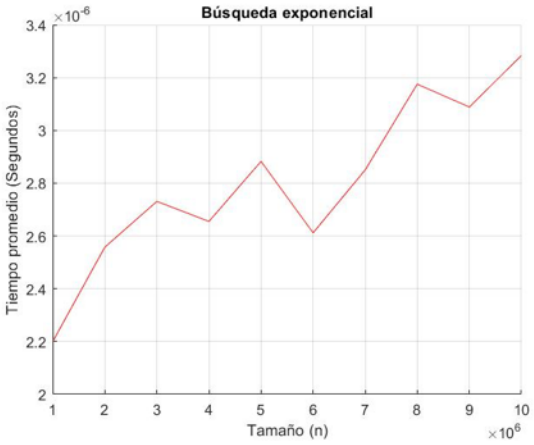

Figura 12: Gráfica de la búsqueda exponencial

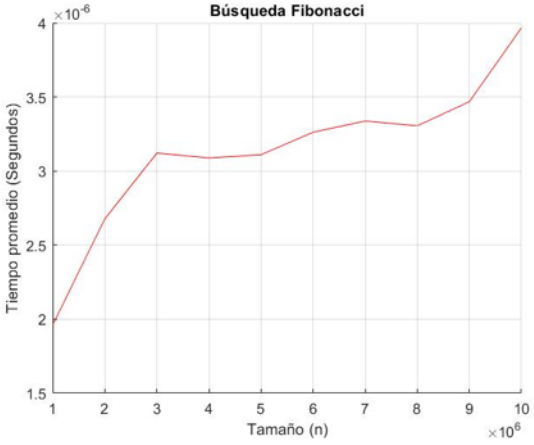

Figura 13: Gráfica de la búsqueda fibonacci

6. Comparación de tiempos promedio de los algoritmos

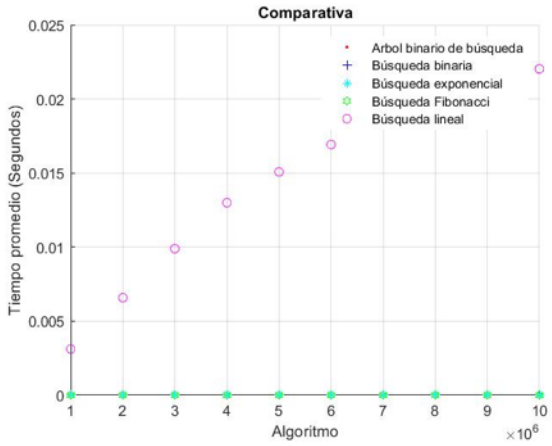

Figura 14: Gráfica de comparativa de algoritmos

7. Ajustes mínimos cuadrados

Todas las gráficas fueron desarrolladas en el software MATLAB 2020b.

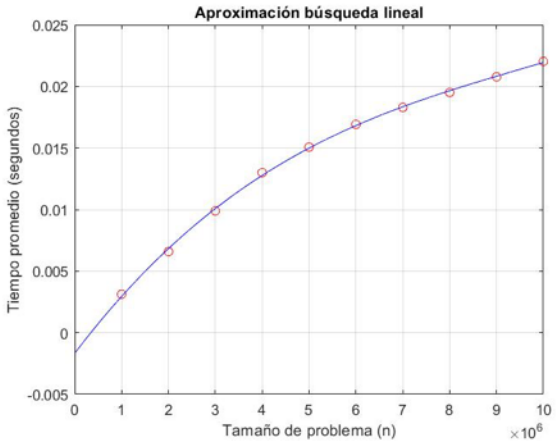

Figura 15: Gráfica del ajuste de mínimos cuadrados de búsqueda lineal

Se elige m=3 porque encaja con la trayectoria de los puntos y porque empieza desde 0.

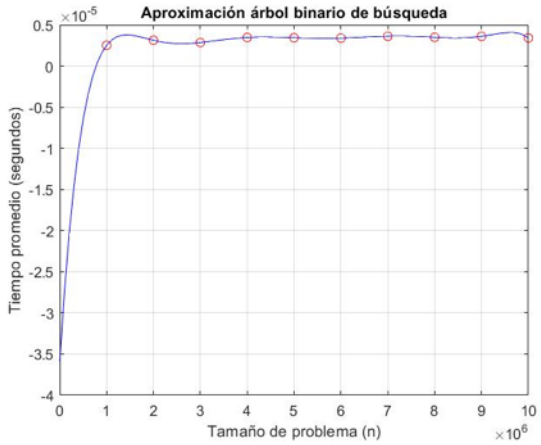

Figura 16: Gráfica del ajuste de mínimos cuadrados de búsqueda en arbol binario

Se elige m=8 porque encaja con todos los puntos y porque empieza desde 0.

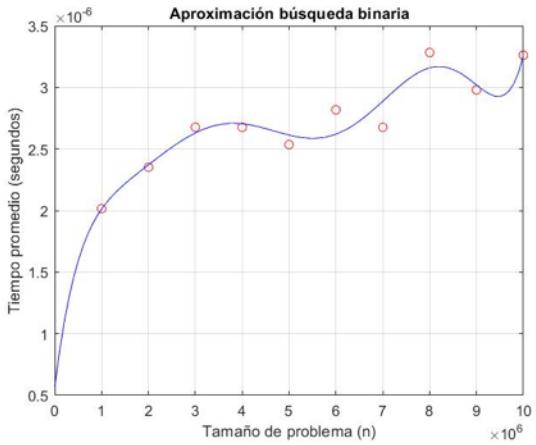

Figura 17: Gráfica del ajuste de mínimos cuadrados de búsqueda binaria

Se elige m=7 porque encaja con la trayectoria de los puntos y porque empieza desde 0.

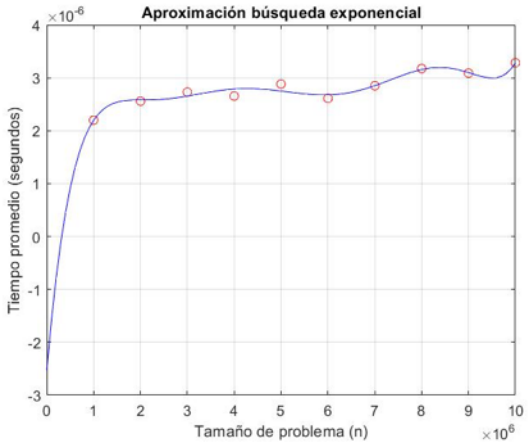

Figura 18: Gráfica del ajuste de mínimos cuadrados de búsqueda exponencial

Se elige m=7 porque encaja con la trayectoria de los puntos y porque empieza desde 0.

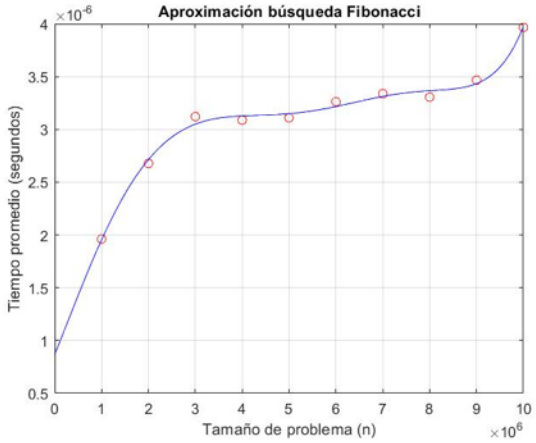

Figura 19: Gráfica del ajuste de mínimos cuadrados de búsqueda fibonacci

Se elige m=7 porque encaja con la trayectoria de los puntos y porque empieza desde 0.

8. Comparativa: Mejores Funciones

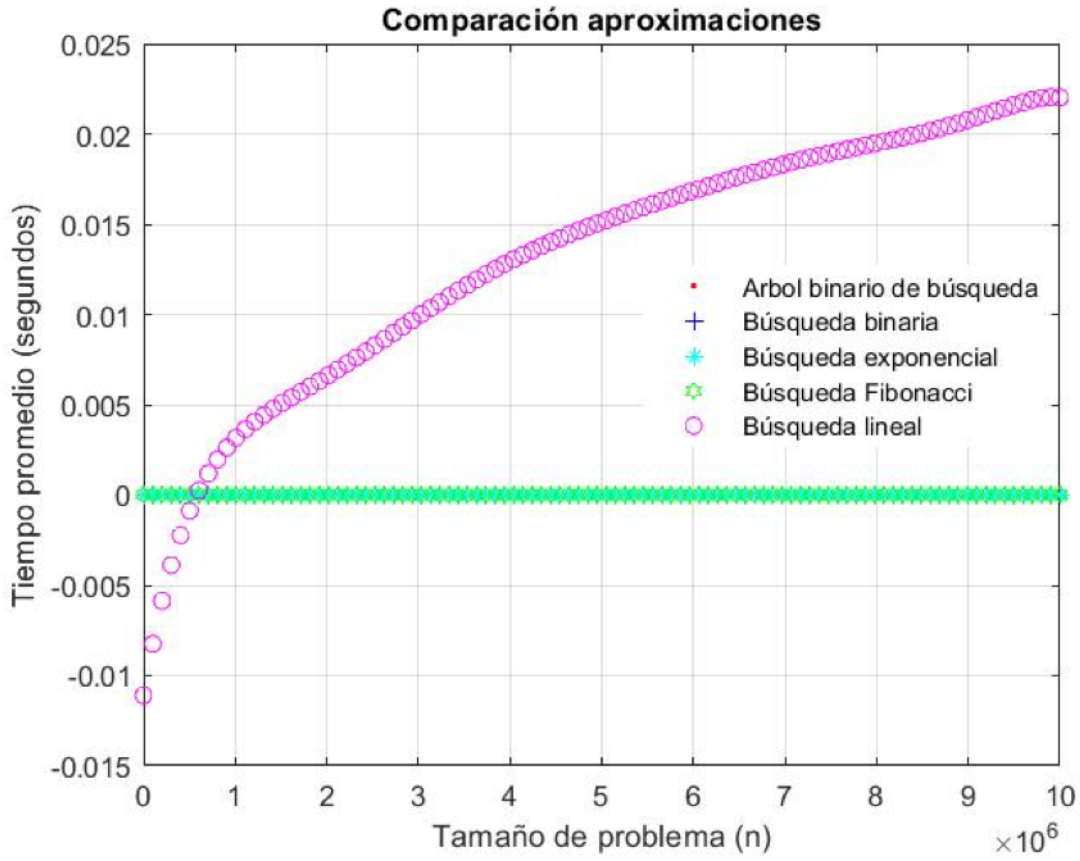

Figura 20: Comparativa de mejores funciones

9. Constante Multiplicativa

Constante multiplicativa: **0.000094249**

3*	Tiempo teórico = (Cnt*n)				
	ABB	Búsqueda binaria	Búsqueda exponencial	Búsqueda Fibonacci	Búsqueda lineal
	(s)	(s)	(s)	(s)	(s)
n=50000000	4712.45	12.20	33310777.50	1.00	4712.45
n=100000000	9424.90	13.20	133243110.02	1.00	9424.90
n=500000000	47124.50	15.52	3331077750.38	1.00	47124.50
n=1000000000	94249.00	16.52	13324311001.50	1.00	94249.00
n=5000000000	471245.00	18.85	333107775037.50	1.00	471245.00

Cuadro 1: Peor caso con "N"diferentes

10. Programas con Hilos

10.1. Árbol de búsqueda binaria

La idea es hacer 2 árboles, donde el primero se llama “a” y el segundo “b”, luego se van insertando los números en los árboles uno y uno. Al terminar tendremos 2 árboles, donde cada uno tiene la misma cantidad de elementos, siendo esa cantidad n/2.
Posteriormente se ponen las posiciones “p” y “q” como raíces de los árboles a y b respectivamente, y cada hilo llama a la función de búsqueda para cada uno de ellos, en caso de que alguno de ellos encuentre el número, pone una bandera en 1.
Se regresa al programa principal y si la bandera está en 1 al terminar los hilos, se dice que se encontró el número buscado.

