Instituto Politécnico Nacional

Escuela Superior de Cómputo

Análisis de Algoritmos

Práctica 02

Gurrola Sánchez Joel.

Profesor: Franco Martinez Edgardo Adrian

3CM11

Índice

Definición del problema

Con base en el archivo de entrada de la práctica 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 a posteriori (empírico) de las complejidades.

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

*Utilizar arreglos ordenados.

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.

Entorno de pruebas

Modelo de hardware: Lenovo ideapad 520s-14IKB
Memoria ram: 8.0 GB
Procesador: Intel Core i5-7200Ux4
Graficos: Mesa Intel HD Graphics 620
Capacidad del disco: 480 GB

1.- Análisis teórico de casos y funciones de complejidad temporal

Búsqueda lineal o secuencial

Código

```
int busquedaLineal(int *Arreglo, int tam, int num){
    for (int i = 0; i < tam; i++) {
        if (Arreglo[i] == num) {
            return i;  // Devuelve el índice donde se encontró el elemento
        }
    }
    return -1; //Devuelve -1 si no encontro el elemento
}
```

Mejor caso

Este caso se da cuando el número se encuentra en la posición 0 del arreglo.

$$f_t(n) = 1$$

Peor caso

Este caso se presenta cuando el número se encuentra en la última posición del arreglo, o no se encuentra en él.

$$f_t(n) = n$$

Caso medio

El caso medio está dado por:

$$f_t(n) = \sum_{i=1}^{k} O(i)P(\bar{I})$$

Donde:

I(n)={X se encuentra en la 1° posición, X se encuentra en la 2° posición, X se encuentra en la 3° posición, ..., X se encuentra en la n° posición, X no se encuentra en el arreglo}

$$O(n) = \{1,\ 2,\ 3,\ \ldots,\ n,\ n\}$$

Teniendo en cuenta que las instancias son equiprobables:

$$P(n) = \{\frac{1}{n+1}, \frac{1}{n+1}, \frac{1}{n+1}, \ldots, \frac{1}{n+1}, \frac{1}{n+1}\}$$

Desarrollamos:

$$f_t(n) = (1)\left(\frac{1}{n+1}\right) + (2)\left(\frac{1}{n+1}\right) + (3)\left(\frac{1}{n+1}\right) + \ldots + (n)\left(\frac{1}{n+1}\right) + (n)\left(\frac{1}{n+1}\right)$$

$$f_t(n) = \left(\frac{1}{n+1}\right)(1 + 2 + 3 + \ldots + n + n) = \left(\frac{1}{n+1}\right)\left(\frac{n(n+1)}{2} + n\right)$$

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

Tenemos como resultado lo siguiente:

$$f_t(n) = \frac{n}{2} + 1 - \frac{2}{2(n+1)}$$

Búsqueda binaria o dicotómica

Código

```
int busquedaBinaria(int *arr, int n, int x)
{
    int anterior = 0;
    int siguiente = n - 1;
    int centro;

    while (anterior <= siguiente)
    {
        centro = anterior + (siguiente - anterior) / 2;
        if (arr[centro] == x)
            return centro;
        if (arr[centro] < x)
            anterior = centro + 1;
        else
            siguiente = centro - 1;
    }

    return -1;
}
```

Mejor caso

Este caso se presenta cuando el número se encuentra a la mitad del rango inicial.

$$f_t(n) = 2$$

Peor caso

Este caso se presenta cuando el número no se encuentra en el rango a buscar.

$$f_t(n) = 4n$$

Caso medio

El caso medio está dado por:

$$f_t(n) = \sum_{i=1}^{k} O(i)P(i)$$

Donde:

I(n)={X se encuentra en el 1er paso, X se encuentra en el 2do paso, X se encuentra en el 3er, X se encuentra en el 4to, …, X se encuentra en el n paso, X no se encuentra en el rango}

Como las instancias son equiprobables de suceder:

$$P(n) = \{\frac{1}{n+1}, \frac{1}{n+1}, \frac{1}{n+1}, \ldots, \frac{1}{n+1}, \frac{1}{n+1}\}$$

Calculando tenemos:

$Sea\ m = n$

$$f_t(n) = (4(0) + 2)\left(\frac{1}{m+1}\right) + (4(1) + 2)\left(\frac{1}{m+1}\right) + (4(2) + 2)\left(\frac{1}{m+1}\right) + \ldots + (4(m-1) + 2)\left(\frac{1}{m+1}\right) + (4m)\left(\frac{1}{m+1}\right)$$

$$f_t(n) = \left(\frac{1}{m+1}\right)(4(0) + 2 + 4(1) + 2 + 4(2) + 2 + \ldots + 4(m-1) + 2 + 4m)$$

$$f_t(n) = \left(\frac{1}{m+1}\right)(2m + 4m + 4(0) + 4(1) + 4(2) + \ldots + 4(m-1))$$

$$f_t(n) = \left(\frac{1}{m+1}\right)(6m + 4(0 + 1 + 2 + \ldots + m - 1))$$

$$f_t(n) = \left(\frac{1}{m+1}\right)\left(6m + 4\frac{(m-1)m}{2}\right) = \left(\frac{1}{m+1}\right)(6m + 2(m-1)m)$$

$$f_t(n) = \frac{6m+2m^2-2m}{m+1} = \frac{2m^2+4m}{m+1};$$

Por lo tanto el peor caso está dado por:

$$f_t(n) = 2n + 2$$

Búsqueda exponencial

Código

```
int busquedaExponencial(int arr[], int n, int x)
{
    int i = 0;
   if (arr[i] == x)
      return i;
   i = 1;
   while (i < n && arr[i] <= x)
   {
      i = i * 2;
   }

   int anterior = i / 2;
   int siguiente = obtener_menor(i, n - 1);
   int centro;

   while (anterior <= siguiente)
   {
      centro = anterior + (siguiente - anterior) / 2;
      if (arr[centro] == x)
         return centro;
      if (arr[centro] < x)
         anterior = centro + 1;
      else
         siguiente = centro - 1;
   }

   return -1;
}
```

Mejor caso

Su mejor caso se presenta cuando el número se encuentra en la posición 0 del arreglo.

$$f_t(n) = 1$$

Peor caso

El peor caso se presenta cuando el número es menor que el arreglo en la posición n-1 y a su vez mayor que el arreglo en la posición (n-1)/2, por lo tanto no se encontrará.

$$f_t(n) = (1) + (4n + 3) + (4\frac{n}{2})$$
$$f_t(n) = (1) + (4n + 3) + (4n - 42)$$

$$f_t(n) = (1) + (4n + 3) + (4n - 4)$$
$$f_t(n) = 8n$$

Caso medio

Está dado por:

$$f_t(n) = \sum_{i=1}^{k} O(i)P(i)$$

Donde:

I(n)={X se encuentra al inicio, El while se rompe con i=1, El while se rompe con i=2, El while se rompe con i=4, El while se rompe con i=8, …, El while se rompe con i=n

Búsqueda de Fibonacci

Código

```
int busquedaFibonacci(int *arr, int x, int n)
{
    int ultimo_fibo = 1;
    int penultimo_fibo = 0;
    int siguiente_fibo = penultimo_fibo + ultimo_fibo;

    while (siguiente_fibo < n)
    {
        penultimo_fibo = ultimo_fibo;
        ultimo_fibo = siguiente_fibo;
        siguiente_fibo = penultimo_fibo + ultimo_fibo;
    }

    int limite = 0;

    while (siguiente_fibo > 1)
    {
        int i = min(limite + penultimo_fibo, n) - 1;

        if (arr[i] < x)
        {
            siguiente_fibo = ultimo_fibo;
            ultimo_fibo = penultimo_fibo;
            penultimo_fibo = siguiente_fibo - ultimo_fibo;
            limite = i + 1;
        }
        else
        {
            if (arr[i] > x)
            {
                siguiente_fibo = penultimo_fibo;
                ultimo_fibo = ultimo_fibo - penultimo_fibo;
                penultimo_fibo = siguiente_fibo - ultimo_fibo;
            }
            else
            {
                return i;
            }
        }
    }
    if (ultimo_fibo && arr[limite] == x)
        return limite;
    return -1;
}
```

