

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Análisis de Algoritmos

Grupo: **3CM3**

Práctica 02:

“Análisis temporal (Algoritmos de búsqueda)”

Equipo:

Equipo de Moy

Alumnos:

Hernández López Moisés

Herrera Merino Roxana Angélica

Jiménez Delgado Luis Diego

Contenido

| | |
|---|----|
| Planteamiento del problema | 4 |
| Actividades y pruebas | 5 |
| Punto 1: Análisis teórico de casos y funciones de complejidad temporal | 5 |
| Búsqueda lineal | 5 |
| Búsqueda en un árbol AVL | 7 |
| Búsqueda binaria | 9 |
| Búsqueda exponencial | 11 |
| Búsqueda de Fibonacci | 14 |
| Punto 2: Implementación de los algoritmos de búsqueda en Lenguaje ANSI C | 17 |
| Punto 3: Adaptaciones sobre los códigos | 17 |
| Punto 4: Registro de los tiempos de búsqueda promedio de todos los algoritmos | 18 |
| Punto 5: Gráficas de comportamiento temporal de los algoritmos de búsqueda | 19 |
| Gráfica 3.1: Comportamiento temporal de la búsqueda lineal | 19 |
| Gráfica 3.2: Comportamiento temporal de la búsqueda en un árbol AVL | 19 |
| Gráfica 3.3: Comportamiento temporal de la búsqueda binaria | 19 |
| Gráfica 3.4: Comportamiento temporal de la búsqueda exponencial | 20 |
| Gráfica 3.5: Comportamiento temporal de la búsqueda de Fibonacci | 20 |
| Punto 6: Gráfica comparativa del comportamiento temporal de los 5 algoritmos de búsqueda | 21 |
| Punto 7: Aproximación del comportamiento temporal en Matlab | 22 |
| Curva 7.1: Aproximación del comportamiento temporal de la búsqueda lineal | 22 |
| | 22 |
| Curva 7.2: Aproximación del comportamiento temporal de la búsqueda en un árbol AVL | 22 |
| Curva 7.3: Aproximación del comportamiento temporal de la búsqueda binaria | 23 |
| Curva 7.4 Aproximación del comportamiento temporal de la búsqueda exponencial | 23 |
| Curva 7.5 Aproximación del comportamiento temporal de la búsqueda de Fibonacci | 24 |
| Punto 8: Gráfica comparativa de las aproximaciones del comportamiento temporal de los 5 algoritmos de búsqueda | 24 |
| Punto 9: Constante multiplicativa | 24 |
| Punto 10: Determinación de los tiempos de búsqueda para diferentes tamaños del problema con base en la función ajustada del peor caso | 25 |
| Punto 11: Mejoras de los algoritmos usando Threads | 26 |

| | |
|--|----|
| Tabla 11: Comparativa de los tiempos promedios de los algoritmos en sus versiones originales y en sus versiones con hilos..... | 26 |
| Punto 12: Cuestionario | 27 |
| Bibliografía..... | 27 |
| Anexos..... | 28 |

Planteamiento del problema

Actividades y pruebas

Punto 1: Análisis teórico de casos y funciones de complejidad temporal

Búsqueda lineal

Código

```
int lineal(int *arr, int x, int n)
{
    int i = 0;
    while (i < n)
    {
        if (arr[i] == x)
            return i;
        i++;
    }
    return -1;
}
```

Operaciones básicas

La operación básica tomada en cuenta fue la comparación:

```
if (arr[i] == x)
```

Mejor caso

Se presenta cuando el número X se encuentra en la posición 0 del arreglo.

$$f_t(n) = 1$$

Peor caso

Se presenta cuando el número X se encuentra en la última posición del arreglo, o no se encuentra dentro de él.

$$f_t(n) = n$$

Caso medio

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

$$I(n) = \begin{cases} X \text{ se encuentra en la 1}^\circ \text{ posición,} \\ X \text{ se encuentra en la 2}^\circ \text{ posición,} \\ X \text{ se encuentra en la 3}^\circ \text{ posición,} \\ \dots, \\ X \text{ se encuentra en la } n^\circ \text{ posición,} \\ X \text{ no se encuentra en el arreglo} \end{cases}$$

$$O(n) = \begin{cases} 1, \\ 2, \\ 3, \\ \dots, \\ n, \\ n \end{cases}$$

Teniendo en cuenta que las instancias son equiprobables de suceder:

$$P(n) = \left\{ \begin{array}{c} \frac{1}{n+1}, \\ \frac{1}{n+1}, \\ \frac{1}{n+1}, \\ \dots, \\ \frac{1}{n+1}, \\ \frac{1}{n+1} \end{array} \right.$$

Desarrollando:

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

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

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

Dividiendo el polinomio obtenemos finalmente:

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

Búsqueda en un árbol AVL

Se supone para este ejercicio un árbol binario balanceado de búsqueda como tamaño del problema.