Búsqueda en árbol binario de búsqueda (Hilos)										
numero a buscar	n									
	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
322486	0.000258923	0.000251055	0.000473976	0.000497818	0.000275135	0.000253916	0.000166893	0.000154972	0.000272036	0.000283957
14700764	0.000488997	0.000253916	0.000479937	0.000277042	0.000261068	0.000265837	0.000141859	0.000477791	0.000490904	0.000261068
3128036	0.000465155	0.000263929	0.000260115	0.000252962	0.00050807	0.000463009	0.000416994	0.000256062	0.000274181	0.00049305
6337399	0.000123024	0.000251055	0.000461102	0.000291109	0.000257969	0.000248194	0.000169039	0.000277996	0.000494003	0.000261068
61396	0.000267029	0.000250101	0.000243902	0.000266075	0.000266075	0.00025487	0.00049901	0.000247002	0.000271082	0.000265122
10393545	0.000258923	0.000308991	0.000473023	0.00047183	0.000490904	0.000266075	0.000341892	0.000274181	0.000272036	0.000263929
2147445644	0.000473023	0.00029397	0.00048399	0.000262976	0.000250101	0.000245094	0.000148058	0.000253916	0.000284195	0.000485897
1295390003	0.000286818	0.000255108	0.000290871	0.000263929	0.000273228	0.000260115	0.000486851	0.000492096	0.000266075	0.000272989
450057883	0.000247955	0.000244141	0.000251055	0.000496864	0.000255108	0.000267983	0.00014782	0.000267029	0.000274181	0.000504971
187645041	0.000257969	0.000257969	0.000262976	0.000255108	0.000477076	0.000782013	0.000365973	0.000478983	0.000249147	0.000271082
1980098116	0.000416994	0.000261068	0.000266075	0.000280142	0.000273943	0.000292063	0.000209808	0.000486136	0.00046587	0.000472784
152503	0.000467062	0.000255823	0.00029397	0.000252962	0.000266075	0.00025487	0.000477076	0.000258923	0.000247955	0.000271082
5000	0.000477076	0.000249863	0.000250816	0.000495911	0.000259876	0.000360012	0.000334024	0.00026989	0.00027585	0.000248909
1493283650	0.000252008	0.000245094	0.000248194	0.000241041	0.000252008	0.000481129	0.000494003	0.000159979	0.000483036	0.000488043
214826	0.000235081	0.000252008	0.000247955	0.00027895	0.000270128	0.000146151	0.000486136	0.000265122	0.000267029	0.000144959
3224862	0.000293016	0.000268936	0.00013113	0.000261068	0.000494003	0.000342131	0.000485897	0.000266075	0.000270844	0.000466108
14932836503	0.000277042	0.000252008	0.000247002	0.000265122	0.000256062	0.000524998	0.000141144	0.000673056	0.000386953	0.000263929
1843349527	0.000256062	0.000263929	0.00026679	0.000262022	0.000273943	0.000180006	0.000172854	0.000488043	0.000253916	0.000488043
1360839354	0.00025487	0.000237942	0.000263929	0.000252962	0.000256062	0.000477076	0.000711203	0.000475884	0.000265837	0.000488997
2109248666	0.000249863	0.00025487	0.000243902	0.000484943	0.000484943	0.000138998	0.000480175	0.000247955	0.000253916	0.000491858
2147470852	0.000248909	0.000247002	0.000485897	0.000248909	0.000253916	0.000355959	0.000347138	0.000271082	0.000691176	0.000151157
0	0.000252962	0.000487089	0.000139952	0.000497103	0.000268936	0.000501156	0.000327826	0.000249863	0.000262022	0.000274181
PROMEDIO	0.000312181	0.000258037	0.000315553	0.000317131	0.000316938	0.00032669	0.000343993	0.000335341	0.00033382	0.000349476

Encontrado:

Figura 21: Hilos - búsqueda ABB

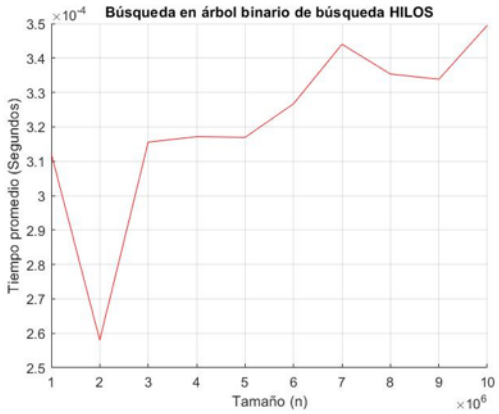

Figura 22: Gráfica de Hilos - búsqueda ABB

Busqueda arbol

Tamaño de n	Tiempo real *Version sin hilos	Tiempo real *Vesion con hilos	Mejora
1000000	2.52505E-06	0.000309489	0.815875683
2000000	3.14277E-06	0.000268448	1.170715553
3000000	0.000002861	0.000307571	0.930192333
4000000	3.47872E-06	0.000325311	1.069350522
5000000	3.44621E-06	0.000314756	1.094883159
6000000	3.40286E-06	0.00033462	1.016930904
7000000	3.63045E-06	0.000343258	1.057644203
8000000	3.5004E-06	0.000331456	1.056066373
9000000	3.61961E-06	0.000330557	1.095004588
10000000	3.4137E-06	0.000346054	0.986464004

Figura 23: Comparativa en búsqueda ABB

10.2. Búsqueda binaria

Se crean 2 arreglos de tamaño n/2, en los cuales se guardarán la mitad de los números introducidos en cada uno. Para no perder que los números están ordenados ni segmentar de manera ineficiente el arreglo completo, se introducen 1 y 1.
Luego con 2 hilos, se llama a la función de búsqueda binaria y cada uno buscará el número en su arreglo propio, si alguno de ellos lo encuentra, pone una bandera en 1.
Se regresa al programa principal y si la bandera está en 1 al terminar los hilos, se dice que se encontró el número buscado.

Búsqueda en árbol binario de búsqueda (Hilos)										
numero a buscar	n									
	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
322486	0.000258923	0.000251055	0.000473976	0.000497818	0.000275135	0.000253916	0.000166893	0.000154972	0.000272036	0.000283957
14700764	0.000488997	0.000253916	0.000479937	0.000277042	0.000261068	0.000265837	0.000141859	0.000477791	0.000490904	0.000261068
3128036	0.000465155	0.000263929	0.000260115	0.000252962	0.00050807	0.000463009	0.000416994	0.000256062	0.000274181	0.00049305
6337399	0.000123024	0.000251055	0.000461102	0.000291109	0.000257969	0.000248194	0.000169039	0.000277996	0.000494003	0.000261068
61396	0.000267029	0.000250101	0.000243902	0.000266075	0.000266075	0.00025487	0.00049901	0.000247002	0.000271082	0.000265122
10393545	0.000258923	0.000308991	0.000473023	0.00047183	0.000490904	0.000266075	0.000341892	0.000274181	0.000272036	0.000263929
2147445644	0.000473023	0.00029397	0.00048399	0.000262976	0.000250101	0.000245094	0.000148058	0.000253916	0.000284195	0.000485897
1295390003	0.000286818	0.000255108	0.000290871	0.000263929	0.000273228	0.000260115	0.000486851	0.000492096	0.000266075	0.000272989
450057883	0.000247955	0.000244141	0.000251055	0.000496864	0.000255108	0.000267983	0.00014782	0.000267029	0.000274181	0.000504971
187645041	0.000257969	0.000257969	0.000262976	0.000255108	0.000477076	0.000782013	0.000365973	0.000478983	0.000249147	0.000271082
1980098116	0.000416994	0.000261068	0.000266075	0.000280142	0.000273943	0.000292063	0.000209808	0.000486136	0.00046587	0.000472784
152503	0.000467062	0.000255823	0.00029397	0.000252962	0.000266075	0.00025487	0.000477076	0.000258923	0.000247955	0.000271082
5000	0.000477076	0.000249863	0.000250816	0.000495911	0.000259876	0.000360012	0.000334024	0.00026989	0.00027585	0.000248909
1493283650	0.000252008	0.000245094	0.000248194	0.000241041	0.000252008	0.000481129	0.000494003	0.000159979	0.000483036	0.000488043
214826	0.000235081	0.000252008	0.000247955	0.00027895	0.000270128	0.000146151	0.000486136	0.000265122	0.000267029	0.000144959
3224862	0.000293016	0.000268936	0.00013113	0.000261068	0.000494003	0.000342131	0.000485897	0.000266075	0.000270844	0.000466108
14932836503	0.000277042	0.000252008	0.000247002	0.000265122	0.000256062	0.000524998	0.000141144	0.000673056	0.000386953	0.000263929
1843349527	0.000256062	0.000263929	0.0026679	0.000262022	0.000273943	0.000180006	0.000172854	0.000488043	0.000253916	0.000488043
1360839354	0.00025487	0.000237942	0.000263929	0.000252962	0.000256062	0.000477076	0.000711203	0.000475884	0.000265837	0.000488997
2109248666	0.000249863	0.00025487	0.000243902	0.000484943	0.000484943	0.000138998	0.000480175	0.000247955	0.000253916	0.000491858
2147470852	0.000248909	0.000247002	0.000485897	0.000248909	0.000253916	0.000355959	0.000347138	0.000271082	0.000691176	0.000151157
0	0.000252962	0.000487089	0.000139952	0.000497103	0.000268936	0.000501156	0.000327826	0.000249863	0.000262022	0.000274181
PROMEDIO	0.000312181	0.000258037	0.000315553	0.000317131	0.000316938	0.00032669	0.000343993	0.000335341	0.00033382	0.000349476

Encontrado:

Figura 24: Hilos - Búsqueda Binaria

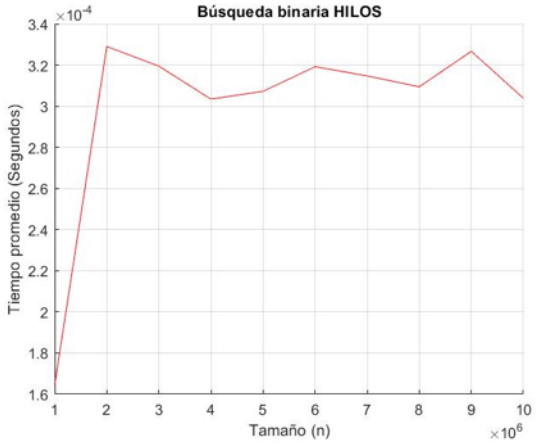

Figura 25: Gráfica de Hilos - búsqueda binaria

Busqueda Binaria

Tamaño de	Tiempo real "Version si	Tiempo real "Vesion con	Mejora
1000000	2.0157E-06	0.000164238	1.227307827
2000000	2.35166E-06	0.000322169	0.729948294
3000000	2.67676E-06	0.000319253	0.838443624
4000000	2.67676E-06	0.000303073	0.883206496
5000000	2.53588E-06	0.000307603	0.824400008
6000000	2.81766E-06	0.000319817	0.881022701
7000000	2.67677E-06	0.000314442	0.851276746
8000000	3.28365E-06	0.000310421	1.057803934
9000000	2.9802E-06	0.000327056	0.911221195
10000000	3.26197E-06	0.000302304	1.079037464

Figura 26: Comparativa en búsqueda binaria

10.3. Búsqueda exponencial

Se crearon 2 arreglos, se guardaron los números 1 y 1 para no perder el orden y poder usar la búsqueda en cada hilo.
Se llama una vez por hilo a la función de búsqueda exponencial, que se encarga de poner unos limites más pequeños y dentro de esos límites hace una búsqueda binaria, cada una con un hilo.
Si se encuentra en alguno, pone en 1 una bandera. Se regresa al programa principal y si la bandera está en 1 al terminar los hilos, se dice que se encontró el número buscado.