Mejor caso

En este caso el primer while se ejecuta tantas veces como números de Fibonacci existan menores o iguales al número, el segundo while se ejecuta solo una vez por su camino más corto, entonces tenemos:

$$f_t(n) = 4[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}] + 4$$

Peor caso

En este caso el primer while se ejecuta tantas veces como números de Fibonacci existan menores o iguales al número, el segundo se ejecutará tantas veces como números de Fibonacci existan menores o iguales al número, por lo tanto tenemos:

$$f_t(n) = 4[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}] + 6[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}]$$

$$f_t(n) = 10[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}]$$

Caso medio

En el caso medio tomaremos en cuenta 3 caminos posibles:

Sus complejidades de cada uno son las siguientes:

$$f_{t\_C1}(n) = 4[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}] + 6[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}] = 10[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}]$$

$$f_{t\_C2}(n) = 4[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}] + 6[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}] = 10[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}]$$

$$f_{t\_C3}(n) = 4[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}] + 4[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}] = 8[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}]$$

$$f_t(n) = O_{t\_C1}P_{t\_C1} + O_{t\_C2}P_{t\_C2} + O_{t\_C3}P_{t\_C3}$$

$$f_t(n) = 10[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}]\left(\frac{1}{3}\right) + 10[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}]\left(\frac{1}{3}\right) + 8[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}]\left(\frac{1}{3}\right)$$

$$f_t(n) = \left(\frac{1}{3}\right)(10[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}] + 10[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}] + 8[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}])$$

Por lo tanto tenemos:

$$f_t(n) = \left(\frac{1}{3}\right)(28[\frac{\log log\, n + \frac{1}{2}\log log\, 5}{\log log\, \emptyset}])$$

2.- Implementación de los algoritmos de búsqueda en Lenguaje ANSI C

Búsqueda lineal o secuencial

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

int busquedaLineal(int *Arreglo, int tam, int num){
    for (int i = 0; i < tam; i++) {
        if (Arreglo[i] == num) {
            return i;  // Devuelve el índice donde se encontró el elemento
        }
    }
    return -1; //Devuelve -1 si no encontro el elemento
}

// PROGRAMA PRINCIPAL
int main(int argc, char *argv[])
{

    // Variables del main
    double utime0, stime0, wtime0, utime1, stime1, wtime1; // Variables para medición de tiempos
    int i; //variables para los ciclos
    int n; //variable para el tamaño del arreglo
    int tam, num, numEncontrado; //Variables para el algoritmo

    // Recepción y decodificación de argumentos
    if (argc >= 2)// Si se introducen 2 o mas argumentos, el numero a buscar y el tamaño del arreglo
    {
        n = atoi(argv[1]); //tamaño del arreglo
        if (argc > 2)
        {
            num = atoi(argv[2]); //a la variable num, le asignamos el numero a buscar
        }
        else
        {
           scanf("%d", &num);
        }
    }
    else
    {
        scanf("%d", &n);
        scanf("%d", &num);
    }
```

```
    //  Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    uswtime(&utime0, &stime0, &wtime0);


    int *Arreglo = malloc(n * sizeof(int));
    for (i = 0; i < n; i++)
    {
        scanf("%d", &Arreglo[i]);
    }


    // Algoritmo
    tam = n / sizeof(Arreglo[0]);

    numEncontrado = busquedaLineal(Arreglo, tam, num);


    // Evaluar los tiempos de ejecución
    uswtime(&utime1, &stime1, &wtime1);

    // Cálculo del tiempo de ejecución del programa
    printf("-- Tiempos con %d elementos ---\n", n);
    printf("Tiempo real (Tiempo total): %.10f segundos\n", wtime1 - wtime0);
    printf("Tiempo real (Tiempo total): %.10e segundos\n", wtime1 - wtime0);
    printf("\n");

    //Se imprimen los resultados
    if (numEncontrado == -1) {
        printf("El elemento %d no se encuentra en el arreglo.\n", num);
    } else {
        printf("El elemento %d se encuentra en la posicion %d del arreglo.\n", num,
numEncontrado);
    }

    printf("\n\n");

    free(Arreglo);
    // Terminar programa normalmente
    exit(0);
}
```

Búsqueda en un árbol binario de búsqueda

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

```
//Estructuras
struct nodo {
    int dato;
    struct nodo *izq, *der;
};

struct nodo* nuevoNodo(int dato) //creamos un nuvo nodo del arbol
{
    struct nodo* temp = (struct nodo*)malloc(sizeof(struct nodo));
    temp->dato = dato;
    temp->izq = temp->der = NULL;
    return temp;
}

struct nodo* insertar(struct nodo* nodo, int dato) //Insertar un dato
{
    if (nodo == NULL)
        return nuevoNodo(dato); //si esta vacio, sera la raiz


    if (dato < nodo->dato) //si el dato es menor al nodo padre, se inserta a la izquierda
        nodo->izq = insertar(nodo->izq, dato);
    else if (dato > nodo->dato)//si el dato es mayor al nodo padre, se inserta a la derecha
        nodo->der = insertar(nodo->der, dato);

    return nodo;
}


struct nodo* buscar(struct nodo* nodo, int dato) //Buscar en el arbol
{

    if (nodo == NULL || nodo->dato == dato) //Si el nodo esta vacio o el dato a buscar es el
mismo al dato en que se encuentra
        return dato; //Retornamos el mismo dato

    if (nodo->dato < dato) //Si el dato del nodo padre es menor al dato ingresado, buscamos
por la derecha
        return buscar(nodo->der, dato);

    return buscar(nodo->izq, dato); //Si no se cumple lo anterior, buscamos por la izquierda
}



// PROGRAMA PRINCIPAL
int main(int argc, char *argv[])
{

    // Variables del main
    double utime0, stime0, wtime0, utime1, stime1, wtime1; // Variables para mediciÃ³n de
tiempos
    int i; //variables para los ciclos
    int n; //variable para el tamaÃ±o del arreglo
```

```
    int num; //variable para guardar el numero a buscar
    int tmp, numEncontrado;
    struct nodo* arbol = NULL;

    // RecepciÃ³n y decodificaciÃ³n de argumentos
    if (argc >= 2)// Si se introducen 2 o mas argumentos, el numero a buscar y el tamaÃ±o
del arreglo
    {
        n = atoi(argv[1]); //tamaÃ±o del arreglo
        if (argc > 2)
        {
            num = atoi(argv[2]); //a la variable num, le asignamos el numero a buscar
        }
        else
        {
           scanf("%d", &num);
        }
    }
    else
    {
        scanf("%d", &n);
        scanf("%d", &num);
    }

    //  Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    uswtime(&utime0, &stime0, &wtime0);



    for (i = 0; i < n; i++)
    {
        scanf("%d", &tmp);
        insertar(arbol,tmp);
    }

    // Algoritmo

    /*if (buscar(arbol, num) == NULL)
        printf("Dato %d, no entontrado\n", num);
    else
        printf("Dato %d encontrado\n", num);
    */


    // Evaluar los tiempos de ejecuciÃ³n
    uswtime(&utime1, &stime1, &wtime1);

    printf("%.10f\n", wtime1 - wtime0);



    // Terminar programa normalmente
    exit(0);
}
```

Búsqueda binaria o dicotómica

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


int busquedaBinaria(int arr[], int l, int r, int x)
{
    while (l <= r) {
        int m = l + (r - l) / 2;

        if (arr[m] == x)
            return m;

        if (arr[m] < x)
            l = m + 1;

        else
            r = m - 1;
    }

    return -1;
}

// PROGRAMA PRINCIPAL
int main(int argc, char *argv[])
{

    // Variables del main
    double utime0, stime0, wtime0, utime1, stime1, wtime1; // Variables para medición de
tiempos
    int i; //variables para los ciclos
    int n; //variable para el tamaño del arreglo
    int tam, num, numEncontrado; //Variables para el algoritmo

    // Recepción y decodificación de argumentos
    if (argc >= 2)// Si se introducen 2 o mas argumentos, el numero a buscar y el tamaño
del arreglo
    {
        n = atoi(argv[1]); //tamaño del arreglo
        if (argc > 2)
        {
            num = atoi(argv[2]); //a la variable num, le asignamos el numero a buscar
        }
        else
        {
           scanf("%d", &num);
        }
```

```
    }
    else
    {
        scanf("%d", &n);
        scanf("%d", &num);
    }

    //  Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    uswtime(&utime0, &stime0, &wtime0);


    int *Arreglo = malloc(n * sizeof(int));
    for (i = 0; i < n; i++)
    {
        scanf("%d", &Arreglo[i]);
    }


    // Algoritmo
    numEncontrado = busquedaBinaria(Arreglo, 0, n - 1, num);

    if (numEncontrado == -1)
        printf("Elemento no encontrado");
    else
        printf("El elemento %d se encontro en la posicion: %d\n", num, numEncontrado);



    // Evaluar los tiempos de ejecución
    uswtime(&utime1, &stime1, &wtime1);

    // Cálculo del tiempo de ejecución del programa
    printf("-- Tiempos con %d elementos ---\n", n);
    printf("Tiempo real (Tiempo total): %.10f segundos\n", wtime1 - wtime0);

    // Mostrar los tiempos en formato exponencial
    printf("Tiempo real (Tiempo total): %.10e segundos\n", wtime1 - wtime0);
    printf("\n");

    free(Arreglo);
    // Terminar programa normalmente
    exit(0);
}
```