Código

```
int estaAVL(AVL a, TipoAVL elem)
{
    while(a != NULL && a -> info != elem)
        a = (elem < (a -> info)) ? a -> izq : a -> der;
    return a != NULL;
}
```

Operaciones básicas

Las operaciones básicas consideradas son las siguientes, en las que se cuentan solo comparaciones y asignaciones.

1. while(a != NULL && a -> info != elem)
2. a = (elem < (a -> info)) ? a -> izq : a -> der;

Mejor caso

Se presenta cuando el número X se encuentra en la raíz del árbol.

$$f_t(n) = 3$$

Peor caso

El elemento X no se encuentra dentro del árbol binario de búsqueda.

$$f_t(n) = 5 \log_2 n$$

Caso medio

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

$$I(n) = \begin{cases} X \text{ se encuentra en el } 1^\circ \text{ nivel del árbol,} \\ X \text{ se encuentra en el } 2^\circ \text{ nivel del árbol,} \\ X \text{ se encuentra en el } 3^\circ \text{ nivel del árbol,} \\ X \text{ se encuentra en el } 4^\circ \text{ nivel del árbol,} \\ \dots, \\ X \text{ se encuentra en el } \log_2 n^\circ \text{ nivel del árbol,} \\ X \text{ se encuentra en el } (\log_2 n + 1)^\circ \text{ nivel del árbol,} \\ X \text{ no se encuentra en el árbol} \end{cases}$$

$$O(n) = \begin{cases} 3 + 5(0), \\ 3 + 5(1), \\ 3 + 5(2), \\ 3 + 5(3), \\ \dots, \\ 3 + 5(\log_2 n), \\ 5(\log_2 n + 1) \end{cases}$$

$$P(n) = \begin{cases} \frac{1}{n+1}, \\ \frac{\frac{1}{2}}{n+1}, \\ \frac{\frac{1}{4}}{n+1}, \\ \dots, \\ \frac{n/2}{n+1}, \\ \frac{1}{n+1} \end{cases}$$

Desarrollando:

$$f_t(n) = \left(\frac{1}{n+1}\right) \{(1)[3 + 5(0)] + (2)[3 + 5(1)] + (4)[3 + 5(2)] + \dots + \left(\frac{n}{2}\right)[3 + 5(\log_2 n)] + (1)[5 \log_2 n]\}$$

$$f_t(n) = \left(\frac{1}{n+1}\right) [(1)(3) + (1)(5)(0) + (2)(3) + (2)(5)(1) + (4)(3) + (4)(5)(2) + \dots + \left(\frac{n}{2}\right)(3) + \left(\frac{n}{2}\right)(5)(\log_2 n) + 5 \log_2 n]$$

$$f_t(n) = \left(\frac{1}{n+1}\right) \{(3) \left(1 + 2 + 4 + \dots + \frac{n}{2}\right) + (5)[(1)(0) + (2)(1) + (4)(2) + (8)(3) + \dots + \frac{n}{2} \log_2 n] + 5 \log_2 n\}$$

Suponiendo un árbol completo en donde su último nivel cuenta con $\frac{n}{2}$ nodos, cantidad que sería potencia de 2

El total de nodos en el árbol $(1 + 2 + 4 \dots)$ sería equivalente al doble de esa cantidad, menos 1, por lo tanto

$$f_t(n) = \left(\frac{1}{n+1}\right) \{(3)(n-1) + (5) \left[(1)(0) + (2)(1) + (4)(2) + (8)(3) + \dots + \frac{n}{2} \log_2 n\right] + 5 \log_2 n\}$$

Expresando la suma expandida, en sigma:

$$f_t(n) = \left(\frac{1}{n+1}\right) \left\{ (3)(n-1) + 5 \sum_{i=1}^{\log_2 n} i 2^i + 5 \log_2 n \right\}$$

Teniendo en cuenta el resultado de la serie de potencias de suma finita de la forma:

$$\sum_{i=1}^n i x^i = x \frac{1 - x^n}{(1 - x)^2} - \frac{n x^{n+1}}{1 - x}$$

Podemos expresar $f_t(n)$ de la siguiente manera:

$$f_t(n) = \left(\frac{1}{n+1}\right) \left\{ (3)(n-1) + 2 \frac{1 - 2^{\log_2 n}}{(1 - 2)^2} - \frac{\log_2 n (2^{\log_2 n + 1})}{1 - 2} + 5 \log_2 n \right\}$$

$$f_t(n) = \left(\frac{1}{n+1}\right) \left\{ (3)(n-1) + 2 \frac{1 - n}{1} - \frac{\log_2 n (2n)}{-1} + 5 \log_2 n \right\}$$

$$f_t(n) = \left(\frac{1}{n+1}\right) \{(3)(n-1) + 2(1-n) + 2n \log_2 n + 5 \log_2 n\}$$

Simplificando obtenemos finalmente:

$$f_t(n) = \frac{2n \log_2 n + n + 5 \log_2 n - 1}{n + 1}$$

Búsqueda binaria

Código

```
int binaria(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;
}
```

Operaciones básicas

Las operaciones básicas consideradas son las siguientes, en las que se cuentan solo comparaciones y asignaciones.