Búsqueda exponencial (Hilos)										
numero a buscar	n									
	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
322486	0.00019503	0.00029182	0.00028896	0.00039721	0.00045395	0.00038886	0.00039697	0.000422	0.00041699	0.00040507
14700764	0.0002439	0.00054193	0.00040507	0.00038099	0.00042605	0.00041699	0.00039005	0.000453	0.00039697	0.00041699
3128036	0.00021505	0.00036883	0.00034809	0.00041485	0.00035787	0.00036097	0.00045991	0.00041294	0.00046897	0.00038004
6337399	0.00020599	0.00024796	0.00037217	0.00040889	0.00042415	0.00042987	0.00043011	0.00041604	0.00036597	0.00047588
61396	0.00023603	0.00025606	0.00042796	0.00041199	0.00035906	0.00039196	0.00040507	0.00035	0.00048184	0.00039697
10393545	0.00024295	0.00022602	0.00029898	0.00035596	0.00050306	0.00037813	0.00038505	0.00040412	0.00041986	0.00036383
2147445644	0.00021505	0.00037622	0.00035501	0.00039601	0.00039887	0.00041699	0.00039291	0.00037193	0.00040698	0.00042081
1295390003	0.00023413	0.00023079	0.00049806	0.0003922	0.00040698	0.00043106	0.00118709	0.00044608	0.00060105	0.00038099
450057883	0.00021315	0.00038695	0.0003469	0.00042391	0.00042295	0.00041699	0.00115299	0.00053787	0.0004468	0.00041103
187645041	0.00019598	0.00060105	0.00041199	0.00039506	0.00035095	0.00036287	0.0011611	0.00037789	0.00044298	0.00044107
1980098116	0.00022292	0.00040698	0.00039697	0.00036716	0.00040603	0.00041318	0.00054598	0.00044489	0.00041103	0.00046515
152503	0.00023198	0.00022078	0.00038195	0.00023794	0.00047612	0.00065207	0.00041914	0.00040102	0.00046492	0.00040507
5000	0.00022507	0.00056195	0.00044489	0.00040197	0.000278	0.00039005	0.00049186	0.00045586	0.00040317	0.00034904
1493283650	0.00017381	0.00037503	0.00041413	0.00041294	0.00040102	0.00037313	0.00044489	0.00042105	0.00039005	0.00035596
214826	0.0002389	0.00080299	0.00040197	0.00036621	0.00037003	0.00041914	0.00053811	0.00050402	0.00037909	0.00051999
3224862	0.00022888	0.00021887	0.00043392	0.00041699	0.00036693	0.00035596	0.00049186	0.00038219	0.00036597	0.00043106
14932836503	0.00025702	0.00036693	0.00040698	0.00040603	0.00038886	0.00042892	0.00039697	0.00045681	0.00054717	0.00044489
1843349527	0.00023198	0.00037217	0.0004549	0.00072885	0.00041819	0.00047708	0.00084186	0.00041103	0.00039697	0.00043106
1360839354	0.00022697	0.00043106	0.00040817	0.00026703	0.00039482	0.000386	0.00047207	0.00038099	0.00049305	0.00048399
2109248666	0.00022292	0.00036311	0.00055003	0.00041008	0.0003891	0.00039601	0.00054193	0.00039721	0.00039911	0.000458
2147470852	0.00021791	0.00048494	0.00044823	0.00042892	0.00035191	0.00038981	0.00044608	0.00059605	0.00041795	0.00043178
0	0.00015092	0.00019407	0.00017715	0.00037289	0.00030804	0.00035286	0.00030994	0.00031996	0.00029206	0.00031185
PROMEDIO	0.00022265	0.00038726	0.00040454	0.00040101	0.00039738	0.00041315	0.00057105	0.00043062	0.00043414	0.00042232

Encontrado:	

Figura 27: Hilos - Búsqueda Exponencial

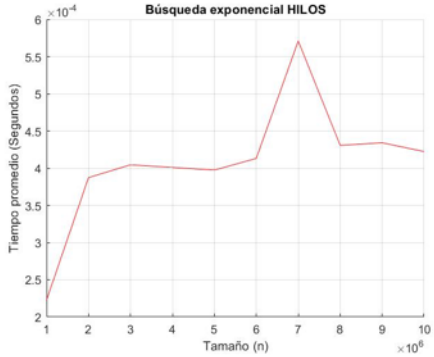

Figura 28: Gráfica de Hilos - búsqueda exponencial

Busqueda Fibo

Tamaño de n	Tiempo real *Version sin hilos	Tiempo real *Vesion con hilos	Mejora
1000000	2.19993E-06	0.000219388	1.002756464
2000000	2.55756E-06	0.000378479	0.675748449
3000000	2.73095E-06	0.000298934	0.913562715
4000000	2.65509E-06	0.00039973	0.664219237
5000000	2.88268E-06	0.000393315	0.732918587
6000000	2.61174E-06	0.000410405	0.636381237
7000000	2.85016E-06	0.000559178	0.509705732
8000000	3.17528E-06	0.000425588	0.746091816
9000000	3.08858E-06	0.00042768	0.722170881
10000000	3.28366E-06	0.0003903	0.841317475

Figura 29: Comparativa en búsqueda exponencial

10.4. Búsqueda Fibonacci

Se crean 2 arreglos y se guardan los números la mitad en uno y la mitad en otro.
Se crean 2 hilos y por cada uno llama a la función de búsqueda Fibonacci, donde se "descomprime" lo que se le manda y dentro viene un id, el cual ayuda a elegir que arreglo usar, si el 1 o el 2 y de ellos hace su búsqueda viendo si el numero buscado es mayor o menor a diferentes números de la sucesión Fibonacci y cuando lo encuentra se pone una bandera en 1.
Se regresa al programa principal y si la bandera está en 1 al terminar los hilos, se dice que se encontró el número buscado.

Búsqueda exponencial (Hilos)										
numero a buscar	n									
	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
322486	0.00448704	0.00991583	0.01381111	0.01476383	0.01937294	0.02009606	0.01990199	0.02515721	0.02794099	0.02865791
14700764	0.00457382	0.0085361	0.01346397	0.01764917	0.01931214	0.02124691	0.02320313	0.02376699	0.02714801	0.02880001
3128036	0.00459194	0.00944018	0.01451111	0.01428294	0.01561284	0.02146697	0.02224708	0.02582908	0.02641797	0.02950311
6337399	0.00465012	0.0095191	0.01454997	0.01686692	0.0180459	0.02048612	0.02247095	0.02527905	0.02742291	0.02911687
61396	0.00461793	0.00840998	0.0144062	0.01514006	0.01850796	0.02040696	0.02232289	0.02423096	0.02524614	0.02751803
10393545	0.00457883	0.00911784	0.01272893	0.01684117	0.01907611	0.0197351	0.02199697	0.02308679	0.02754402	0.02679205
2147445644	0.00447202	0.00930595	0.01412797	0.016752	0.01920199	0.01953101	0.02412915	0.02391505	0.02613091	0.02707314
1295390003	0.00467896	0.00944996	0.01421499	0.01643515	0.0185039	0.02109694	0.02204514	0.02448416	0.02770901	0.02938104
450057883	0.00453782	0.009727	0.01379895	0.01598215	0.01858807	0.01860595	0.01981282	0.02471614	0.026896	0.02912211
187645041	0.00454712	0.0095849	0.01386309	0.01401186	0.01810288	0.02156496	0.02243495	0.02277017	0.12940001	0.02768421
1980098116	0.00461793	0.00931716	0.01354194	0.01688313	0.01900697	0.01675081	0.02290607	0.02515101	0.12933183	0.02874994
152503	0.00453496	0.00834894	0.01386619	0.01784897	0.01741123	0.01976419	0.02177405	0.02584791	0.02660489	0.02901793
5000	0.00450182	0.0087769	0.01308012	0.01694083	0.01930404	0.01902795	0.02016616	0.02521706	0.02724195	0.02684379
1493283650	0.00448704	0.0083859	0.01376891	0.01601887	0.01908803	0.020154	0.0228641	0.02426696	0.02621102	0.02902007
214826	0.00450802	0.00924611	0.0141592	0.01684499	0.01967907	0.02174997	0.02397013	0.02416801	0.02796793	0.02981186
3224862	0.00456786	0.00938892	0.01385999	0.01665807	0.01799202	0.02299905	0.02224708	0.02495599	0.02635694	0.02775788
14932836503	0.00459099	0.00940013	0.01334381	0.017308	0.01918602	0.02050805	0.02269387	0.02629399	0.02682614	0.02905989
1843349527	0.00473905	0.00948501	0.01410699	0.01674914	0.018291	0.01995611	0.02321291	0.02531004	0.02706003	0.02790618
1360839354	0.00437808	0.00881505	0.0140729	0.01661801	0.01775098	0.02181792	0.02216506	0.02331686	0.02762985	0.02930903
2109248666	0.00454092	0.00950599	0.01398087	0.01727486	0.01796603	0.02167606	0.02109694	0.02422285	0.02628613	0.02836394
2147470852	0.0045352	0.00950694	0.01305985	0.01624608	0.01834393	0.02072501	0.02322817	0.02374005	0.02679706	0.02884698
0	0.00448513	0.00918412	0.01389313	0.01733208	0.01826596	0.01991296	0.02133489	0.02533007	0.02740502	0.02720809
PROMEDIO	0.00455893	0.00919923	0.01382462	0.01638649	0.01849257	0.02044601	0.02223284	0.0245584	0.03667475	0.02849219

Encontrado:	

Figura 30: Hilos - Búsqueda Fibonacci

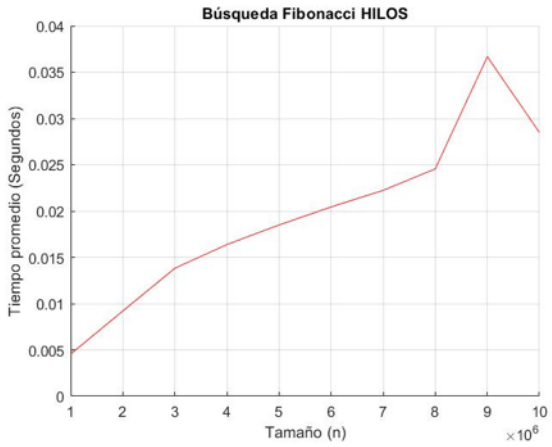

Figura 31: Gráfica de Hilos - búsqueda Fibonacci

Busqueda Fibo

Tamaño de n	Tiempo real *Version sin hilos	Tiempo real *Vesion con hilos	Mejora
1000000	1.96151E-06	0.004555572	0.04305736
2000000	2.67677E-06	0.009198546	0.029099904
3000000	3.1211E-06	0.013827736	0.022571335
4000000	3.08858E-06	0.016429468	0.01879901
5000000	3.11026E-06	0.018482273	0.016828361
6000000	3.26197E-06	0.020421776	0.015973012
7000000	3.33784E-06	0.022192023	0.015040724
8000000	3.30533E-06	0.024593472	0.013439874
9000000	3.46789E-06	0.036253398	0.009565699
10000000	3.9664E-06	0.028433821	0.013949601

Figura 32: Comparativa en búsqueda Fibonacci

10.5. Busqueda lineal

Se guarda todo en un arreglo y se hacen 2 hilos, el primer hilo buscara desde 0 hasta n/2 y el segundo desde n/2 hasta n, si alguno lo encuentra pone en 1 la bandera. Y al regresar al main si la bandera es 1, se dice que se encontró.

Búsqueda exponencial (Hilos)										
numero a buscar	n									
	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
322486	0.003317833	0.004147053	0.007061005	0.008341074	0.010570049	0.014998913	0.016351938	0.019326925	0.018674135	0.024117947
14700764	0.002156019	0.004199028	0.008883953	0.008414984	0.010482788	0.012691975	0.014698029	0.018770218	0.023072004	0.021507978
3128036	0.002419949	0.00287199	0.002854824	0.002864122	0.003452063	0.003396034	0.002933025	0.002867937	0.003134966	0.003280878
6337399	0.002223015	0.004208088	0.006313801	0.008416891	0.011765003	0.012647867	0.014971018	0.015127897	0.014274836	0.013603926
61396	0.002182961	0.00419879	0.006247997	0.009305	0.014148951	0.016168833	0.015069962	0.016772986	0.019512892	0.025426149
10393545	0.002619982	0.004107952	0.006225824	0.008525133	0.01137805	0.012486935	0.014663935	0.014697075	0.016582966	0.01636982
2147445644	0.002057076	0.004227877	0.00623703	0.008512974	0.010507107	0.012473822	0.014904022	0.01719594	0.01939702	0.022493124
1295390003	0.002059937	0.004220963	0.006285906	0.008685112	0.010653019	0.012631178	0.014790058	0.019846916	0.020698071	0.018449068
450057883	0.002062082	0.004199982	0.006224871	0.008482933	0.014065981	0.012548924	0.014750004	0.016945124	0.01916194	0.02135396
187645041	0.002090931	0.004216194	0.006250143	0.008461952	0.01047802	0.016621113	0.014716148	0.018857956	0.022379875	0.02553606
1980098116	0.002089977	0.004227161	0.006220102	0.008303881	0.012326002	0.012783051	0.014693022	0.018589973	0.021023989	0.024766922
152503	0.002085924	0.004175186	0.006322146	0.008328915	0.014383078	0.012539864	0.019505024	0.01926899	0.020321131	0.023633003
5000	0.002085924	0.004199028	0.006287098	0.008422136	0.014096975	0.014950991	0.014662981	0.019458771	0.021950007	0.021244049
1493283650	0.001837015	0.001147986	0.001143932	0.001174212	0.00135994	0.001103878	0.0011549	0.00122714	0.001284838	0.001178026
214826	0.002071142	0.004233122	0.006206036	0.010051966	0.014248848	0.012676954	0.016149998	0.020341158	0.020442963	0.021497965
3224862	0.002499104	0.004241943	0.007147074	0.008481979	0.01215601	0.016938925	0.014878988	0.01728797	0.019631147	0.019190073
14932836503	0.001131058	0.001150847	0.001106978	0.001348019	0.002141953	0.00116396	0.001343966	0.001177073	0.001170158	0.001343966
1843349527	0.002085924	0.004251003	0.005074024	0.005102873	0.005838871	0.006006002	0.005091906	0.006226063	0.005584955	0.006047964
1360839354	0.002774954	0.00422883	0.006268024	0.011522055	0.012564898	0.012689114	0.014729023	0.020808935	0.028777123	0.021616936
2109248666	0.002897024	0.004238844	0.006247044	0.009760141	0.011860132	0.012510061	0.019032002	0.014832974	0.016041994	0.018635035
2147470852	0.002816916	0.004230022	0.006294966	0.0098629	0.010826826	0.014871121	0.017163038	0.017235994	0.019979954	0.017481089
0	0.00215888	0.004206896	0.006341934	0.009163141	0.011190891	0.014931917	0.016023874	0.0179739	0.019501925	0.023164034
PROMEDIO	0.002264988	0.003853423	0.005757275	0.007731869	0.010443074	0.011661881	0.013154904	0.015088763	0.016814141	0.017560664