Búsqueda exponencial

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```c
#include "../tiempo.h"
#define min

int busquedaBinaria(int arr[], int, int, int);

int obtener_menor(int x, int y)
{
    return (x <= y) ? x : y;
}

int busquedaExponencial(int arr[], int n, int x)
{
     int i = 0;
    if (arr[i] == x)
        return i;
    i = 1;
    while (i < n && arr[i] <= x)
    {
        i = i * 2;
    }

    int anterior = i / 2;
    int siguiente = obtener_menor(i, n - 1);
    int centro;

    while (anterior <= siguiente)
    {
        centro = anterior + (siguiente - anterior) / 2;
        if (arr[centro] == x)
            return centro;
        if (arr[centro] < x)
            anterior = centro + 1;
        else
            siguiente = centro - 1;
    }

    return -1;
}


// PROGRAMA PRINCIPAL
int main(int argc, char *argv[])
{

    // Variables del main
    double utime0, stime0, wtime0, utime1, stime1, wtime1; // Variables para medición de
tiempos
    int i; //variables para los ciclos
    int n; //variable para el tamaño del arreglo
    int tam, num, numEncontrado; //Variables para el algoritmo

    // Recepción y decodificación de argumentos
    if (argc >= 2)// Si se introducen 2 o mas argumentos, el numero a buscar y el tamaño
del arreglo
```

```
    {
        n = atoi(argv[1]); //tamaño del arreglo
        if (argc > 2)
        {
            num = atoi(argv[2]); //a la variable num, le asignamos el numero a buscar
        }
        else
        {
           scanf("%d", &num);
        }
    }
    else
    {
        scanf("%d", &n);
        scanf("%d", &num);
    }

    //  Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    uswtime(&utime0, &stime0, &wtime0);


    int *Arreglo = malloc(n * sizeof(int));
    for (i = 0; i < n; i++)
    {
        scanf("%d", &Arreglo[i]);
    }

    tam = n / sizeof(Arreglo[0]);

    // Algoritmo
    numEncontrado =  busquedaExponencial(Arreglo, tam, num);

    if (numEncontrado == -1)
        printf("Elemento no encontrado");
    else
        printf("El elemento %d se encontro en la posicion: %d\n", num, numEncontrado);


    // Evaluar los tiempos de ejecución
    uswtime(&utime1, &stime1, &wtime1);

    printf("%.10f\n", wtime1 - wtime0);

    free(Arreglo);
    // Terminar programa normalmente
    exit(0);
}
```

Búsqueda de Fibonacci

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

int min(int x, int y) {
    return (x <= y) ? x : y;
}


int busquedaFibonacci(int *arr, int x, int n)
{
    int ultimo_fibo = 1;
    int penultimo_fibo = 0;
    int siguiente_fibo = penultimo_fibo + ultimo_fibo;

    while (siguiente_fibo < n)
    {
        penultimo_fibo = ultimo_fibo;
        ultimo_fibo = siguiente_fibo;
        siguiente_fibo = penultimo_fibo + ultimo_fibo;
    }

    int limite = 0;

    while (siguiente_fibo > 1)
    {
        int i = min(limite + penultimo_fibo, n) - 1;

        if (arr[i] < x)
        {
            siguiente_fibo = ultimo_fibo;
            ultimo_fibo = penultimo_fibo;
            penultimo_fibo = siguiente_fibo - ultimo_fibo;
            limite = i + 1;
        }
        else
        {
            if (arr[i] > x)
            {
                siguiente_fibo = penultimo_fibo;
                ultimo_fibo = ultimo_fibo - penultimo_fibo;
                penultimo_fibo = siguiente_fibo - ultimo_fibo;
            }
            else
            {
                return i;
            }
        }
    }
    if (ultimo_fibo && arr[limite] == x)
        return limite;
    return -1;
}
```

```
// PROGRAMA PRINCIPAL
int main(int argc, char *argv[])
{

    // Variables del main
    double utime0, stime0, wtime0, utime1, stime1, wtime1; // Variables para medición de
tiempos
    int i; //variables para los ciclos
    int n; //variable para el tamaño del arreglo
    int tam, num, numEncontrado; //Variables para el algoritmo

    // Recepción y decodificación de argumentos
    if (argc >= 2)// Si se introducen 2 o mas argumentos, el numero a buscar y el tamaño
del arreglo
    {
        n = atoi(argv[1]); //tamaño del arreglo
        if (argc > 2)
        {
            num = atoi(argv[2]); //a la variable num, le asignamos el numero a buscar
        }
        else
        {
           scanf("%d", &num);
        }
    }
    else
    {
        scanf("%d", &n);
        scanf("%d", &num);
    }

    //  Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    uswtime(&utime0, &stime0, &wtime0);


    int *Arreglo = malloc(n * sizeof(int));
    for (i = 0; i < n; i++)
    {
        scanf("%d", &Arreglo[i]);
    }

    tam = n / sizeof(Arreglo[0]);

    // Algoritmo
    numEncontrado = busquedaFibonacci(Arreglo, tam, num);

    /*if (numEncontrado >= 0)
        printf("El elemento %d se encontro en la posicion: %d\n", num, numEncontrado);
    else
        printf("Elemento no encontrado");

    */
```

```
    // Evaluar los tiempos de ejecución
    uswtime(&utime1, &stime1, &wtime1);

    // Cálculo del tiempo de ejecución del programa
    /*printf("-- Tiempos con %d elementos ---\n", n);
    printf("Tiempo real (Tiempo total): %.10f segundos\n", wtime1 - wtime0);
    printf("\n");

    // Mostrar los tiempos en formato exponencial
    printf("Tiempo real (Tiempo total): %.10e segundos\n", wtime1 - wtime0);
    printf("\n");
*/

    printf("%.10f\n", wtime1 - wtime0);
    free(Arreglo);
    // Terminar programa normalmente
    exit(0);
}
```

3.- Adaptaciones sobre los códigos

Las modificaciones del código estan subrayados con amarillo

Búsqueda lineal o secuencial

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

// Cambio en el tipo de la variable n para manejar grandes cantidades de elementos
long long int busquedaLineal(int *Arreglo, long long int tam, int num) {
    for (long long int i = 0; i < tam; i++) {
        if (Arreglo[i] == num) {
            return i;
        }
    }
    return -1;
}

int main(int argc, char *argv[]) {
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    long long int i, n; // Cambio en el tipo de la variable n
    int tam, num, numEncontrado;

    if (argc >= 2) {
        n = atoll(argv[1]); // Uso de atoll para convertir a long long int
        if (argc > 2) {
            num = atoi(argv[2]);
        } else {
            scanf("%d", &num);
        }
    } else {
        scanf("%lld", &n); // Uso de %lld para leer long long int
        scanf("%d", &num);
    }

    uswtime(&utime0, &stime0, &wtime0);

    int *Arreglo = malloc(n * sizeof(int));
    for (i = 0; i < n; i++) {
        scanf("%d", &Arreglo[i]);
    }

    tam = n;

    numEncontrado = busquedaLineal(Arreglo, tam, num);

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

    printf("-- Tiempos con %lld elementos ---\n", n);
    printf("Tiempo real (Tiempo total): %.10f segundos\n", wtime1 - wtime0);
```

```
    printf("Tiempo real (Tiempo total): %.10e segundos\n", wtime1 - wtime0);
    printf("\n");

    if (numEncontrado == -1) {
        printf("El elemento %d no se encuentra en el arreglo.\n", num);
    } else {
        printf("El elemento %d se encuentra en la posicion %lld del arreglo.\n", num,
numEncontrado);
    }

    free(Arreglo);

    exit(0);
}
```

Búsqueda en un árbol binario de búsqueda

```
  #include <stdio.h>
#include <stdlib.h>
#include "../tiempo.h"