1. `centro = anterior + (siguiente - anterior) / 2;`
2. `if (arr[centro] == x)`
3. `if (arr[centro] < x)`
4. `anterior = centro + 1;`
5. `siguiente = centro - 1;`

Mejor caso

Se presenta cuando el número X se encuentra a la mitad del rango inicial.

$$f_t(n) = 2$$

Peor caso

Se presenta cuando el número X no se encuentra en el rango a buscar.

$$f_t(n) = 4 \log_2 n$$

Caso medio

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

$$I(n) = \begin{cases} X \text{ se encuentra en el 1º paso de la binaria,} \\ X \text{ se encuentra en el 2º paso de la binaria,} \\ X \text{ se encuentra en el 3º paso de la binaria,} \\ X \text{ se encuentra en el 4º paso de la binaria,} \\ \dots, \\ X \text{ se encuentra en el } \log_2 n^\circ \text{ paso de la binaria,} \\ X \text{ no se encuentra en el rango a buscar} \end{cases}$$

$$O(n) = \begin{cases} 2, \\ 2 + 4(1), \\ 2 + 4(2), \\ 2 + 4(3), \\ 2 + 4(4), \\ \dots, \\ 2 + 4(\log_2 n - 1), \\ 4 \log_2 n \end{cases}$$

Teniendo en cuenta que las instancias son equiprobables de suceder:

$$P(n) = \begin{cases} \frac{1}{\log_2 n + 1}, \\ \frac{1}{\log_2 n + 1}, \\ \frac{1}{\log_2 n + 1}, \\ \dots, \\ \frac{1}{\log_2 n + 1}, \\ \frac{1}{\log_2 n + 1} \end{cases}$$

Desarrollando:

Sea $m = \log_2 n$

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

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

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

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

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

$$f_t(n) = \frac{6m + 2m^2 - 2m}{m+1} = \frac{2m^2 + 4m}{m+1}; \text{ dividiendo el polinomio obtenemos:}$$

$$f_t(n) = 2m + 2 + \frac{2}{m+1}; \text{ al ser el residuo pequeño consideramos desprecialo por lo tanto:}$$

$f_t(n) = 2m + 2$; sustituyendo m a su valor original obtenemos finalmente:

$$f_t(n) = 2 \log_2 n + 2$$

Búsqueda exponencial

Código

```
int exponencial(int *arr, int x, int n)
{
    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;
}
```

Operaciones básicas

Las operaciones básicas consideradas son las siguientes, en las que se cuentan solo comparaciones y asignaciones.

1. if (arr[i] == x)
2. while (i < n && arr[i] <= x)
3. i = i * 2;
4. centro = anterior + (siguiente - anterior) / 2;
5. if (arr[centro] == x)
6. if (arr[centro] < x)
7. anterior = centro + 1;
8. siguiente = centro - 1;

Mejor caso

Se presenta cuando el número X se encuentra en la posición 0 del arreglo.

$$f_t(n) = 1$$

Peor caso

De existir el elemento (que no es el caso), éste es menor que el arreglo en la posición n-1 y mayor que el arreglo en la posición (n-1)/2 (el elemento forma parte del último segmento de la búsqueda exponencial) y la búsqueda binaria no lo encontrará.

$$f_t(n) = (1) + (4 \log_2 n + 3) + (4 \log_2 \frac{n}{2})$$

$$f_t(n) = (1) + (4 \log_2 n + 3) + (4 \log_2 n - 4 \log_2 2) = (1) + (4 \log_2 n + 3) + (4 \log_2 n - 4)$$

$$f_t(n) = 8 \log_2 n$$

Caso medio

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

$$I(n) = \begin{cases} X \text{ se encuentra al inicio,} \\ \text{El while se rompe con } i = 1, \\ \text{El while se rompe con } i = 2, \\ \text{El while se rompe con } i = 4, \\ \text{El while se rompe con } i = 8, \\ \dots, \\ \text{El while se rompe con } i = n \end{cases}$$