Encontrado:	

Figura 33: Hilos - Búsqueda lineal

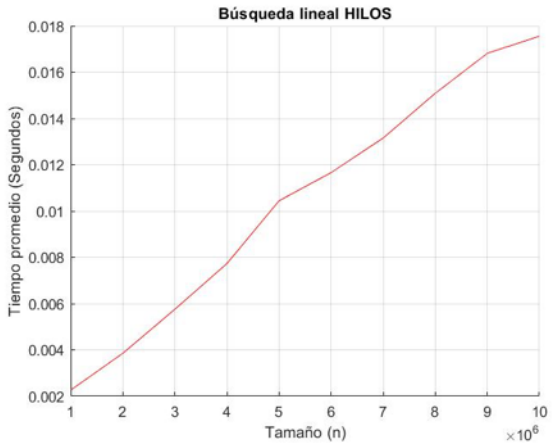

Figura 34: Gráfica de Hilos - búsqueda lineal

Busqueda lineal

Tamaño de n	Tiempo real *Version sin hilos	Tiempo real *Vesion con hilos	Mejora
1000000	0.003122069	0.002260165	138.1345568
2000000	0.006582283	0.00386949	170.1072366
3000000	0.009894678	0.00578385	171.0742281
4000000	0.012996571	0.007796927	166.6883829
5000000	0.015074446	0.010477066	143.8804149
6000000	0.016923201	0.011810519	143.2892139
7000000	0.018300772	0.013285312	137.7519174
8000000	0.019513652	0.015219905	128.2113931
9000000	0.020775012	0.016936313	122.6654915
10000000	0.022032772	0.017815362	123.6728808

Figura 35: Comparativa en búsqueda lineal

11. Preguntas

1- ¿Cuál de los 5 algoritmos es más fácil de implementar?
Búsqueda lineal
2- ¿Cuál de los 5 algoritmos es el más difícil de implementar?
Búsqueda Fibonacci.
3. ¿Cuál de los 5 algoritmos fue el más difícil de modelar en sus variantes con hilos?
Búsqueda con árbol binario de búsqueda.
4. ¿Cuál de los 5 algoritmos en su variante con hilos resultó ser más rápido? ¿Por qué?
Búsqueda binaria porqué, aunque hilos mejora, sigue siendo muy lenta y esta empeora, pero sigue terminó siendo la mejor.
5. ¿Cuál de los 5 algoritmos en su variante no represento alguna ventaja? ¿Por qué?
Casi todos, en realidad solo mejoró lineal. Esto porque los algoritmos están demasiado optimizados que al hacerlo con hilos va a haber hilos que se desperdicien.
6. ¿Cuál algoritmo tiene mejor complejidad temporal?
Búsqueda en árbol binario de búsqueda.
7. ¿Cuál algoritmo tiene mayor complejidad temporal?
Búsqueda lineal.
8. ¿El comportamiento experimental de los algoritmos era el esperado? ¿Por qué?
Si, porque en el análisis previo se veía venir.
9. ¿Sus resultados experimentales difieren mucho de los análisis teóricos que realizo? ¿A qué se debe?
Un poco, porque se calculó lo ideal y no tomamos todas las operaciones como básicas.
10. ¿En la versión con hilos, usar n hilos dividió el tiempo en n? ¿Lo hizo n veces más rápido?
No, no funcionó así.
11. ¿Cuál es el porcentaje de mejora que tiene cada uno de los algoritmos en su variante con hilos? ¿Es lo que esperas? ¿Por qué?

- • Búsqueda en árbol binario de búsqueda——————1.02
- • Búsqueda binaria————————————————0.92
- • Búsqueda exponencial————————————————0.74
- • Búsqueda Fibonacci————————————————0.19
- • Búsqueda lineal————————————————144.54

12. ¿Existió un entorno controlado para realizar las pruebas experimentales? ¿Cuál fue?
Si, se realiz o en una maquina virtual de una computadora con 16 Ram, Ryzen 2600 y Gpu Radeon 5500xt.

13. ¿Si solo se realizara el análisis teórico de un algoritmo antes de implementarlo, podrías asegurar cual es el mejor?
Sí.

14. ¿Qué tan difícil fue realizar el análisis teórico de cada algoritmo?
Fue difícil y más en el Fibonacci.

15. ¿Qué recomendaciones darían a nuevos equipos para realizar esta práctica?
Que se preocupen más por el análisis que por la implementación.

12. Anexos: Códigos

Las pruebas se llevaron en una máquina virtual con ubuntu 20.04.1 en un equipo con 16 Ram, Ryzen 2600 y Gpu Radeon 5500xt. Podemos compilarlo y hacer que se lleve el proceso mediante el comando: ./script.sh