// Estructuras
struct nodo {
    int dato;
    struct nodo *izq, *der;
};

struct nodo* nuevoNodo(int dato) {
    struct nodo* temp = (struct nodo*)malloc(sizeof(struct nodo));
    temp->dato = dato;
    temp->izq = temp->der = NULL;
    return temp;
}

struct nodo* insertar(struct nodo* nodo, int dato) {
    if (nodo == NULL)
        return nuevoNodo(dato);

    if (dato < nodo->dato)
        nodo->izq = insertar(nodo->izq, dato);
    else if (dato > nodo->dato)
        nodo->der = insertar(nodo->der, dato);

    return nodo;
}

struct nodo* buscar(struct nodo* nodo, int dato) {
    if (nodo == NULL || nodo->dato == dato)
        return nodo;
```

```
    if (nodo->dato < dato)
        return buscar(nodo->der, dato);

    return buscar(nodo->izq, dato);
}

// PROGRAMA PRINCIPAL
int main(int argc, char *argv[]) {
    // Variables para medición de tiempos
    double utime0, stime0, wtime0, utime1, stime1, wtime1;

    int i; // Variables para los ciclos
    long long int n; // Variable para el tamaño del arreglo (cambiado a long long int)
    int num; // Variable para guardar el numero a buscar
    int tmp;
    struct nodo* arbol = NULL;

    // Recepción y decodificación de argumentos
    if (argc >= 2) {
        n = atoll(argv[1]); // Cambio en el tipo de la variable n y uso de atoll
        if (argc > 2) {
            num = atoi(argv[2]);
        } else {
            scanf("%d", &num);
        }
    } else {
        scanf("%lld", &n); // Cambio en la lectura de n a %lld
        scanf("%d", &num);
    }

    // Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    uswtime(&utime0, &stime0, &wtime0);

    // Lectura de datos e inserción en el árbol
    for (i = 0; i < n; i++) {
        scanf("%d", &tmp);
        arbol = insertar(arbol, tmp);
    }

    // Algoritmo de búsqueda en el árbol
    /*if (buscar(arbol, num) == NULL)
        printf("Dato %d, no encontrado\n", num);
    else
        printf("Dato %d encontrado\n", num);
    */

    // Evaluar los tiempos de ejecución
    uswtime(&utime1, &stime1, &wtime1);

    // Cálculo del tiempo de ejecución del programa
    printf("-- Tiempos con %lld elementos ---\n", n);
    printf("Tiempo real (Tiempo total): %.10f segundos\n", wtime1 - wtime0);
```

```
    // Mostrar los tiempos en formato exponencial
    printf("Tiempo real (Tiempo total): %.10e segundos\n", wtime1 - wtime0);
    printf("\n");

    // Liberar memoria
    free(arbol);

    // Terminar programa normalmente
    exit(0);
}
```

Búsqueda binaria o dicotómica

```
#include <stdio.h>
#include <stdlib.h>
#include "../tiempo.h"

int busquedaBinaria(int arr[], int l, int r, int x) {
    while (l <= r) {
        int m = l + (r - l) / 2;

        if (arr[m] == x)
            return m;

        if (arr[m] < x)
            l = m + 1;
        else
            r = m - 1;
    }

    return -1;
}

int main(int argc, char *argv[]) {
    // Variables para medición de tiempos
    double utime0, stime0, wtime0, utime1, stime1, wtime1;

    int i; // Variables para los ciclos
    long long int n; // Variable para el tamaño del arreglo (cambiado a long long int)
    int num, numEncontrado; // Variables para el algoritmo

    // Recepción y decodificación de argumentos
    if (argc >= 2) {
        n = atoll(argv[1]); // Cambio en el tipo de la variable n y uso de atoll
        if (argc > 2) {
            num = atoi(argv[2]);
        } else {
            scanf("%d", &num);
        }
```

```
    } else {
        scanf("%lld", &n); // Cambio en la lectura de n a %lld
        scanf("%d", &num);
    }

    // Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    uswtime(&utime0, &stime0, &wtime0);

    int *Arreglo = malloc(n * sizeof(int));
    for (i = 0; i < n; i++) {
        scanf("%d", &Arreglo[i]);
    }

    // Algoritmo
    numEncontrado = busquedaBinaria(Arreglo, 0, n - 1, num);

    // Impresión de resultados
    if (numEncontrado == -1)
        printf("Elemento no encontrado");
    else
        printf("El elemento %d se encontro en la posicion: %d\n", num, numEncontrado);

    // Evaluar los tiempos de ejecución
    uswtime(&utime1, &stime1, &wtime1);

    // Cálculo del tiempo de ejecución del programa
    printf("-- Tiempos con %lld elementos ---\n", n);
    printf("Tiempo real (Tiempo total): %.10f segundos\n", wtime1 - wtime0);

    // Mostrar los tiempos en formato exponencial
    printf("Tiempo real (Tiempo total): %.10e segundos\n", wtime1 - wtime0);
    printf("\n");

    free(Arreglo);
    // Terminar programa normalmente
    exit(0);
}
```

Búsqueda exponencial

```
#include <stdio.h>
#include <stdlib.h>
#include "../tiempo.h"

int busquedaBinaria(int arr[], int, int, int);

int obtener_menor(int x, int y) {
    return (x <= y) ? x : y;
}
```

```c
int busquedaExponencial(int arr[], int n, int x) {
    int i = 0;
    if (arr[i] == x)
        return i;
    i = 1;
    while (i < n && arr[i] <= x) {
        i = i * 2;
    }

    int anterior = i / 2;
    int siguiente = obtener_menor(i, n - 1);
    int centro;

    while (anterior <= siguiente) {
        centro = anterior + (siguiente - anterior) / 2;
        if (arr[centro] == x)
            return centro;
        if (arr[centro] < x)
            anterior = centro + 1;
        else
            siguiente = centro - 1;
    }

    return -1;
}

// PROGRAMA PRINCIPAL
int main(int argc, char *argv[]) {
    // Variables para medición de tiempos
    double utime0, stime0, wtime0, utime1, stime1, wtime1;

    int i; // Variables para los ciclos
    long long int n; // Variable para el tamaño del arreglo (cambiado a long long int)
    int tam, num, numEncontrado; // Variables para el algoritmo

    // Recepción y decodificación de argumentos
    if (argc >= 2) {
        n = atoll(argv[1]); // Cambio en el tipo de la variable n y uso de atoll
        if (argc > 2) {
            num = atoi(argv[2]);
        } else {
            scanf("%d", &num);
        }
    } else {
        scanf("%lld", &n); // Cambio en la lectura de n a %lld
        scanf("%d", &num);
    }

    // Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    uswtime(&utime0, &stime0, &wtime0);

    int *Arreglo = malloc(n * sizeof(int));
    for (i = 0; i < n; i++) {
        scanf("%d", &Arreglo[i]);
```

```
    }

    tam = n; // Cambio en el cálculo del tamaño

    // Algoritmo
    numEncontrado = busquedaExponencial(Arreglo, tam, num);

    // Impresión de resultados
    if (numEncontrado == -1)
        printf("Elemento no encontrado");
    else
        printf("El elemento %d se encontro en la posicion: %d\n", num, numEncontrado);

    // Evaluar los tiempos de ejecución
    uswtime(&utime1, &stime1, &wtime1);

    // Cálculo del tiempo de ejecución del programa
    printf("-- Tiempos con %lld elementos ---\n", n);
    printf("Tiempo real (Tiempo total): %.10f segundos\n", wtime1 - wtime0);

    // Mostrar los tiempos en formato exponencial
    printf("Tiempo real (Tiempo total): %.10e segundos\n", wtime1 - wtime0);
    printf("\n");

    free(Arreglo);
    // Terminar programa normalmente
    exit(0);
}
```

Búsqueda de Fibonacci

```
#include <stdio.h>
#include <stdlib.h>
#include "../tiempo.h"

int min(int x, int y) {
    return (x <= y) ? x : y;
}

int busquedaFibonacci(int *arr, int x, int n) {
    int ultimo_fibo = 1;
    int penultimo_fibo = 0;
    int siguiente_fibo = penultimo_fibo + ultimo_fibo;

    while (siguiente_fibo < n) {
        penultimo_fibo = ultimo_fibo;
        ultimo_fibo = siguiente_fibo;
        siguiente_fibo = penultimo_fibo + ultimo_fibo;
    }
```