Considerando la distancia entre cada par i 's de las iteraciones, así como tomando en cuenta el caso medio de la binaria para el cálculo de las operaciones de las instancias $f_t(n)$

$$= 2 \log_2 n + 2$$

$$O(n) = \begin{cases} 1, \\ 3 + (4)(0) + 2 \log_2 1 + 2, \\ 3 + (4)(1) + 2 \log_2 1 + 2, \\ 3 + (4)(2) + 2 \log_2 2 + 2, \\ 3 + (4)(3) + 2 \log_2 4 + 2, \\ 3 + (4)(4) + 2 \log_2 8 + 2 \\ \dots, \\ 3 + (4)(\log_2 n) + 2 \log_2 \frac{n}{2} + 2 \end{cases}$$

Teniendo en cuenta que las instancias son equiprobables de suceder:

$$P(n) = \begin{cases} \frac{1}{\log_2 n + 2}, \\ \frac{1}{\log_2 n + 2}, \\ \frac{1}{\log_2 n + 2}, \\ \dots, \\ \frac{1}{\log_2 n + 2}, \\ \frac{1}{\log_2 n + 2} \end{cases}$$

Desarrollando:

$$f_t(n) = (\frac{1}{\log_2 n + 2})(6 + 5 \log_2 n + 4 \frac{\log_2 n (\log_2 n + 1)}{2} + 2(0 + 1 + 2 + 3 + \dots + \log_2 n - 1))$$

$$f_t(n) = (\frac{1}{\log_2 n + 2})(6 - 2 + 5 \log_2 n + 2 \log_2 n (\log_2 n + 1) + \log_2 n (\log_2 n + 1))$$

$$f_t(n) = \left(\frac{1}{\log_2 n + 2}\right)(4 + 5 \log_2 n + 3 \log_2 n (\log_2 n + 1))$$

$$f_t(n) = \left(\frac{1}{\log_2 n + 2}\right)(4 + 5 \log_2 n + 3(\log_2 n)^2 + 3 \log_2 n)$$

$$f_t(n) = \left(\frac{1}{\log_2 n + 2}\right)(3(\log_2 n)^2 + 8 \log_2 n + 4); \text{ sea } m = \log_2 n$$

$$f_t(n) = \frac{3m^2 + 8m + 4}{m + 2}; \text{ dividiendo el polinomio obtenemos:}$$

$$f_t(n) = 3m + 2; \text{ sustituyendo } m \text{ a su original obtenemos finalmente:}$$

$$f_t(n) = 3 \log_2 n + 2$$

Búsqueda de Fibonacci

Código

```
int fibonacci(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 = obtener_menor(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;
}
```

Operaciones básicas

ENLISTAR LAS COMPARACIONES BÁSICAS TOMADAS EN CUENTA

Nota

Para el análisis de complejidad del ejercicio, se hace uso de la fórmula de Binet, la cual da como resultado una buena aproximación para la cantidad de números Fibonacci que son menores o iguales a n , con valores de n grandes. A continuación, se describe la fórmula de Binet.

$$F(n) = \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor; \text{ donde } \phi = \frac{1}{2}(1 + \sqrt{5})$$

Mejor caso

En la ejecución de un mejor escenario, el primer while se ejecutaría tantas veces como números Fibonacci existan menores o iguales a n . Añadido a esto, el segundo while se ejecutaría solamente una vez, por su camino más corto, por lo tanto:

$$f_t(n) = 4 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor + 4$$

Peor caso

El primer while se ejecutaría tantas veces como números Fibonacci existan menores o iguales a n . Añadido a esto, el segundo while se ejecutaría tantas veces como números Fibonacci existan menores o iguales a n y pasa por el peor camino.

$$f_t(n) = 4 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor + 6 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor$$

$$f_t(n) = 10 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor$$

Caso medio

Para el análisis del caso medio, se tomaron en cuenta 3 caminos posibles, equiprobables dentro del segundo while.

Sus complejidades individuales de cada uno se muestran a continuación:

$$f_{t_c1}(n) = 4 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor + 6 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor = 10 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor$$

$$f_{t_c2}(n) = 4 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor + 6 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor = 10 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor$$

$$f_{t_c3}(n) = 4 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor + 4 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor = 8 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor$$

$$f_t(n) = O_{t_c1}P_{t_c1} + O_{t_c2}P_{t_c2} + O_{t_c3}P_{t_c3}$$

$$f_t(n) = 10 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor \left(\frac{1}{3}\right) + 10 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor \left(\frac{1}{3}\right) + 8 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor \left(\frac{1}{3}\right)$$

$$f_t(n) = \left(\frac{1}{3}\right) \left(10 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor + 10 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor + 8 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \phi} \right\rfloor\right)$$

Finalmente, obtenemos:

$$f_t(n) = \left(\frac{1}{3}\right) (28 \left\lfloor \frac{\log n + \frac{1}{2} \log 5}{\log \varnothing} \right\rfloor)$$

Punto 2: Implementación de los algoritmos de búsqueda en Lenguaje ANSI C
Revisar los anexos.