12.1. Ejemplo de script

```
#!/bin/bash
gcc 'abb.c' -o 'ABB'

./ABB 1000000 322486 <numeros10millones.txt >arbol.txt
./ABB 1000000 14700764 <numeros10millones.txt >>arbol.txt
./ABB 1000000 3128036 <numeros10millones.txt >>arbol.txt
./ABB 1000000 6337399 <numeros10millones.txt >>arbol.txt
./ABB 1000000 61396 <numeros10millones.txt >>arbol.txt
./ABB 1000000 10393545 <numeros10millones.txt >>arbol.txt
./ABB 1000000 2147445644 <numeros10millones.txt >>arbol.txt
./ABB 1000000 1295390003 <numeros10millones.txt >>arbol.txt
./ABB 1000000 450057883 <numeros10millones.txt >>arbol.txt
./ABB 1000000 187645041 <numeros10millones.txt >>arbol.txt
./ABB 1000000 1980098116 <numeros10millones.txt >>arbol.txt
./ABB 1000000 152503 <numeros10millones.txt >>arbol.txt
./ABB 1000000 5000 <numeros10millones.txt >>arbol.txt
./ABB 1000000 1493283650 <numeros10millones.txt >>arbol.txt
./ABB 1000000 214826 <numeros10millones.txt >>arbol.txt
./ABB 1000000 1843349527 <numeros10millones.txt >>arbol.txt
./ABB 1000000 1360839354 <numeros10millones.txt >>arbol.txt
./ABB 1000000 2109248666 <numeros10millones.txt >>arbol.txt
./ABB 1000000 2147470852 <numeros10millones.txt >>arbol.txt
./ABB 1000000 0 <numeros10millones.txt >>arbol.txt

./ABB 2000000 322486 <numeros10millones.txt >>arbol.txt
./ABB 2000000 14700764 <numeros10millones.txt >>arbol.txt
./ABB 2000000 3128036 <numeros10millones.txt >>arbol.txt
./ABB 2000000 6337399 <numeros10millones.txt >>arbol.txt
./ABB 2000000 61396 <numeros10millones.txt >>arbol.txt
./ABB 2000000 10393545 <numeros10millones.txt >>arbol.txt
./ABB 2000000 2147445644 <numeros10millones.txt >>arbol.txt
./ABB 2000000 1295390003 <numeros10millones.txt >>arbol.txt
./ABB 2000000 450057883 <numeros10millones.txt >>arbol.txt
./ABB 2000000 187645041 <numeros10millones.txt >>arbol.txt
./ABB 2000000 1980098116 <numeros10millones.txt >>arbol.txt
./ABB 2000000 152503 <numeros10millones.txt >>arbol.txt
./ABB 2000000 5000 <numeros10millones.txt >>arbol.txt
./ABB 2000000 1493283650 <numeros10millones.txt >>arbol.txt
./ABB 2000000 214826 <numeros10millones.txt >>arbol.txt
./ABB 2000000 1843349527 <numeros10millones.txt >>arbol.txt
./ABB 2000000 1360839354 <numeros10millones.txt >>arbol.txt
./ABB 2000000 2109248666 <numeros10millones.txt >>arbol.txt
./ABB 2000000 2147470852 <numeros10millones.txt >>arbol.txt
./ABB 2000000 0 <numeros10millones.txt >>arbol.txt

./ABB 3000000 322486 <numeros10millones.txt >>arbol.txt
./ABB 3000000 14700764 <numeros10millones.txt >>arbol.txt
./ABB 3000000 3128036 <numeros10millones.txt >>arbol.txt
./ABB 3000000 6337399 <numeros10millones.txt >>arbol.txt
./ABB 3000000 61396 <numeros10millones.txt >>arbol.txt
./ABB 3000000 10393545 <numeros10millones.txt >>arbol.txt
./ABB 3000000 2147445644 <numeros10millones.txt >>arbol.txt
./ABB 3000000 1295390003 <numeros10millones.txt >>arbol.txt
./ABB 3000000 450057883 <numeros10millones.txt >>arbol.txt
```

```
./ABB 3000000 187645041 <numeros10millones.txt >>arbol.txt
./ABB 3000000 1980098116 <numeros10millones.txt >>arbol.txt
./ABB 3000000 152503 <numeros10millones.txt >>arbol.txt
./ABB 3000000 5000 <numeros10millones.txt >>arbol.txt
./ABB 3000000 1493283650 <numeros10millones.txt >>arbol.txt
./ABB 3000000 214826 <numeros10millones.txt >>arbol.txt
./ABB 3000000 1843349527 <numeros10millones.txt >>arbol.txt
./ABB 3000000 1360839354 <numeros10millones.txt >>arbol.txt
./ABB 3000000 2109248666 <numeros10millones.txt >>arbol.txt
./ABB 3000000 2147470852 <numeros10millones.txt >>arbol.txt
./ABB 3000000 0 <numeros10millones.txt >>arbol.txt

./ABB 4000000 322486 <numeros10millones.txt >>arbol.txt
./ABB 4000000 14700764 <numeros10millones.txt >>arbol.txt
./ABB 4000000 3128036 <numeros10millones.txt >>arbol.txt
./ABB 4000000 6337399 <numeros10millones.txt >>arbol.txt
./ABB 4000000 61396 <numeros10millones.txt >>arbol.txt
./ABB 4000000 10393545 <numeros10millones.txt >>arbol.txt
./ABB 4000000 2147445644 <numeros10millones.txt >>arbol.txt
./ABB 4000000 1295390003 <numeros10millones.txt >>arbol.txt
./ABB 4000000 450057883 <numeros10millones.txt >>arbol.txt
./ABB 4000000 187645041 <numeros10millones.txt >>arbol.txt
./ABB 4000000 1980098116 <numeros10millones.txt >>arbol.txt
./ABB 4000000 152503 <numeros10millones.txt >>arbol.txt
./ABB 4000000 5000 <numeros10millones.txt >>arbol.txt
./ABB 4000000 1493283650 <numeros10millones.txt >>arbol.txt
./ABB 4000000 214826 <numeros10millones.txt >>arbol.txt
./ABB 4000000 1843349527 <numeros10millones.txt >>arbol.txt
./ABB 4000000 1360839354 <numeros10millones.txt >>arbol.txt
./ABB 4000000 2109248666 <numeros10millones.txt >>arbol.txt
./ABB 4000000 2147470852 <numeros10millones.txt >>arbol.txt
./ABB 4000000 0 <numeros10millones.txt >>arbol.txt

./ABB 5000000 322486 <numeros10millones.txt >>arbol.txt
./ABB 5000000 14700764 <numeros10millones.txt >>arbol.txt
./ABB 5000000 3128036 <numeros10millones.txt >>arbol.txt
./ABB 5000000 6337399 <numeros10millones.txt >>arbol.txt
./ABB 5000000 61396 <numeros10millones.txt >>arbol.txt
./ABB 5000000 10393545 <numeros10millones.txt >>arbol.txt
./ABB 5000000 2147445644 <numeros10millones.txt >>arbol.txt
./ABB 5000000 1295390003 <numeros10millones.txt >>arbol.txt
./ABB 5000000 450057883 <numeros10millones.txt >>arbol.txt
./ABB 5000000 187645041 <numeros10millones.txt >>arbol.txt
./ABB 5000000 1980098116 <numeros10millones.txt >>arbol.txt
./ABB 5000000 152503 <numeros10millones.txt >>arbol.txt
./ABB 5000000 5000 <numeros10millones.txt >>arbol.txt
./ABB 5000000 1493283650 <numeros10millones.txt >>arbol.txt
./ABB 5000000 214826 <numeros10millones.txt >>arbol.txt
./ABB 5000000 1843349527 <numeros10millones.txt >>arbol.txt
./ABB 5000000 1360839354 <numeros10millones.txt >>arbol.txt
./ABB 5000000 2109248666 <numeros10millones.txt >>arbol.txt
./ABB 5000000 2147470852 <numeros10millones.txt >>arbol.txt
./ABB 5000000 0 <numeros10millones.txt >>arbol.txt

./ABB 6000000 322486 <numeros10millones.txt >>arbol.txt
./ABB 6000000 14700764 <numeros10millones.txt >>arbol.txt
./ABB 6000000 3128036 <numeros10millones.txt >>arbol.txt
./ABB 6000000 6337399 <numeros10millones.txt >>arbol.txt
./ABB 6000000 61396 <numeros10millones.txt >>arbol.txt
./ABB 6000000 10393545 <numeros10millones.txt >>arbol.txt
./ABB 6000000 2147445644 <numeros10millones.txt >>arbol.txt
```

```
./ABB 6000000 1295390003 <numeros10millones.txt >>arbol.txt
./ABB 6000000 450057883 <numeros10millones.txt >>arbol.txt
./ABB 6000000 187645041 <numeros10millones.txt >>arbol.txt
./ABB 6000000 1980098116 <numeros10millones.txt >>arbol.txt
./ABB 6000000 152503 <numeros10millones.txt >>arbol.txt
./ABB 6000000 5000 <numeros10millones.txt >>arbol.txt
./ABB 6000000 1493283650 <numeros10millones.txt >>arbol.txt
./ABB 6000000 214826 <numeros10millones.txt >>arbol.txt
./ABB 6000000 1843349527 <numeros10millones.txt >>arbol.txt
./ABB 6000000 1360839354 <numeros10millones.txt >>arbol.txt
./ABB 6000000 2109248666 <numeros10millones.txt >>arbol.txt
./ABB 6000000 2147470852 <numeros10millones.txt >>arbol.txt
./ABB 6000000 0 <numeros10millones.txt >>arbol.txt

./ABB 7000000 322486 <numeros10millones.txt >>arbol.txt
./ABB 7000000 14700764 <numeros10millones.txt >>arbol.txt
./ABB 7000000 3128036 <numeros10millones.txt >>arbol.txt
./ABB 7000000 6337399 <numeros10millones.txt >>arbol.txt
./ABB 7000000 61396 <numeros10millones.txt >>arbol.txt
./ABB 7000000 10393545 <numeros10millones.txt >>arbol.txt
./ABB 7000000 2147445644 <numeros10millones.txt >>arbol.txt
./ABB 7000000 1295390003 <numeros10millones.txt >>arbol.txt
./ABB 7000000 450057883 <numeros10millones.txt >>arbol.txt
./ABB 7000000 187645041 <numeros10millones.txt >>arbol.txt
./ABB 7000000 1980098116 <numeros10millones.txt >>arbol.txt
./ABB 7000000 152503 <numeros10millones.txt >>arbol.txt
./ABB 7000000 5000 <numeros10millones.txt >>arbol.txt
./ABB 7000000 1493283650 <numeros10millones.txt >>arbol.txt
./ABB 7000000 214826 <numeros10millones.txt >>arbol.txt
./ABB 7000000 1843349527 <numeros10millones.txt >>arbol.txt
./ABB 7000000 1360839354 <numeros10millones.txt >>arbol.txt
./ABB 7000000 2109248666 <numeros10millones.txt >>arbol.txt
./ABB 7000000 2147470852 <numeros10millones.txt >>arbol.txt
./ABB 7000000 0 <numeros10millones.txt >>arbol.txt

./ABB 8000000 322486 <numeros10millones.txt >>arbol.txt
./ABB 8000000 14700764 <numeros10millones.txt >>arbol.txt
./ABB 8000000 3128036 <numeros10millones.txt >>arbol.txt
./ABB 8000000 6337399 <numeros10millones.txt >>arbol.txt
./ABB 8000000 61396 <numeros10millones.txt >>arbol.txt
./ABB 8000000 10393545 <numeros10millones.txt >>arbol.txt
./ABB 8000000 2147445644 <numeros10millones.txt >>arbol.txt
./ABB 8000000 1295390003 <numeros10millones.txt >>arbol.txt
./ABB 8000000 450057883 <numeros10millones.txt >>arbol.txt
./ABB 8000000 187645041 <numeros10millones.txt >>arbol.txt
./ABB 8000000 1980098116 <numeros10millones.txt >>arbol.txt
./ABB 8000000 152503 <numeros10millones.txt >>arbol.txt
./ABB 8000000 5000 <numeros10millones.txt >>arbol.txt
./ABB 8000000 1493283650 <numeros10millones.txt >>arbol.txt
./ABB 8000000 214826 <numeros10millones.txt >>arbol.txt
./ABB 8000000 1843349527 <numeros10millones.txt >>arbol.txt
./ABB 8000000 1360839354 <numeros10millones.txt >>arbol.txt
./ABB 8000000 2109248666 <numeros10millones.txt >>arbol.txt
./ABB 8000000 2147470852 <numeros10millones.txt >>arbol.txt
./ABB 8000000 0 <numeros10millones.txt >>arbol.txt

./ABB 9000000 322486 <numeros10millones.txt >>arbol.txt
./ABB 9000000 14700764 <numeros10millones.txt >>arbol.txt
./ABB 9000000 3128036 <numeros10millones.txt >>arbol.txt
./ABB 9000000 6337399 <numeros10millones.txt >>arbol.txt
./ABB 9000000 61396 <numeros10millones.txt >>arbol.txt
```

```
./ABB 9000000 10393545 <numeros10millones.txt >>arbol.txt
./ABB 9000000 2147445644 <numeros10millones.txt >>arbol.txt
./ABB 9000000 1295390003 <numeros10millones.txt >>arbol.txt
./ABB 9000000 450057883 <numeros10millones.txt >>arbol.txt
./ABB 9000000 187645041 <numeros10millones.txt >>arbol.txt
./ABB 9000000 1980098116 <numeros10millones.txt >>arbol.txt
./ABB 9000000 152503 <numeros10millones.txt >>arbol.txt
./ABB 9000000 5000 <numeros10millones.txt >>arbol.txt
./ABB 9000000 1493283650 <numeros10millones.txt >>arbol.txt
./ABB 9000000 214826 <numeros10millones.txt >>arbol.txt
./ABB 9000000 1843349527 <numeros10millones.txt >>arbol.txt
./ABB 9000000 1360839354 <numeros10millones.txt >>arbol.txt
./ABB 9000000 2109248666 <numeros10millones.txt >>arbol.txt
./ABB 9000000 2147470852 <numeros10millones.txt >>arbol.txt
./ABB 9000000 0 <numeros10millones.txt >>arbol.txt

./ABB 10000000 322486 <numeros10millones.txt >>arbol.txt
./ABB 10000000 14700764 <numeros10millones.txt >>arbol.txt
./ABB 10000000 3128036 <numeros10millones.txt >>arbol.txt
./ABB 10000000 6337399 <numeros10millones.txt >>arbol.txt
./ABB 10000000 61396 <numeros10millones.txt >>arbol.txt
./ABB 10000000 10393545 <numeros10millones.txt >>arbol.txt
./ABB 10000000 2147445644 <numeros10millones.txt >>arbol.txt
./ABB 10000000 1295390003 <numeros10millones.txt >>arbol.txt
./ABB 10000000 450057883 <numeros10millones.txt >>arbol.txt
./ABB 10000000 187645041 <numeros10millones.txt >>arbol.txt
./ABB 10000000 1980098116 <numeros10millones.txt >>arbol.txt
./ABB 10000000 152503 <numeros10millones.txt >>arbol.txt
./ABB 10000000 5000 <numeros10millones.txt >>arbol.txt
./ABB 10000000 1493283650 <numeros10millones.txt >>arbol.txt
./ABB 10000000 214826 <numeros10millones.txt >>arbol.txt
./ABB 10000000 1843349527 <numeros10millones.txt >>arbol.txt
./ABB 10000000 1360839354 <numeros10millones.txt >>arbol.txt
./ABB 10000000 2109248666 <numeros10millones.txt >>arbol.txt
./ABB 10000000 2147470852 <numeros10millones.txt >>arbol.txt
./ABB 10000000 0 <numeros10millones.txt >>arbol.txt
```

12.2. Código: Arbol de Busqueda Binaria

```c
//Practica 1
//Arbol binario de busqueda, busqueda

#include<stdio.h>
#include<stdlib.h>
#include <sys/resource.h>
#include <sys/time.h>

struct Arbol
{
  int dato;
  struct Arbol *izq;
  struct Arbol *der;
};

void uswtime(double *usertime, double *systime, double *walltime);
struct Arbol *AgregarElemento(struct Arbol *,int);
void BuscarElemento(struct Arbol *raiz, int dato);
void ImprimirTiempos(double,double,double,double,double,double);
int count = 0;

int main(int argc, char const *argv[])
{
  struct Arbol *arbol = NULL;
  int num, n, i,bus;
  double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para
      medici n de tiempos

  //Numero de numeros que se ordenar n
  n=atoi(argv[1]);
  int *numeros = (int*) malloc(sizeof(int)*n);

  for(i = 0; i < n + 1; i++){
    scanf("%d",&num);
    arbol = AgregarElemento(arbol,num);
  }

  bus = atoi(argv[2]);

  uswtime(&utime0, &stime0, &wtime0);
  printf("\tABB con %d numeros, buscando el %d\n",n,bus);
  BuscarElemento(arbol,bus);
  uswtime(&utime1, &stime1, &wtime1);

  //for (i = 0; i < 10000000; ++i) printf("%d. %d\n",i+1,*(numeros + i));


  ImprimirTiempos(utime0, stime0, wtime0,utime1, stime1, wtime1);
  printf("-----------------------------------------------\n");

  return 0;
}

struct Arbol *AgregarElemento(struct Arbol *raiz, int dato)
{
  if(raiz == NULL) // Caso base
  {
    struct Arbol *nuevo = NULL;
    nuevo = (struct Arbol *) malloc(sizeof(struct Arbol));
```

```
59      nuevo -> dato = dato;
60      nuevo -> izq = NULL;
61      nuevo -> der = NULL;
62      return nuevo;
63    }
64
65    if(dato < raiz -> dato){
66      raiz -> izq = AgregarElemento(raiz -> izq, dato);
67    }
68    else
69    {
70      raiz -> der = AgregarElemento(raiz -> der, dato);
71    }
72
73    return raiz;
74  }
75
76  void BuscarElemento(struct Arbol *raiz, int dato)
77  {
78      struct Arbol *aux = raiz;
79      int flag = 0;
80
81      while(aux != NULL && flag == 0) {
82        if(dato == aux->dato)
83          flag = 1;
84        else if(dato < aux->dato)
85          aux = aux->izq;
86        else
87          aux = aux->der;
88      }
89
90      return;
91  }
92
93  void uswtime(double *usertime, double *systime, double *walltime)
94  {
95    double mega = 1.0e-6;
96    struct rusage buffer;
97    struct timeval tp;
98    struct timezone tzp;
99    getrusage(RUSAGE_SELF, &buffer);
100   gettimeofday(&tp, &tzp);
101   *usertime = (double) buffer.ru_utime.tv_sec +1.0e-6 * buffer.ru_utime.
        tv_usec;
102   *systime  = (double) buffer.ru_stime.tv_sec +1.0e-6 * buffer.ru_stime.
        tv_usec;
103   *walltime = (double) tp.tv_sec + 1.0e-6 * tp.tv_usec;
104 }
105
106 void ImprimirTiempos(double utime0,double stime0,double wtime0,double utime1
      ,double stime1,double wtime1){
107   printf("\n");
108   /*printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
109   printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0
        );
110   printf("sys (Tiempo en acci nes de E/S)  %.10f s\n",  stime1 - stime0);
111   printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0
        ) / (wtime1 - wtime0));*/
112   printf("Tiempo: %.10f\n", wtime1 - wtime0);
113 }
```