```
    int limite = 0;

    while (siguiente_fibo > 1) {
        int i = min(limite + penultimo_fibo, n) - 1;

        if (arr[i] < x) {
            siguiente_fibo = ultimo_fibo;
            ultimo_fibo = penultimo_fibo;
            penultimo_fibo = siguiente_fibo - ultimo_fibo;
            limite = i + 1;
        } else if (arr[i] > x) {
            siguiente_fibo = penultimo_fibo;
            ultimo_fibo = ultimo_fibo - penultimo_fibo;
            penultimo_fibo = siguiente_fibo - ultimo_fibo;
        } else {
            return i;
        }
    }

    if (ultimo_fibo && arr[limite] == x)
        return limite;

    return -1;
}

int main(int argc, char *argv[]) {
    // Variables para medición de tiempos
    double utime0, stime0, wtime0, utime1, stime1, wtime1;

    int i; // Variables para los ciclos
    long long int n; // Variable para el tamaño del arreglo (cambiado a long long int)
    int tam, num, numEncontrado; // Variables para el algoritmo

    // Recepción y decodificación de argumentos
    if (argc >= 2) {
        n = atoll(argv[1]); // Cambio en el tipo de la variable n y uso de atoll
        if (argc > 2) {
            num = atoi(argv[2]);
        } else {
            scanf("%d", &num);
        }
    } else {
        scanf("%lld", &n); // Cambio en la lectura de n a %lld
        scanf("%d", &num);
    }

    // Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    uswtime(&utime0, &stime0, &wtime0);

    int *Arreglo = malloc(n * sizeof(int));
    for (i = 0; i < n; i++) {
        scanf("%d", &Arreglo[i]);
    }
```

```
    tam = n; // Cambio en el cálculo del tamaño

    // Algoritmo
    numEncontrado = busquedaFibonacci(Arreglo, num, tam);

    // Impresión de resultados
    if (numEncontrado >= 0)
        printf("El elemento %d se encontro en la posicion: %d\n", num, numEncontrado);
    else
        printf("Elemento no encontrado");

    // Evaluar los tiempos de ejecución
    uswtime(&utime1, &stime1, &wtime1);

    // Cálculo del tiempo de ejecución del programa
    printf("-- Tiempos con %lld elementos ---\n", n);
    printf("Tiempo real (Tiempo total): %.10f segundos\n", wtime1 - wtime0);

    // Mostrar los tiempos en formato exponencial
    printf("Tiempo real (Tiempo total): %.10e segundos\n", wtime1 - wtime0);
    printf("\n");