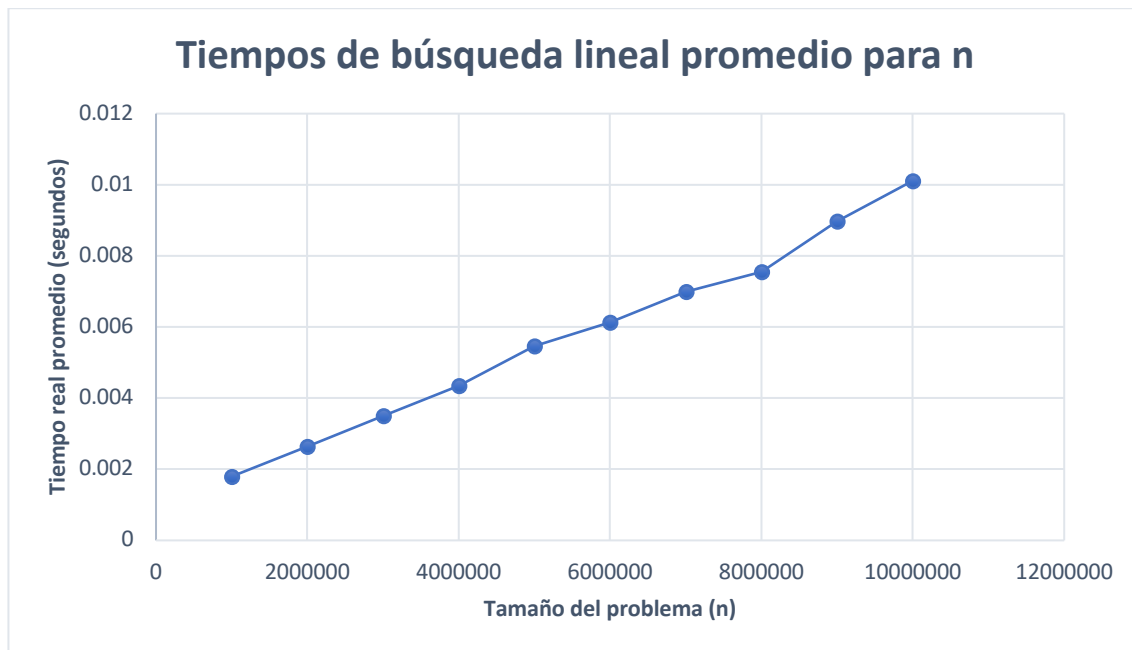
Punto 3: Adaptaciones sobre los códigos
Revisar los anexos.