12.3. Código: Arbol de Busqueda Binaria con Hilos

```
1  //Practica 1
2  //Arbol binario de busqueda, busqueda
3  #include <pthread.h>
4  #include<stdio.h>
5  #include<stdlib.h>
6  #include <sys/resource.h>
7  #include <sys/time.h>
8
9  struct Arbol
10 {
11   int dato;
12   struct Arbol *izq;
13   struct Arbol *der;
14 };
15
16 void *funcion(void *arbol);
17 void *funcionPrimerNodo(void *arbol);
18 void uswtime(double *usertime, double *systime, double *walltime);
19 struct Arbol *AgregarElemento(struct Arbol *,int);
20 void BuscarElemento(struct Arbol *raiz, int dato);
21 void BuscarElemento_PrimerNodo(struct Arbol *raiz, int dato);
22 void ImprimirTiempos(double,double,double,double,double,double);
23 int count = 0,bus,flag;
24
25 int main(int argc, char const *argv[])
26 {
27   struct Arbol *arbol = NULL;
28   int num, n, i;
29   double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para
       medici n de tiempos
30   pthread_t hilo1, hilo2, hilo3;
31
32   //Numero de numeros que se ordenar n
33   n=atoi(argv[1]);
34   int *numeros = (int*) malloc(sizeof(int)*n);
35
36   for(i = 0; i < n + 1; i++){
37     scanf("%d",&num);
38     arbol = AgregarElemento(arbol,num);
39   }
40
41   bus = atoi(argv[2]);
42
43   printf("\tABB con %d numeros, buscando el %d",n,bus);
44
45   uswtime(&utime0, &stime0, &wtime0);
46   if(pthread_create (&hilo1, NULL, funcionPrimerNodo,(void*)arbol) != 0 )
47     { printf("Error\n"); return -1; }
48
49   if(pthread_create (&hilo2, NULL, funcion,(void*)arbol->izq) != 0 )
50     { printf("Error\n"); return -1; }
51
52   if(pthread_create (&hilo3, NULL, funcion,(void*)arbol->der) != 0 )
53     { printf("Error\n"); return -1; }
54
55   pthread_join(hilo1,NULL);
56   pthread_join(hilo2,NULL);
57   pthread_join(hilo3,NULL);
58   uswtime(&utime1, &stime1, &wtime1);
```

```
59
60    //for (i = 0; i < 10000000; ++i) printf("%d. %d\n",i+1,*(numeros + i));
61    printf(" (%d)\n",flag);
62
63    ImprimirTiempos(utime0, stime0, wtime0,utime1, stime1, wtime1);
64    printf("-----------------------------------------------\n");
65
66    return 0;
67  }
68
69  struct Arbol *AgregarElemento(struct Arbol *raiz, int dato)
70  {
71    if(raiz == NULL) // Caso base
72    {
73      struct Arbol *nuevo = NULL;
74      nuevo = (struct Arbol *) malloc(sizeof(struct Arbol));
75      nuevo -> dato = dato;
76      nuevo -> izq = NULL;
77      nuevo -> der = NULL;
78      return nuevo;
79    }
80
81    if(dato < raiz -> dato){
82      raiz -> izq = AgregarElemento(raiz -> izq, dato);
83    }
84    else
85    {
86      raiz -> der = AgregarElemento(raiz -> der, dato);
87    }
88
89    return raiz;
90  }
91
92  void BuscarElemento(struct Arbol *raiz, int dato)
93  {
94      struct Arbol *aux = raiz;
95      flag = 0;
96
97      while(aux != NULL && flag == 0) {
98        if(dato == aux->dato)
99          flag = 1;
100       else if(dato < aux->dato)
101         aux = aux->izq;
102       else
103         aux = aux->der;
104     }
105
106     return;
107 }
108 void BuscarElemento_PrimerNodo(struct Arbol *raiz, int dato){
109
110     struct Arbol *aux = raiz;
111     flag = 0;
112
113     if(aux != NULL)
114       if(dato == aux->dato)
115         flag = 1;
116       else
117         return;
118   return;
119 }
```

```
120
121 void uswtime(double *usertime, double *systime, double *walltime)
122 {
123   double mega = 1.0e-6;
124   struct rusage buffer;
125   struct timeval tp;
126   struct timezone tzp;
127   getrusage(RUSAGE_SELF, &buffer);
128   gettimeofday(&tp, &tzp);
129   *usertime = (double) buffer.ru_utime.tv_sec +1.0e-6 * buffer.ru_utime.
        tv_usec;
130   *systime  = (double) buffer.ru_stime.tv_sec +1.0e-6 * buffer.ru_stime.
        tv_usec;
131   *walltime = (double) tp.tv_sec + 1.0e-6 * tp.tv_usec;
132 }
133
134 void ImprimirTiempos(double utime0,double stime0,double wtime0,double utime1
      ,double stime1,double wtime1){
135   printf("\n");
136   /*printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
137   printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0
        );
138   printf("sys (Tiempo en acci nes de E/S)  %.10f s\n",  stime1 - stime0);
139   printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0
        ) / (wtime1 - wtime0));*/
140   printf("Tiempo: %.10f\n", wtime1 - wtime0);
141 }
142
143 void *funcion(void *arbol){
144   BuscarElemento((struct Arbol *)arbol,bus);
145 }
146
147 void *funcionPrimerNodo(void *arbol){
148   BuscarElemento_PrimerNodo((struct Arbol *)arbol,bus);
149 }
```

12.4. Código: Busqueda Binaria

```c
//Busqueda Binaria
//Primero ordenaremos la entrada
#include<stdio.h>
#include<stdlib.h>
#include <sys/resource.h>
#include <sys/time.h>

struct Arbol
{
  int dato;
  struct Arbol *izq;
  struct Arbol *der;
};

void uswtime(double *usertime, double *systime, double *walltime);
struct Arbol *AgregarElemento(struct Arbol *,int);
void InOrden(struct Arbol *,int *);
void ImprimirTiempos(double,double,double,double,double,double);
int buscar(int*,int,int,int);
int count = 0,cuenta = 0;

int main(int argc, char const *argv[])
{
  struct Arbol *arbol = NULL;
  int num, n, i;
  double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para
      medici n de tiempos

  //Numero de numeros que se ordenar n
  n=atoi(argv[1]);
  int *numeros = (int*) malloc(sizeof(int)*n);
  int bus=atoi(argv[2]);

  for(i = 0; i < n + 1; i++){
    scanf("%d",&num);
    arbol = AgregarElemento(arbol,num);
  }
  InOrden(arbol,numeros);
  //for (i = 0; i < n + 1; ++i) printf("%d. %d\n",i+1,*(numeros + i));

  uswtime(&utime0, &stime0, &wtime0);
  int encontrado = buscar(numeros,bus,n,0);
  uswtime(&utime1, &stime1, &wtime1);

  printf("\nBusqueda Binaria con %d numeros, buscando el %d: (%d)",n,bus,
      encontrado);
  if(encontrado == -1) printf(" %d vueltas\n",cuenta);
  else printf("\n");


  ImprimirTiempos(utime0, stime0, wtime0,utime1, stime1, wtime1);
  printf("-----------------------------------------------------\n");

  return 0;
}

struct Arbol *AgregarElemento(struct Arbol *raiz, int dato)
{
  if(raiz == NULL) // Caso base
```

```
58    {
59      struct Arbol *nuevo = NULL;
60      nuevo = (struct Arbol *) malloc(sizeof(struct Arbol));
61      nuevo -> dato = dato;
62      nuevo -> izq = NULL;
63      nuevo -> der = NULL;
64      return nuevo;
65    }
66
67    if(dato < raiz -> dato){
68      raiz -> izq = AgregarElemento(raiz -> izq, dato);
69    }
70    else
71    {
72      raiz -> der = AgregarElemento(raiz -> der, dato);
73    }
74
75    return raiz;
76  }
77
78  void InOrden(struct Arbol *arbol, int *numeros)
79  {
80    if(arbol == NULL)
81      return;
82
83    InOrden(arbol -> izq,numeros);
84    *(numeros + count) = arbol -> dato;
85    count++;
86    //printf("%d\n",arbol -> dato);
87    InOrden(arbol -> der,numeros);
88  }
89
90  void uswtime(double *usertime, double *systime, double *walltime)
91  {
92    double mega = 1.0e-6;
93    struct rusage buffer;
94    struct timeval tp;
95    struct timezone tzp;
96    getrusage(RUSAGE_SELF, &buffer);
97    gettimeofday(&tp, &tzp);
98    *usertime = (double) buffer.ru_utime.tv_sec +1.0e-6 * buffer.ru_utime.
        tv_usec;
99    *systime  = (double) buffer.ru_stime.tv_sec +1.0e-6 * buffer.ru_stime.
        tv_usec;
100   *walltime = (double) tp.tv_sec + 1.0e-6 * tp.tv_usec;
101 }
102
103 void ImprimirTiempos(double utime0,double stime0,double wtime0,double utime1
      ,double stime1,double wtime1)
104 {
105   printf("\n");
106   printf("Tiempo: %.10f\n", wtime1 - wtime0);
107 }
108 int buscar(int *numeros, int bus, int tam, int inicio)
109 {
110   cuenta = 1;
111   int mid = inicio + (tam - inicio) / 2;
112
113   if (*(numeros + inicio) == bus)
114
115   if (*(numeros + mid) == bus)
```

```
116       return 1;
117     if (*(numeros + tam) == bus)
118       return 1;
119
120     if(*(numeros + mid) > bus)
121       tam = mid - 1;
122     else
123       inicio = mid + 1;
124
125     while(inicio <= tam)
126     {
127       cuenta++;
128       mid = inicio + (tam - inicio) / 2;
129       if (*(numeros + inicio) == bus)
130         return 1;
131       if (*(numeros + mid) == bus)
132         return 1;
133       if (*(numeros + tam) == bus)
134         return 1;
135       if(*(numeros + mid) > bus)
136         tam = mid - 1;
137       else
138         inicio = mid + 1;
139     }
140     return -1;
141 }
```

12.5. Código: Busqueda Binaria con Hilos

```
1  //Busqueda Binaria
2  //Primero ordenaremos la entrada
3  #include <pthread.h>
4  #include<stdio.h>
5  #include<stdlib.h>
6  #include <sys/resource.h>
7  #include <sys/time.h>
8
9  struct Arbol
10 {
11   int dato;
12   struct Arbol *izq;
13   struct Arbol *der;
14 };
15
16 void uswtime(double *usertime, double *systime, double *walltime);
17 struct Arbol *AgregarElemento(struct Arbol *,int);
18 void InOrden(struct Arbol *,int *);
19 void ImprimirTiempos(double,double,double,double,double,double);
20 void buscar(int*,int,int,int);
21 void *funcion(void *arbol);
22 int count = 0,cuenta = 0,bus,n;
23
24 int main(int argc, char const *argv[])
25 {
26   struct Arbol *arbol = NULL;
27   int num, i;
28   double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para
       medici n de tiempos
29   pthread_t hilo1,hilo2;
30
31   //Numero de numeros que se ordenar n
32   n=atoi(argv[1]);
33   int *numeros = (int*) malloc(sizeof(int)*n);
34   bus=atoi(argv[2]);
35   int ind_mitad = n/2;
36
37   for(i = 0; i < n + 1; i++){
38     scanf("%d",&num);
39     arbol = AgregarElemento(arbol,num);
40   }
41
42   InOrden(arbol,numeros);
43
44   int *primera = (int*) malloc(sizeof(int)*n/2);
45   int *segunda = (int*) malloc(sizeof(int)*n/2);
46
47   for (i = 0; i <= ind_mitad; ++i) *(primera + i) = *(numeros + i);
48   for (int j = ind_mitad+1,i = 0; j <= n; ++j,i++) *(segunda + i) = *(
       numeros +j);
49
50   //for (i = 0; i <= n/2; ++i) printf("%d. %d\n",i+1,*(primera + i));
51   //for (i = 0; i <= n/2; ++i) printf("%d. %d\n",i+1,*(segunda + i));
52
53
54   uswtime(&utime0, &stime0, &wtime0);
55
56   if(pthread_create (&hilo1, NULL, funcion,(void*)primera) != 0 )
57     { printf("Error\n"); return -1; }
```

```
58    if(pthread_create (&hilo2, NULL, funcion,(void*)segunda) != 0 )
59      { printf("Error\n"); return -1; }
60    pthread_join(hilo1,NULL);
61    pthread_join(hilo2,NULL);
62
63    uswtime(&utime1, &stime1, &wtime1);
64
65    printf("\nBusqueda Binaria con %d numeros, buscando el %d\n",n,bus);
66
67    ImprimirTiempos(utime0, stime0, wtime0,utime1, stime1, wtime1);
68    printf("-----------------------------------------------------\n");
69
70    return 0;
71  }
72
73  struct Arbol *AgregarElemento(struct Arbol *raiz, int dato)
74  {
75    if(raiz == NULL) // Caso base
76    {
77      struct Arbol *nuevo = NULL;
78      nuevo = (struct Arbol *) malloc(sizeof(struct Arbol));
79      nuevo -> dato = dato;
80      nuevo -> izq = NULL;
81      nuevo -> der = NULL;
82      return nuevo;
83    }
84
85    if(dato < raiz -> dato){
86      raiz -> izq = AgregarElemento(raiz -> izq, dato);
87    }
88    else
89    {
90      raiz -> der = AgregarElemento(raiz -> der, dato);
91    }
92
93    return raiz;
94  }
95
96  void InOrden(struct Arbol *arbol, int *numeros)
97  {
98    if(arbol == NULL)
99      return;
100
101   InOrden(arbol -> izq,numeros);
102   *(numeros + count) = arbol -> dato;
103   count++;
104   //printf("%d\n",arbol -> dato);
105   InOrden(arbol -> der,numeros);
106 }
107
108 void uswtime(double *usertime, double *systime, double *walltime)
109 {
110   double mega = 1.0e-6;
111   struct rusage buffer;
112   struct timeval tp;
113   struct timezone tzp;
114   getrusage(RUSAGE_SELF, &buffer);
115   gettimeofday(&tp, &tzp);
116   *usertime = (double) buffer.ru_utime.tv_sec +1.0e-6 * buffer.ru_utime.
        tv_usec;
```

```
117   *systime  = (double) buffer.ru_stime.tv_sec +1.0e-6 * buffer.ru_stime.
          tv_usec;
118   *walltime = (double) tp.tv_sec + 1.0e-6 * tp.tv_usec;
119 }
120
121 void ImprimirTiempos(double utime0,double stime0,double wtime0,double utime1
        ,double stime1,double wtime1)
122 {
123   printf("\n");
124   printf("Tiempo: %.10f\n", wtime1 - wtime0);
125 }
126 void buscar(int *numeros, int bus, int tam, int inicio)
127 {
128   cuenta = 1;
129   int mid = inicio + (tam - inicio) / 2;
130
131   if (*(numeros + inicio) == bus)
132     return;
133   if (*(numeros + mid) == bus)
134     return;
135   if (*(numeros + tam) == bus)
136     return;
137
138   if(*(numeros + mid) > bus)
139     tam = mid - 1;
140   else
141     inicio = mid + 1;
142
143   while(inicio <= tam)
144   {
145     cuenta++;
146     mid = inicio + (tam - inicio) / 2;
147     if (*(numeros + inicio) == bus)
148       return;
149     if (*(numeros + mid) == bus)
150       return;
151     if (*(numeros + tam) == bus)
152       return;
153
154     if(*(numeros + mid) > bus)
155       tam = mid - 1;
156     else
157       inicio = mid + 1;
158   }
159   return;
160 }
161
162 void *funcion(void *numeros){
163   buscar((int *)numeros,bus,n,0);
164 }
```