    free(Arreglo);
    // Terminar programa normalmente
    exit(0);
}
```

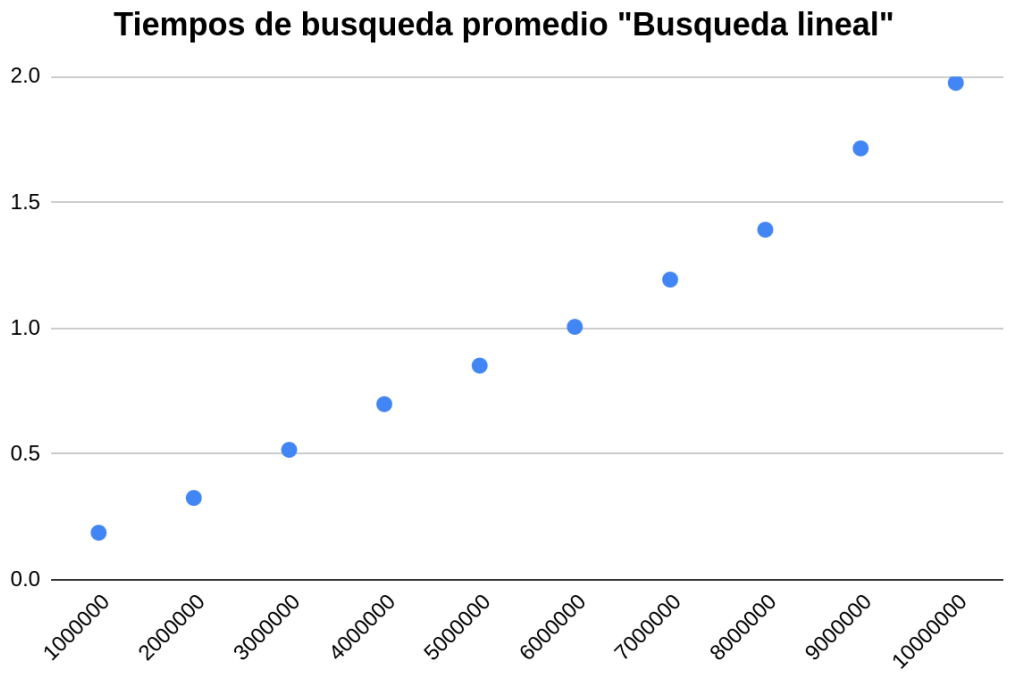
4.- Registro de los tiempos de búsqueda promedio de todos los algoritmos.

Algoritmo	Tamaño de N	Tiempo promedio de todas las búsquedas
Búsqueda lineal o secuencial	1000000	0.1878911018
	2000000	0.3264364958
	3000000	0.5171173811
	4000000	0.6984700799
	5000000	0.8521822929
	6000000	1.005517328
	7000000	1.193224823
	8000000	1.392010105
	9000000	1.714698637
	10000000	1.974507928
Búsqueda en un arbol binario de busqueda	1000000	0.2075582266
	2000000	0.4304417253
	3000000	0.6336193323
	4000000	0.9931870937
	5000000	1.237547743
	6000000	1.418361235
	7000000	1.64032315
	8000000	1.80688585
	9000000	2.125991535
	10000000	2.496997654
Búsqueda binaria o diatónica	1000000	0.1840609312
	2000000	0.3688258052
	3000000	0.5296885967
	4000000	0.6760047793
	5000000	0.9811458945
	6000000	1.087353945
	7000000	1.261013937
	8000000	1.457639515
	9000000	1.699121702
	10000000	1.807818246

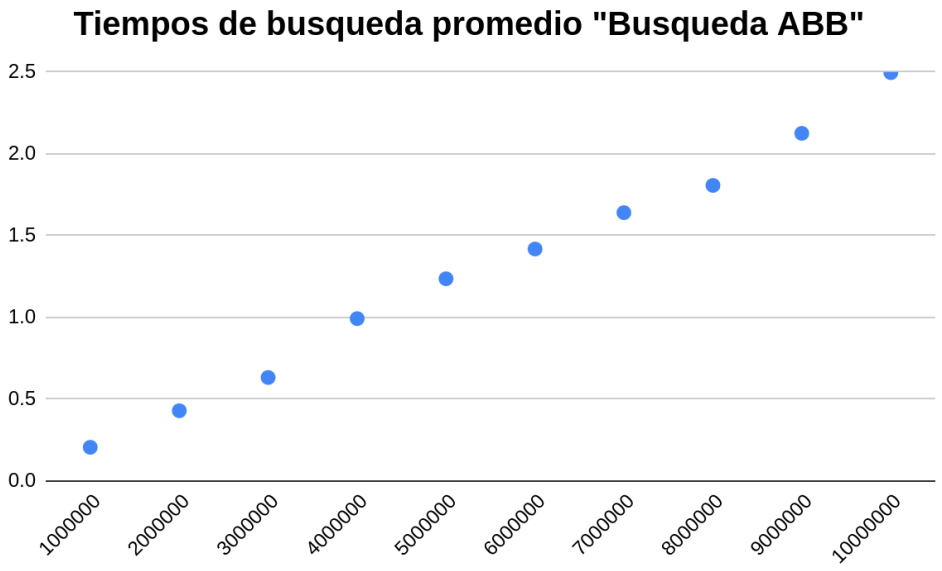
Búsqueda exponencial	1000000	0.1737502337
	2000000	0.3247193217
	3000000	0.4846215725
	4000000	0.7096844077
	5000000	0.8829488516
	6000000	1.063800991
	7000000	1.236817396
	8000000	1.398847616
	9000000	1.604394138
	10000000	1.742619991
Búsqueda de Fibonacci	1000000	0.000776434
	2000000	0.000767338
	3000000	0.000786662
	4000000	0.000755215
	5000000	0.000774324
	6000000	0.000772548
	7000000	0.000919342
	8000000	0.000804424
	9000000	0.000904667
	10000000	0.00098803

5.- Gráficas de comportamiento temporal de los algoritmos de búsqueda

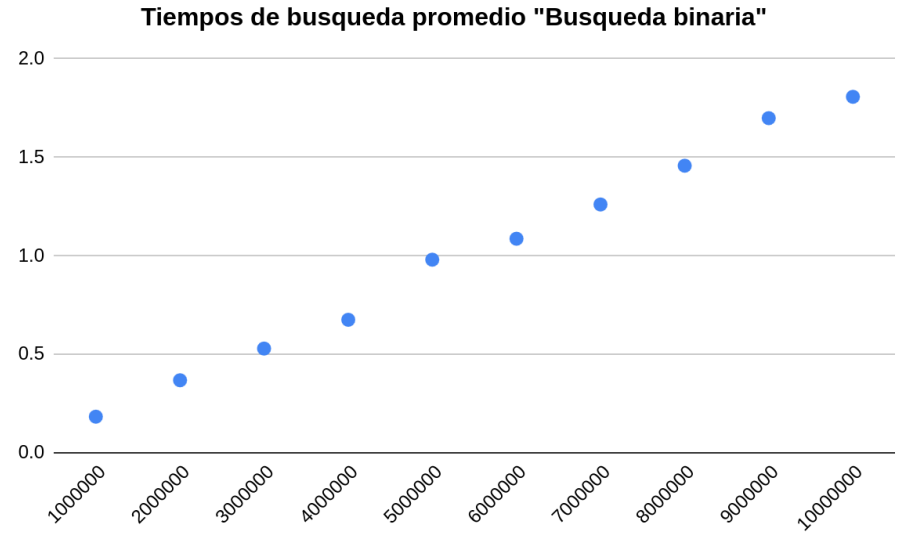