Punto 4: Registro de los tiempos de búsqueda promedio de todos los algoritmos

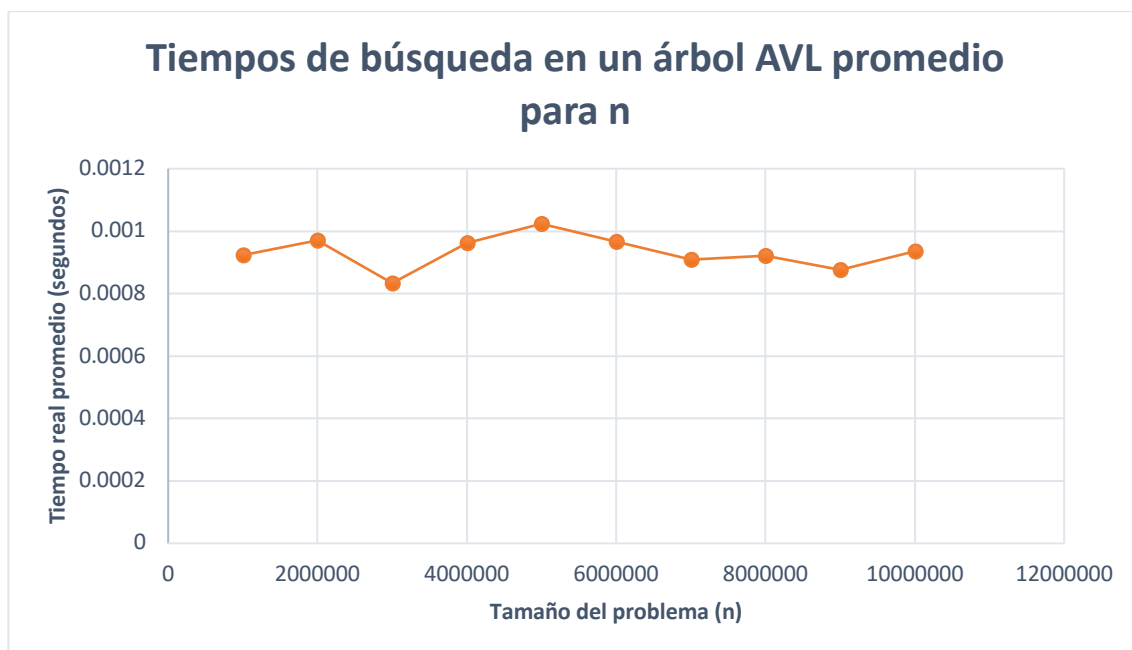
| Algoritmo | Tamaño de n | Tiempo real promedio (segundos) |
|-------------|-------------|---------------------------------|
| Lineal | 1,000,000 | 0.001797152 |
| | 2,000,000 | 0.002637243 |
| | 3,000,000 | 0.003495455 |
| | 4,000,000 | 0.004341638 |
| | 5,000,000 | 0.005462074 |
| | 6,000,000 | 0.006128943 |
| | 7,000,000 | 0.006999063 |
| | 8,000,000 | 0.007558894 |
| | 9,000,000 | 0.008976912 |
| | 10,000,000 | 0.010112357 |
| Árbol AVL | 1,000,000 | 0.000922227 |
| | 2,000,000 | 0.000970972 |
| | 3,000,000 | 0.000833464 |
| | 4,000,000 | 0.000962472 |
| | 5,000,000 | 0.001023459 |
| | 6,000,000 | 0.000965285 |
| | 7,000,000 | 0.000909293 |
| | 8,000,000 | 0.000920165 |
| | 9,000,000 | 0.000876212 |
| | 10,000,000 | 0.000935459 |
| Binaria | 1,000,000 | 0.000731754 |
| | 2,000,000 | 0.000771832 |
| | 3,000,000 | 0.000667727 |
| | 4,000,000 | 0.000851357 |
| | 5,000,000 | 0.000830579 |
| | 6,000,000 | 0.000823128 |
| | 7,000,000 | 0.000800455 |
| | 8,000,000 | 0.000808811 |
| | 9,000,000 | 0.000871813 |
| | 10,000,000 | 0.000740504 |
| Exponencial | 1,000,000 | 0.000875664 |
| | 2,000,000 | 0.000937891 |
| | 3,000,000 | 0.000947976 |
| | 4,000,000 | 0.000818825 |
| | 5,000,000 | 0.000949478 |
| | 6,000,000 | 0.000965202 |
| | 7,000,000 | 0.000994802 |
| | 8,000,000 | 0.000925291 |
| | 9,000,000 | 0.000894165 |
| | 10,000,000 | 0.000870275 |
| | 1,000,000 | 0.000776434 |
| | 2,000,000 | 0.000767338 |
| | 3,000,000 | 0.000786662 |
| | 4,000,000 | 0.000755215 |
| | 5,000,000 | 0.000774324 |

Punto 5: Gráficas de comportamiento temporal de los algoritmos de búsqueda

Gráfica 3.1: Comportamiento temporal de la búsqueda lineal



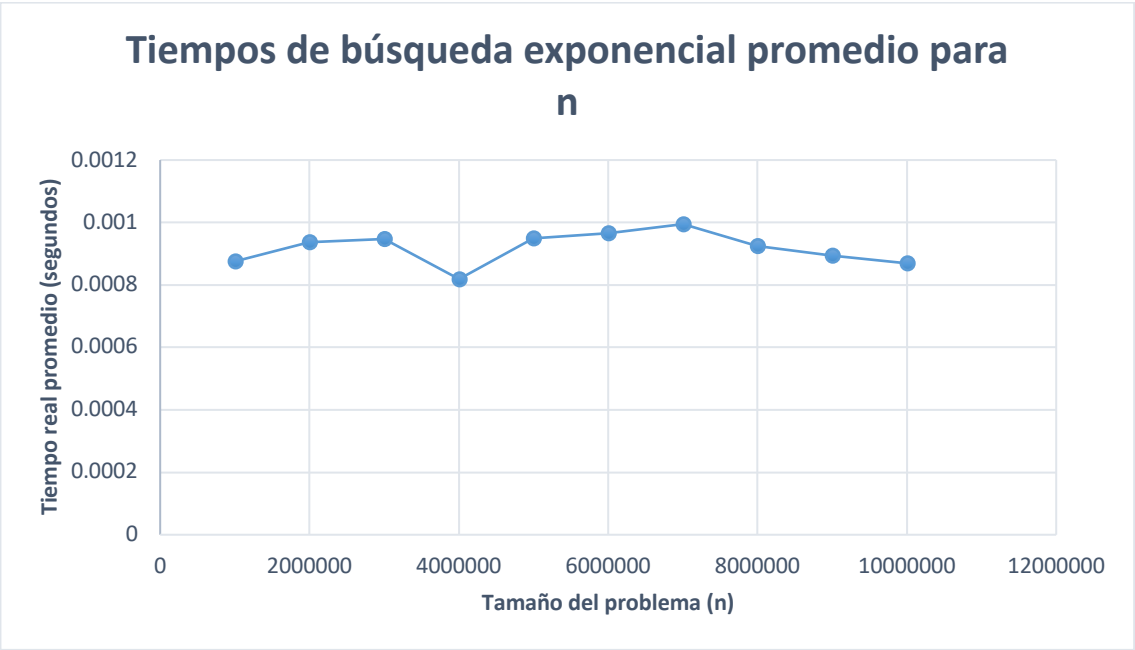
Gráfica 3.2: Comportamiento temporal de la búsqueda en un árbol AVL