12.6. Código: Busqueda Exponencial

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/resource.h>
#include <sys/time.h>
#include <pthread.h>

typedef struct elem{
    int *arr1;
    int n1;
    int x;
    int result;
}*Elem;

int res;

void uswtime(double *usertime, double *systime, double *walltime);



double utime0, stime0, wtime0,utime1, stime1, wtime1;

int binarySearch(int Arr[], int l, int r, int x);
//int arr[], int n, int x
void *exponentialSearch(void * arg)
{
    //printf("llego a binary %d", ((Elem)arg)->n1);
    //((Elem)arg)->n1=50;

    int min=0;
    // If x is present at firt location itself
    if (((Elem)arg)->arr1[0] == ((Elem)arg)->x)
        return 0;

    // Find range for binary search by
    // repeated doubling
    int i = 1;
    while (i < ((Elem)arg)->n1 && ((Elem)arg)->arr1[i] <= ((Elem)arg)->x)
        i = i*2;

    if(i<=((Elem)arg)->n1-1) min=i;
    else min=((Elem)arg)->n1-1;


    //  Call binary search for the found range.
    //encargar a dos hilos de la busqeuda L - C -R
    uswtime(&utime0, &stime0, &wtime0);
    ((Elem)arg)->result=binarySearch(((Elem)arg)->arr1, i/2, min, ((Elem)arg
       )->x);
    uswtime(&utime1, &stime1, &wtime1);
}

int binarySearch(int arr[], int l, int r, int x)
{

    if (r >= l)
    {
        int mid = l + (r - l)/2;

        // If the element is present at the middle
```

```
59          // itself
60          if (arr[mid] == x)
61              return mid;
62
63          // If element is smaller than mid, then it
64          // can only be present n left subarray
65          if (arr[mid] > x)
66              return binarySearch(arr, l, mid-1, x);
67
68          // Else the element can only be present
69          // in right subarray
70          return binarySearch(arr, mid+1, r, x);
71      }
72
73      // We reach here when element is not present
74      // in array
75      return -1;
76  }
77
78  //Librerias para medir el tiempo
79
80  void uswtime(double *usertime, double *systime, double *walltime)
81  {
82    double mega = 1.0e-6;
83    struct rusage buffer;
84    struct timeval tp;
85    struct timezone tzp;
86    getrusage(RUSAGE_SELF, &buffer);
87    gettimeofday(&tp, &tzp);
88    *usertime = (double) buffer.ru_utime.tv_sec +1.0e-6 * buffer.ru_utime.
        tv_usec;
89    *systime  = (double) buffer.ru_stime.tv_sec +1.0e-6 * buffer.ru_stime.
        tv_usec;
90    *walltime = (double) tp.tv_sec + 1.0e-6 * tp.tv_usec;
91  }
92
93
94  //funcion para imprimir tiempos en linux
95  void ImprimirTiempos(double utime0,double stime0,double wtime0,double utime1
        ,double stime1,double wtime1){
96    printf("\n");
97    printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
98    printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0
        );
99    printf("sys (Tiempo en acci     nes de E/S)  %.10f s\n",  stime1 -
        stime0);
100   printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0
        ) / (wtime1 - wtime0));
101   printf("\n");
102
103   //Mostrar los tiempos en formato exponecial
104   printf("\n");
105   printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
106   printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1 - utime0
        );
107   printf("sys (Tiempo en acci     nes de E/S)  %.10e s\n",  stime1 -
        stime0);
108   printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0
        ) / (wtime1 - wtime0));
109   printf("\n");
110 }
```

```
111
112
113
114 int main(int argc, char *argv[]){
115
116     //int arr[]={2,3,4,10,40};
117     pthread_t hilo0, hilo1, hilo2;
118     Elem e1=malloc(sizeof(Elem));
119
120     e1->arr1=(int *)malloc(sizeof(int)*10000000);
121     //e1->n1=sizeof(e1->arr1)/sizeof(e1->arr1[0]);
122
123     int cantidad=atoi(argv[1]);//cantidad de numeros a leer
124     //int *arr=(int *)malloc(sizeof(int)*10000000);
125    // int n = sizeof(arr)/sizeof(arr[0]);
126     int var=0;
127     e1->n1=cantidad;
128
129     for(int j = 0; j < cantidad ; j++){
130     scanf("%d",&var);
131     e1->arr1[j]=var;
132     }
133     //7772647
134     e1->x=119847072;
135
136     //****************************
137     //Lamada al hilo y funciones de tiempo
138     //*************************
139
140     //uswtime(&utime0, &stime0, &wtime0);
141     //int result = exponentialSearch(e1->arr1, e1->n1, x);
142     pthread_create(&hilo0, NULL, exponentialSearch, e1);
143     //uswtime(&utime1, &stime1, &wtime1);
144
145     pthread_join(hilo0, NULL);
146
147     printf("Soy el chafa\n");
148     printf("\n%d", e1->result);
149     (e1->result == -1)? printf("Elemento %d isnt in array\n", e1->x)
150                       : printf("Element %d is at %d index \n",e1->x, e1->
                             result);
151
152
153     ImprimirTiempos(utime0, stime0, wtime0,utime1, stime1, wtime1);
154
155
156     return 0;
157 }
```

12.7. Código: Busqueda Exponencial con Hilos

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/resource.h>
#include <sys/time.h>
#include <pthread.h>

typedef struct elem{
    int *arr1;
    int n1;
    int x;
    int result;
}*Elem;

typedef struct rangos{
    int l;
    int r;
}*Rangos;

typedef struct args{
    Elem e2;
    Rangos r1;
} *Args;


double utime0, stime0, wtime0,utime1, stime1, wtime1;

void uswtime(double *usertime, double *systime, double *walltime);

void *binarySearch(void *arg);
//int arr[], int n, int x
void *exponentialSearch(void * arg)
{
    pthread_t hilo1, hilo2;
    int min=0;
    // If x is present at firt location itself
    if (((Elem)arg)->arr1[0] == ((Elem)arg)->x)
        return 0;

    // Find range for binary search by
    // repeated doubling
    int i = 1;
    while (i < ((Elem)arg)->n1 && ((Elem)arg)->arr1[i] <= ((Elem)arg)->x)
        i = i*2;

    if(i<=((Elem)arg)->n1-1) min=i;
    else min=((Elem)arg)->n1-1;

    Args parte1=malloc(sizeof(Args));
    Args parte2=malloc(sizeof(Args));

    parte1->r1=malloc(sizeof(Rangos));
    parte2->r1=malloc(sizeof(Rangos));


    parte1->e2=&(*((Elem)arg));
    parte2->e2=&(*((Elem)arg));


    int div=i/2;
```

```
60
61      int mitad= div + (min - div)/2;
62
63
64
65
66      parte1->r1->l=div;
67      parte1->r1->r=mitad;
68
69      parte2->r1->l=mitad;
70      parte2->r1->r=min;
71
72
73      printf("intervalos [%d,%d]  [%d, %d]",div,mitad,mitad, min );
74      //  Call binary search for the found range.
75      //encargar a dos hilos de la busqeuda L - C -R
76      // ((Elem)arg)->result=
77
78      //hilo1
79      uswtime(&utime0, &stime0, &wtime0);
80
81
82      pthread_create(&hilo1, NULL, binarySearch, parte1);
83
84      pthread_create(&hilo2, NULL, binarySearch, parte2);
85      uswtime(&utime1, &stime1, &wtime1);
86     //binarySearch(((Elem)arg)->arr1, i/2, min, ((Elem)arg)->x);
87      pthread_join(hilo1, NULL);
88      pthread_join(hilo2, NULL);
89  }
90
91
92  //argumentos int arr[], int l, int r, int x
93  void *binarySearch(void *arg)
94  {
95      Args aux=&(*((Args)arg));
96      int mid;
97
98      int le=aux->r1->l;
99      int ri=aux->r1->r;
100     while (ri >= le)
101     {
102          mid = le + (ri - le)/2;
103
104         // If the element is present at the middle
105         // itself
106         if ((aux->e2->arr1[mid]) == (aux->e2->x)){
107             ((Args)arg)->e2->result=mid;
108
109             //printf("termine %d", mid);
110             return 0;
111         }
112
113
114         // If element is smaller than mid, then it
115         // can only be present n left subarray
116         if (aux->e2->arr1[mid] > aux->e2->x){
117             ri=mid-1;
118             //return binarySearch(arr, l, mid-1, x);
119         }
120         else{
```

```
121             le=mid+1;
122             //return binarySearch(arr, mid+1, r, x);
123         }
124 
125         // Else the element can only be present
126         // in right subarray
127 
128 
129     }
130     // We reach here when element is not present
131     // in array
132     return 0;
133 }
134 
135 //Librerias para medir el tiempo
136 
137 void uswtime(double *usertime, double *systime, double *walltime)
138 {
139   double mega = 1.0e-6;
140   struct rusage buffer;
141   struct timeval tp;
142   struct timezone tzp;
143   getrusage(RUSAGE_SELF, &buffer);
144   gettimeofday(&tp, &tzp);
145   *usertime = (double) buffer.ru_utime.tv_sec +1.0e-6 * buffer.ru_utime.
      tv_usec;
146   *systime  = (double) buffer.ru_stime.tv_sec +1.0e-6 * buffer.ru_stime.
      tv_usec;
147   *walltime = (double) tp.tv_sec + 1.0e-6 * tp.tv_usec;
148 }
149 
150 
151 //funcion para imprimir tiempos en linux
152 void ImprimirTiempos(double utime0,double stime0,double wtime0,double utime1
    ,double stime1,double wtime1){
153   printf("\n");
154   printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
155   printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0
      );
156   printf("sys (Tiempo en acci     nes de E/S)  %.10f s\n",  stime1 -
      stime0);
157   printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0
      ) / (wtime1 - wtime0));
158   printf("\n");
159 
160   //Mostrar los tiempos en formato exponecial
161   printf("\n");
162   printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
163   printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1 - utime0
      );
164   printf("sys (Tiempo en acci     nes de E/S)  %.10e s\n",  stime1 -
      stime0);
165   printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0
      ) / (wtime1 - wtime0));
166   printf("\n");
167 }
168 
169 
170 
171 int main(int argc, char *argv[]){
172 
```

```
173     //int arr[]={2,3,4,10,40};
174     pthread_t hilo0;
175     Elem e1=malloc(sizeof(Elem));
176 
177     e1->arr1=(int *)malloc(sizeof(int)*10000000);
178     //e1->n1=sizeof(e1->arr1)/sizeof(e1->arr1[0]);
179 
180     int cantidad=atoi(argv[1]);//cantidad de numeros a leer
181     //int *arr=(int *)malloc(sizeof(int)*10000000);
182    // int n = sizeof(arr)/sizeof(arr[0]);
183     int var=0;
184     e1->n1=cantidad;
185 
186     for(int j = 0; j < cantidad ; j++){
187     scanf("%d",&var);
188     e1->arr1[j]=var;
189     }
190 
191     e1->x=10406;
192 
193     //*****************************
194     //Lamada al hilo y funciones de tiempo
195     //*************************
196 
197     //uswtime(&utime0, &stime0, &wtime0);
198     //int result = exponentialSearch(e1->arr1, e1->n1, x);
199     //pthread_create(&hilo0, NULL, exponentialSearch, e1);
200     exponentialSearch(e1);
201     //pthread_join(hilo0, NULL);
202 
203     //uswtime(&utime1, &stime1, &wtime1);
204 
205 
206     (e1->result ==-1)? printf("Elemento %d isnt in array\n", e1->x)
207                      : printf("Element %d is at %d index \n",e1->x, e1->
                            result);
208 
209     ImprimirTiempos(utime0, stime0, wtime0,utime1, stime1, wtime1);
210 
211 
212     return 0;
213 }
```