Comportamiento temporal de la búsqueda lineal



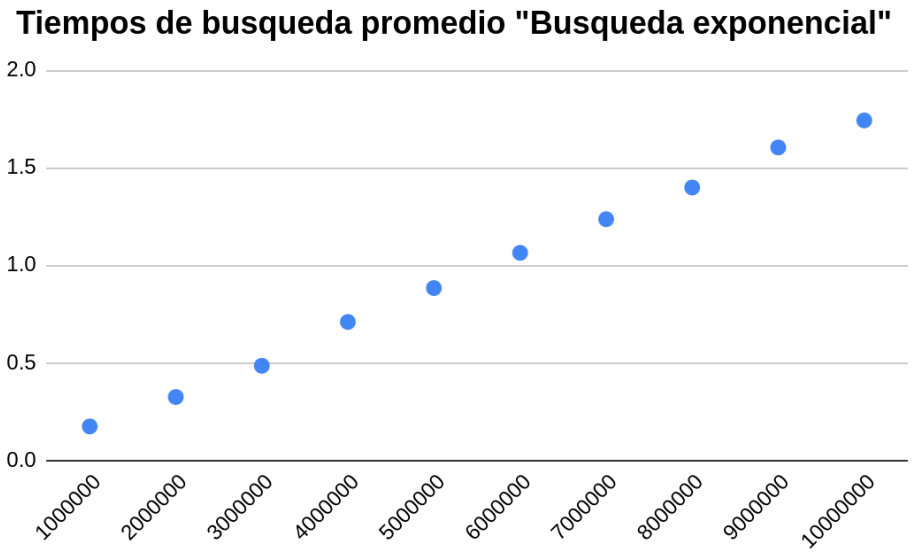
Comportamiento temporal de la búsqueda en un árbol binario de búsqueda



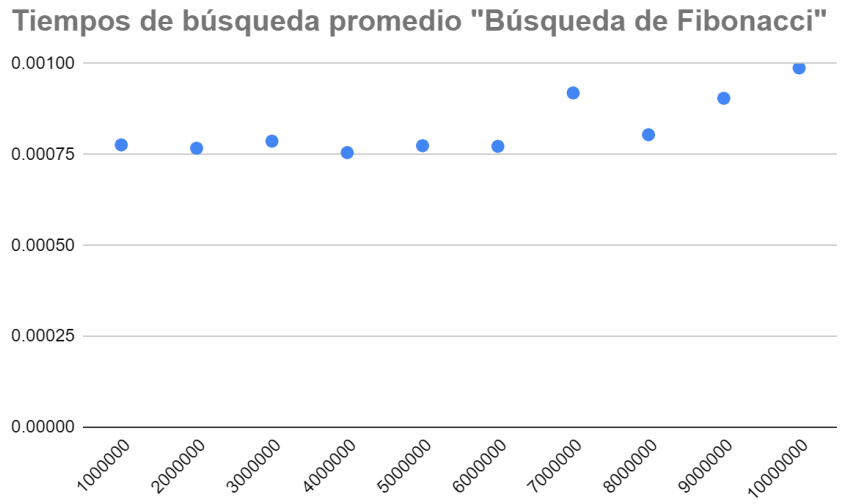
Comportamiento temporal de la búsqueda binaria



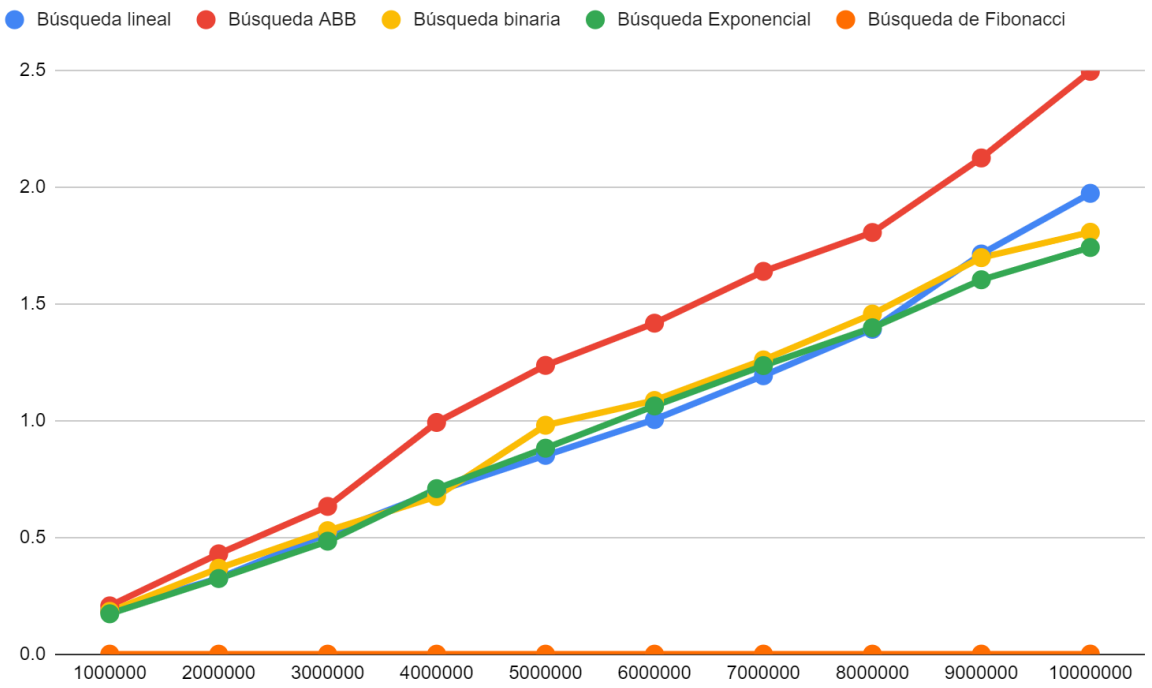
Comportamiento temporal de la búsqueda exponencial



Comportamiento temporal de la búsqueda de Fibonacci

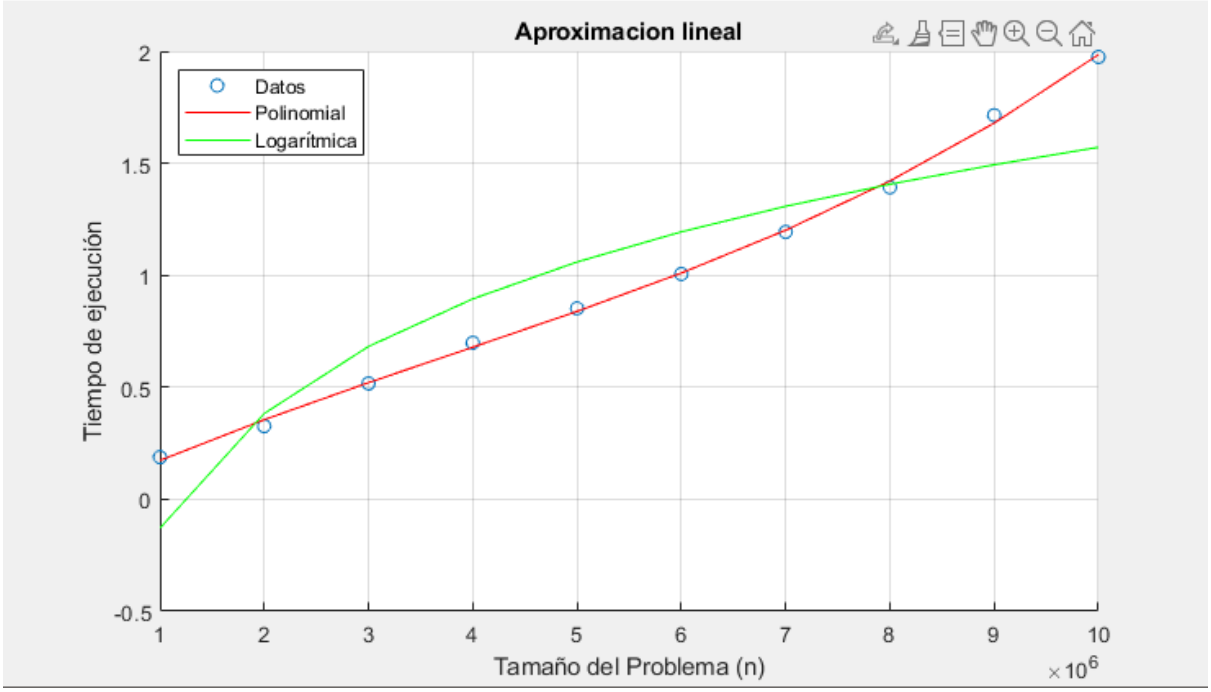


6.- Gráfica comparativa del comportamiento temporal de los 5 algoritmos de búsqueda

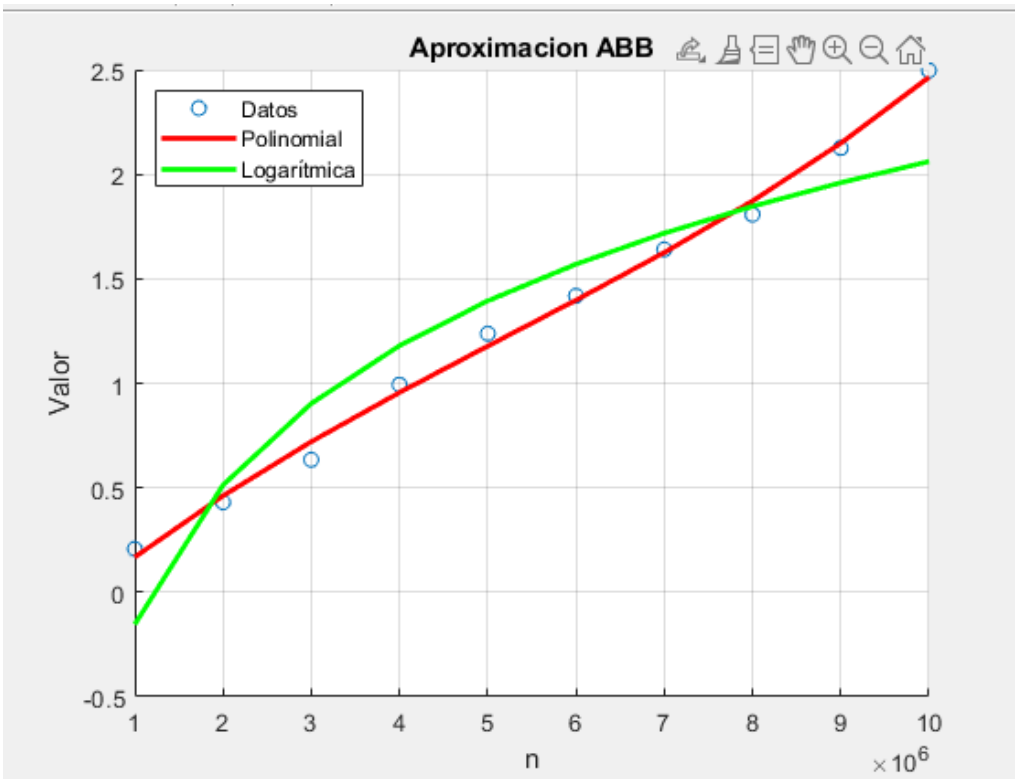


7.- Aproximación del comportamiento temporal en Matlab

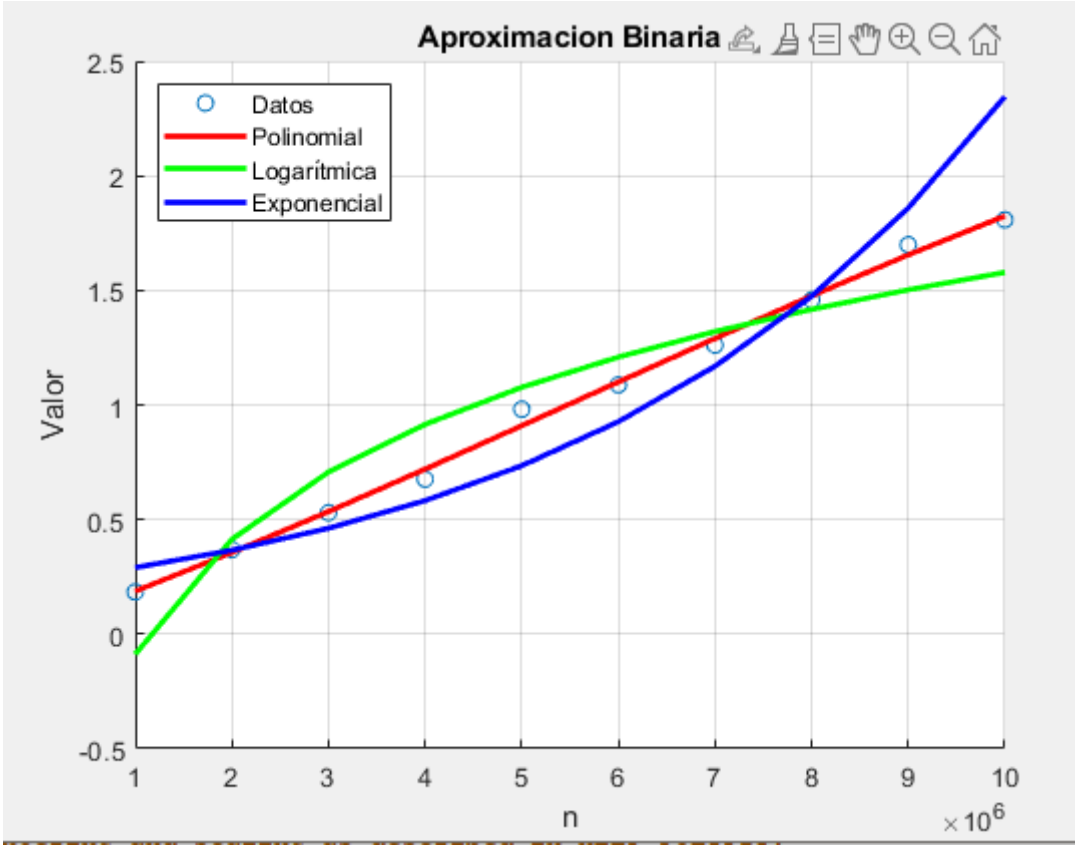
Aproximación del algoritmo de búsqueda lineal:



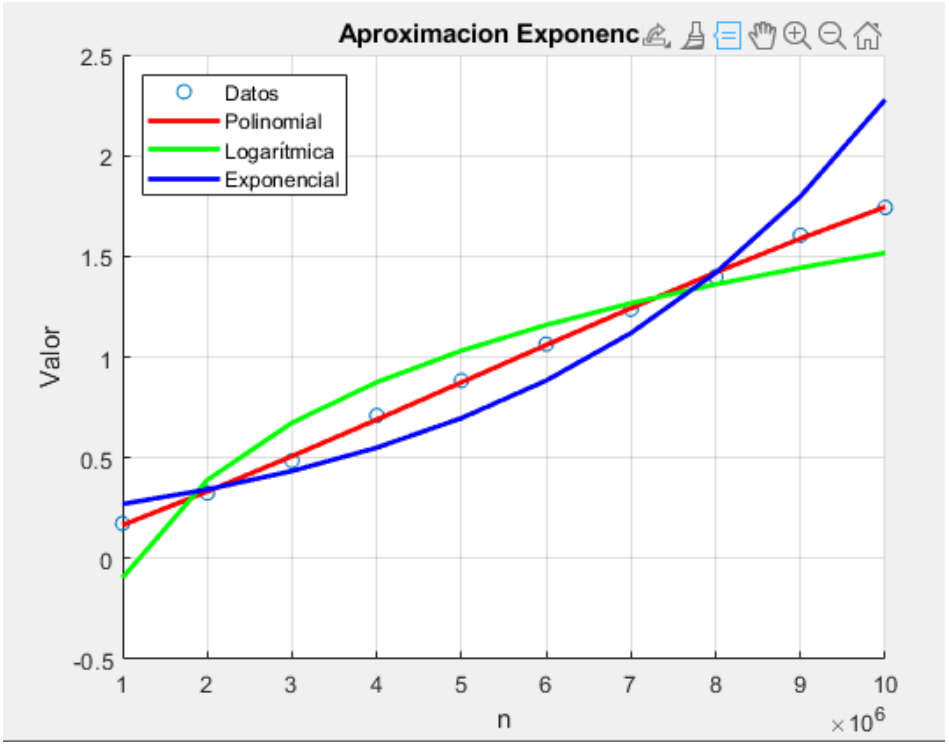
Aproximación del Algoritmo con ABB:



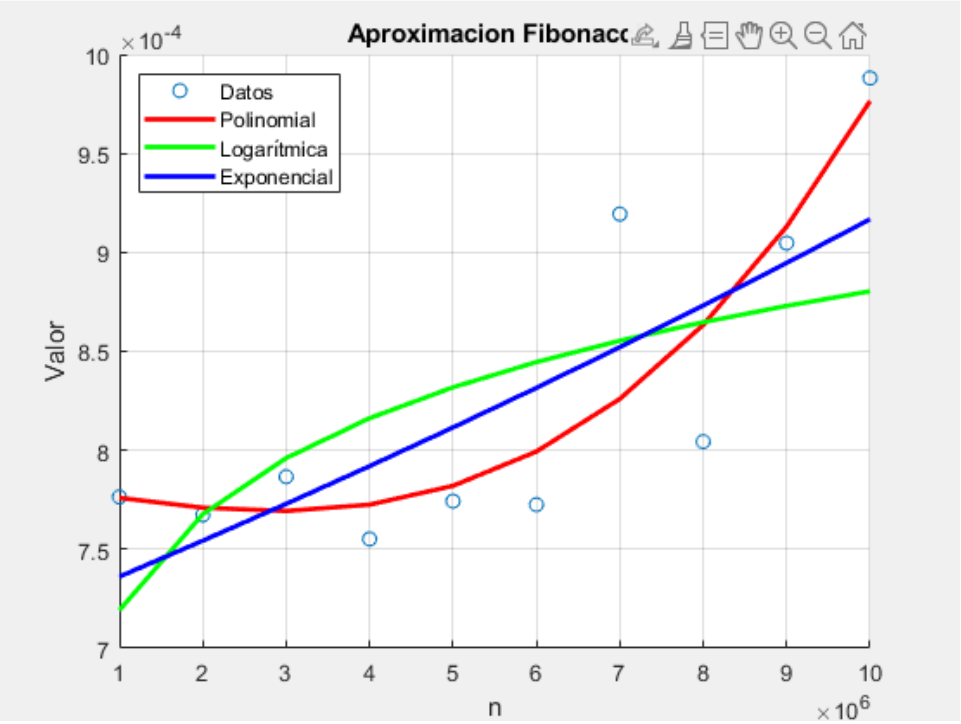
Aproximación del algoritmo por búsqueda binaria:



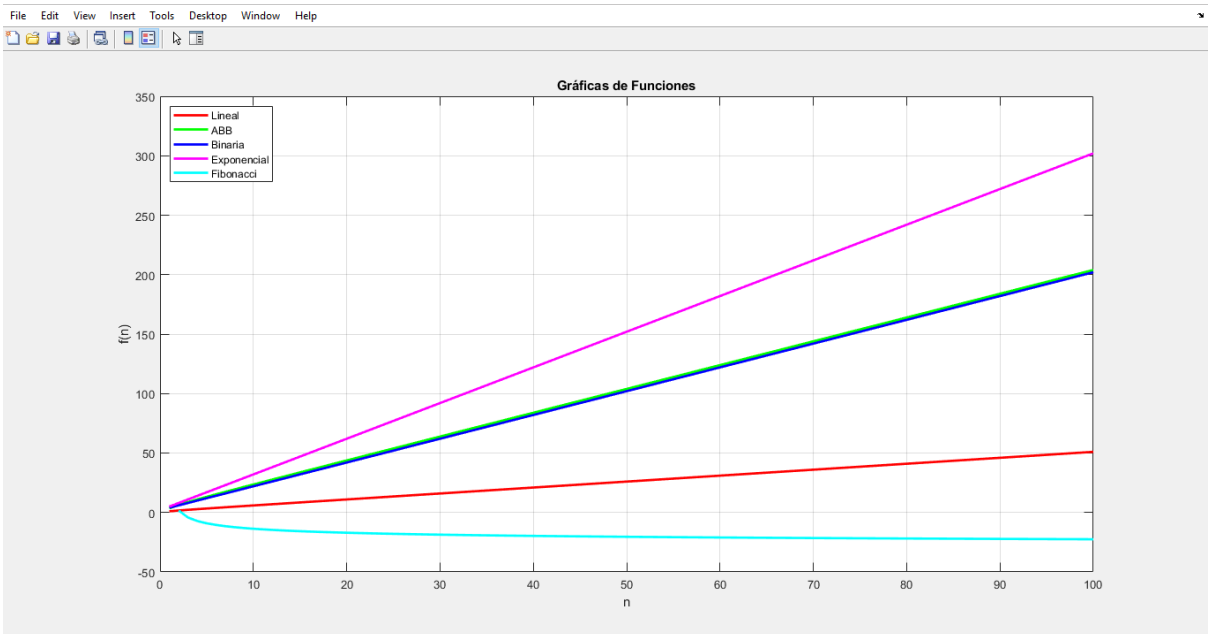
Aproximación del algoritmo por búsqueda exponencial:



Aproximación del algoritmo de búsqueda de Fibonacci:



8.- Gráfica comparativa de las aproximaciones del comportamiento temporal de los 5 algoritmos de búsqueda



12.- Cuestionario

1.- ¿Cuál de los 5 algoritmos es el más fácil de implementar?
El algoritmo por búsqueda lineal.

2.- ¿Cuál de los 5 algoritmos es el más difícil de implementar?
Algoritmo de búsqueda en un arbol binario.

3.- ¿Cuál de los 5 algoritmos fue el más difícil de implementar en su variante con hilos?

4.- ¿Cuál de los 5 algoritmos resultó ser más rápido en su variante con hilos? ¿Por qué?