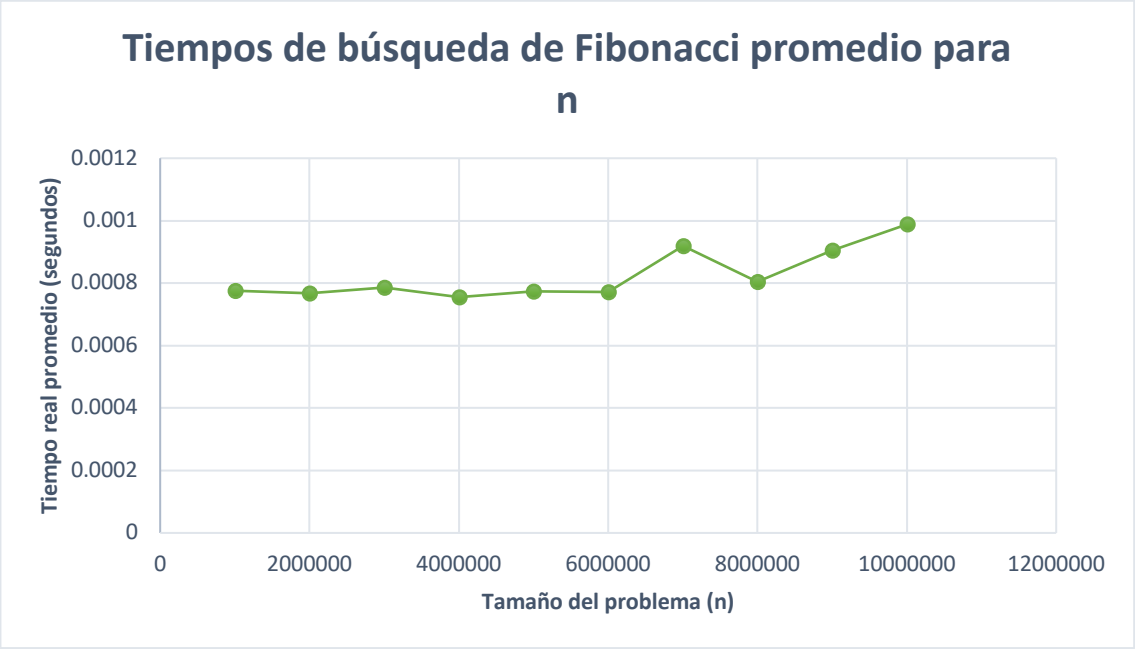
Gráfica 3.3: Comportamiento temporal de la búsqueda binaria



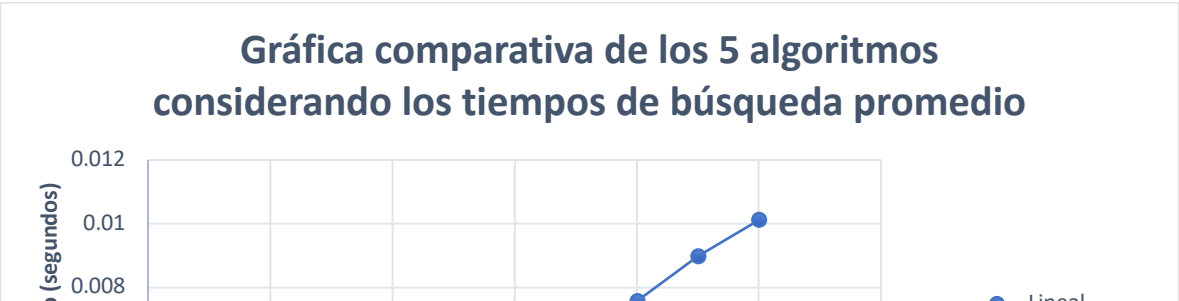
Gráfica 3.4: Comportamiento temporal de la búsqueda exponencial