12.8. Código: Busqueda Fibonacci

```
1
2  //Compilaci n: "gcc main.c tiempo.c
3  //Ejecuci n: "./script.sh
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include "tiempo.h"
9
10 //Constantes del programa
11 #define MIN(a, b) (((a) < (b)) ? (a) : (b))
12
13 //Funciones
14 int busquedaFibonacci(int A[], int n, int x);
15
16 //Programa Principal
17 int main(int argc, char *argv[])
18 {
19   //Variables
20   int n;  //n determina el tama o del algorito dado por argumento al
         ejecutar
21   int i, j, bandera = 0; //Variables para loops
22   int *A;
23   int numeros[20] = {322468, 14700764, 3128036, 6337399, 61396, 10393545,
         2147445644, 129539003, 450057883, 187645041, 1980098116, 152503, 5000,
         1493283650, 214826, 1843349527, 1360839354, 2109248666, 2147470852, 0};
24   n = atoi(argv[1]);
25
26   int encontrados;
27
28   //Recepci n y decodificaci n de argumentos
29   A = (int *)malloc(n * sizeof(int));
30   for (i = 0; i < n; i++)
31   {
32     scanf("%d", &A[i]);
33   }
34
35   //Algoritmo
36   for (i = 0; i < 20; i++)
37   {
38     double utime0, stime0, wtime0, utime1, stime1, wtime1; //Variables para
          medici n de tiempos
39     uswtime(&utime0, &stime0, &wtime0);
40     encontrados = busquedaFibonacci(A, n - 1, numeros[i]);
41
42     if (encontrados != -1)
43     {
44       printf("Se encontro el numero %d\n", numeros[i]);
45     }
46     else
47     {
48       printf("No se encontro %d", numeros[i]);
49     }
50
51
52     //Evaluar los tiempos de ejecuci n
53     uswtime(&utime1, &stime1, &wtime1);
54
55     //C lculo del tiempo de ejecuci n del programa
```

```
56      printf("\n");
57      printf("real (Tiempo total)  %.10f s\n", wtime1 - wtime0);
58      printf("user (Tiempo de procesamiento en CPU) %.10f s\n", utime1 -
            utime0);
59      printf("sys (Tiempo en acci nes de E/S)  %.10f s\n", stime1 - stime0);
60      printf("CPU/Wall   %.10f %% \n", 100.0 * (utime1 - utime0 + stime1 -
            stime0) / (wtime1 - wtime0));
61      printf("\n");
62
63      //Mostrar los tiempos en formato exponecial
64      printf("\n");
65      printf("real (Tiempo total)  %.10e s\n", wtime1 - wtime0);
66      printf("user (Tiempo de procesamiento en CPU) %.10e s\n", utime1 -
            utime0);
67      printf("sys (Tiempo en acci nes de E/S)  %.10e s\n", stime1 - stime0);
68      printf("CPU/Wall   %.10f %% \n", 100.0 * (utime1 - utime0 + stime1 -
            stime0) / (wtime1 - wtime0));
69      printf("\n");
70      printf("
            ------------------------------------------------------------------------------
            ");
71      //Terminar programa normalmente
72    }
73
74    return 0;
75  }
76
77
78  //Busqueda Fibonacci, busca el valor de x en el arreglo
79
80  int busquedaFibonacci(int A[], int n, int x)
81  {
82    // Inicializamos los numeros de fibonacci
83    int fibMMm2 = 0;        // (m-2)'th n mero de fibonacci
84    int fibMMm1 = 1;        // (m-1)'th n mero de fibonacci
85    int fibM = fibMMm2 + fibMMm1; // m n mero de fibonacci
86
87    // fibM Almacenara el menor n mero de fib.
88    while (fibM < n)
89    {
90      fibMMm2 = fibMMm1;
91      fibMMm1 = fibM;
92      fibM = fibMMm2 + fibMMm1;
93    }
94
95    // se marca el rango de eliminacion.
96    int offset = -1;
97
98    // Cuando fibM se convierte en 1, fibMm2 se convierte en 0
99    while (fibM > 1)
100   {
101     // Busca ubicacion valida
102     int i = MIN(offset + fibMMm2, n - 1);
103
104     // Comprueba si fibMm2 es valido
105     if (A[i] < x)
106     {
107       fibM = fibMMm1;
108       fibMMm1 = fibMMm2;
109       fibMMm2 = fibM - fibMMm1;
110       offset = i;
```

```
111     }
112
113     // si x es mayor que el indice de fibMm2,se corta el subarreglo i+1
114     else if (A[i] > x)
115     {
116       fibM = fibMMm2;
117       fibMMm1 = fibMMm1 - fibMMm2;
118       fibMMm2 = fibM - fibMMm1;
119     }
120
121     // elemento encontrado, retorna indice
122     else
123       return i;
124   }
125
126   // compara el  ltimo  elemento con x
127   if (fibMMm1 && A[offset + 1] == x)
128     return offset + 1;
129
130   // Si no encuentra el elemento retorna -1
131   return -1;
132 }
```

12.9. Código: Busqueda Lineal

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/resource.h>
#include <sys/time.h>

int search(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

void uswtime(double *usertime, double *systime, double *walltime)
{
  double mega = 1.0e-6;
  struct rusage buffer;
  struct timeval tp;
  struct timezone tzp;
  getrusage(RUSAGE_SELF, &buffer);
  gettimeofday(&tp, &tzp);
  *usertime = (double) buffer.ru_utime.tv_sec +1.0e-6 * buffer.ru_utime.
     tv_usec;
  *systime  = (double) buffer.ru_stime.tv_sec +1.0e-6 * buffer.ru_stime.
     tv_usec;
  *walltime = (double) tp.tv_sec + 1.0e-6 * tp.tv_usec;
}

//Funcion para Imprimir los tiempos.
void ImprimirTiempos(double utime0,double stime0,double wtime0,double utime1
    ,double stime1,double wtime1){
  printf("\n");
  /*printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
  printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0
     );
  printf("sys (Tiempo en acci nes de E/S)  %.10f s\n",  stime1 - stime0);
  printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0
     ) / (wtime1 - wtime0));*/
  printf("Tiempo: %.10f\n", wtime1 - wtime0);
}


int main(int argc, char const *argv[])
{

  //Numero de numeros que se ordenar n
  int n = atoi(argv[1]);
  int *arr = (int*) malloc(sizeof(int)*n);

    int x = atoi(argv[2]);
    double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para
        m e d i c i n  de tiempos.

    //For para poder leer las entradas.
  for(int i = 0; i < n; i++){

    scanf("%d",&arr[i]);

```

```
54    }
55
56
57      uswtime(&utime0, &stime0, &wtime0); //Se llama la funcion para los
            tiempos.
58
59      int result = search(arr, n, x); //LLAMADA A LA FUNCION DEL ALGORITMO.
60
61
62      uswtime(&utime1, &stime1, &wtime1); //Se llama la funcion para los
            tiempos.
63      if(result == -1)
64          printf("Con %d numeros, el numero %d no esta en el arreglo \n", n, x
                );
65      else
66          printf("Con %d numeros, el numero %d esta en el lugar %d \n", n , x,
                result);
67
68      ImprimirTiempos(utime0, stime0, wtime0,utime1, stime1, wtime1);
69    printf("
        ------------------------------------------------------------------------\
        n");
70
71      return 0;
72  }
```

12.10. Código: Busqueda Lineal con Hilos

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/resource.h>
#include <sys/time.h>
#include <pthread.h>

int n, x, mitad;
int *arr;
int bandera = 0;


void* procesar(void* id)
{
  int n_thread=(int)id;

  if(n_thread==1){

  for (int i = 0; i < mitad; i++)
        if (arr[i] == x)
      bandera = 1;
  }

  else{
    for (int i = mitad; i < n; i++)
        if (arr[i] == x)
          bandera = 1;
  }

}

void uswtime(double *usertime, double *systime, double *walltime)
{
  double mega = 1.0e-6;
  struct rusage buffer;
  struct timeval tp;
  struct timezone tzp;
  getrusage(RUSAGE_SELF, &buffer);
  gettimeofday(&tp, &tzp);
  *usertime = (double) buffer.ru_utime.tv_sec +1.0e-6 * buffer.ru_utime.
     tv_usec;
  *systime  = (double) buffer.ru_stime.tv_sec +1.0e-6 * buffer.ru_stime.
     tv_usec;
  *walltime = (double) tp.tv_sec + 1.0e-6 * tp.tv_usec;
}

//Funcion para Imprimir los tiempos.
void ImprimirTiempos(double utime0,double stime0,double wtime0,double utime1
    ,double stime1,double wtime1){
  printf("\n");
  /*printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
  printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0
     );
  printf("sys (Tiempo en acci nes de E/S)  %.10f s\n",  stime1 - stime0);
  printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0
     ) / (wtime1 - wtime0));*/
  printf("Tiempo: %.10f\n", wtime1 - wtime0);
}


```

```
55 int main(int argc, char const *argv[])
56 {
57
58   //Numero de numeros que se ordenar n
59   n = atoi(argv[1]);
60   arr = (int*) malloc(sizeof(int)*n);
61
62     x = atoi(argv[2]);
63     double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para
           m e d i c i n  de tiempos.
64
65     //For para poder leer las entradas.
66   for(int i = 0; i < n; i++){
67
68     scanf("%d",&arr[i]);
69
70   }
71
72   pthread_t thread1;
73     pthread_t thread2;
74     int i = 1;
75     int k = 2;
76     mitad = n/2;
77
78     uswtime(&utime0, &stime0, &wtime0); //Se llama la funcion para los
          tiempos.
79
80     if (pthread_create (&thread1, NULL, procesar,(void*)i) != 0 )
81     {
82       perror("El thread no  pudo crearse");
83       exit(-1);
84     }
85
86   if (pthread_create (&thread2, NULL, procesar,(void*)k) != 0 )
87     {
88       perror("El thread no  pudo crearse");
89       exit(-1);
90     }
91
92   //Esperar a que terminen los threads
93   pthread_join(thread1, NULL);
94   pthread_join(thread2, NULL);
95
96     uswtime(&utime1, &stime1, &wtime1); //Se llama la funcion para los
          tiempos.
97
98     if(bandera == 0)
99         printf("Con %d numeros, el numero %d no esta en el arreglo \n", n, x
              );
100    else
101        printf("Con %d numeros, el numero %d esta en el arreglo \n", n , x);
102
103    ImprimirTiempos(utime0, stime0, wtime0,utime1, stime1, wtime1);
104  printf("
       ------------------------------------------------------------------------\
       n");
105
106    return 0;
107 }
```

13. Bibliografía

- Scalise, E., Carmona, R. (2001). Análisis de Algoritmos y Complejidad.
- Nishihara, S., Nishino, H. (1987). Binary search revisited: Another advantage of Fibonacci search. IEEE transactions on computers, 100(9), 1132-1135.
- "GeeksforGeeks — A computer science portal for geeks", GeeksforGeeks, 2020. [Online]. Available: https://www.geeksforgeeks.org/. [Accessed: 29- April - 2021].
- "Plataforma LMS del Prof. Edgardo Adrián Franco Martínez: Ingresar al sitio", Eafranco.eakdemy.com, 2020. [Online]. Available: https://eafranco.eakdemy.com/mod/page/view.php?id=148. [Accessed: 29- April - 2021].
- Boyer, R. S., Moore, J. S. (1977). A fast string searching algorithm. Communications of the ACM, 20(10), 762-772.