5.- ¿Cuál de los 5 algoritmos no representó una ventaja en su variante con hilos? ¿Por qué?

6.- ¿Cuál algoritmo tiene menor complejidad temporal?
El algoritmo del árbol binario y el fibonacci.

7.- ¿Cuál algoritmo tiene mayor complejidad temporal?
El lineal.

8.- ¿El comportamiento experimental de los algoritmos era el esperado? ¿Por qué?
Si, porque los métodos teóricos nos arrojaban eso.

9.- ¿Sus resultados experimentales difieren mucho de los análisis teóricos que realizó? ¿A qué se debe?
No mucho, si graficamos ambas funciones de complejidad, serían muy parecidas.

10.- ¿En la versión con hilos, usar n hilos, dividió el tiempo en n? ¿Lo hizo n veces más rápido?

11.- ¿Cuál es el porcentaje de mejora que tiene cada uno de los algoritmos en su variante con hilos? ¿Es lo que se esperaba? ¿Por qué?

12.- ¿Existió un entorno controlado para realizar las pruebas experimentales? ¿Cuál fue?
Si, todas las pruebas fueron en la misma computadora, por lo que los resultados fueron lo más precisos que se pudieron realizar.

13.- ¿Si solo se realizará el análisis teórico de un algoritmo antes de implementarlo, podrías asegurar cuál es el mejor?
Si.

14.- ¿Qué tan difícil fue realizar el análisis teórico de cada algoritmo?
Muy difícil, el del árbol binario y el de fibonacci.

15.- ¿Qué recomendaciones darían a los nuevos equipos para realizar esta práctica?
Que tengan conocimientos sobre las series, ya que se necesitan para los cálculos, que entiendan bien las estructuras de datos en c.

Bibliografía

GeeksforGeeks | A computer science portal for geeks. (s.f.). GeeksforGeeks. https://www.geeksforgeeks.org/