Gráfica 3.5: Comportamiento temporal de la búsqueda de Fibonacci

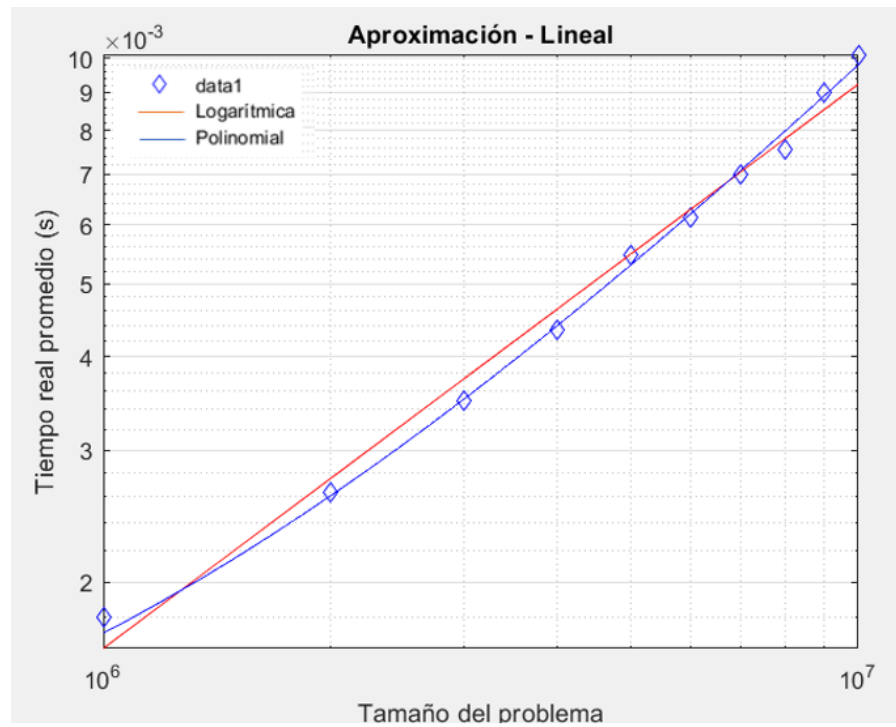


Punto 6: Gráfica comparativa del comportamiento temporal de los 5 algoritmos de búsqueda

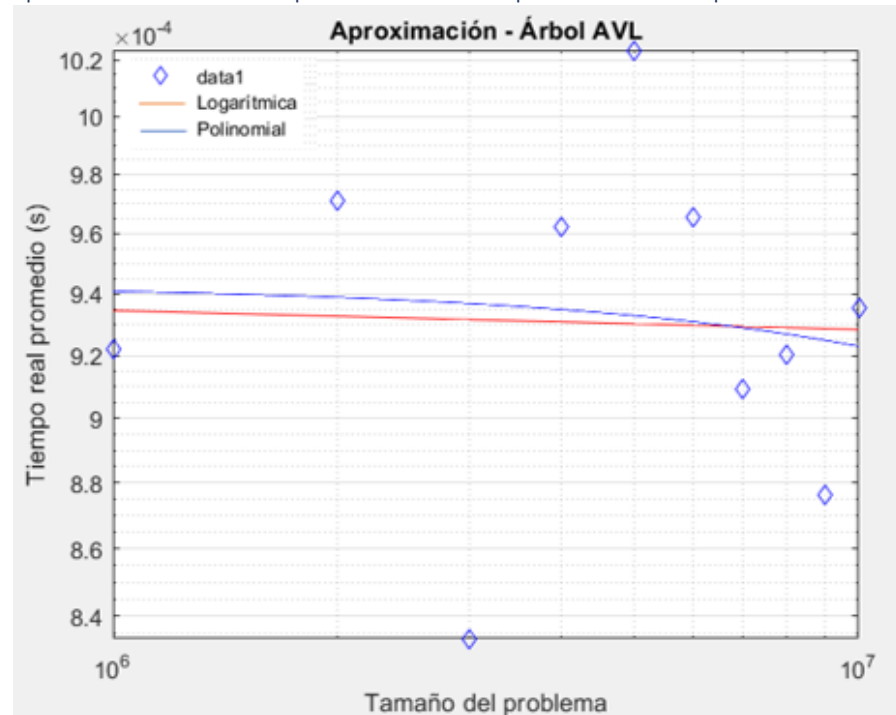


Punto 7: Aproximación del comportamiento temporal en Matlab

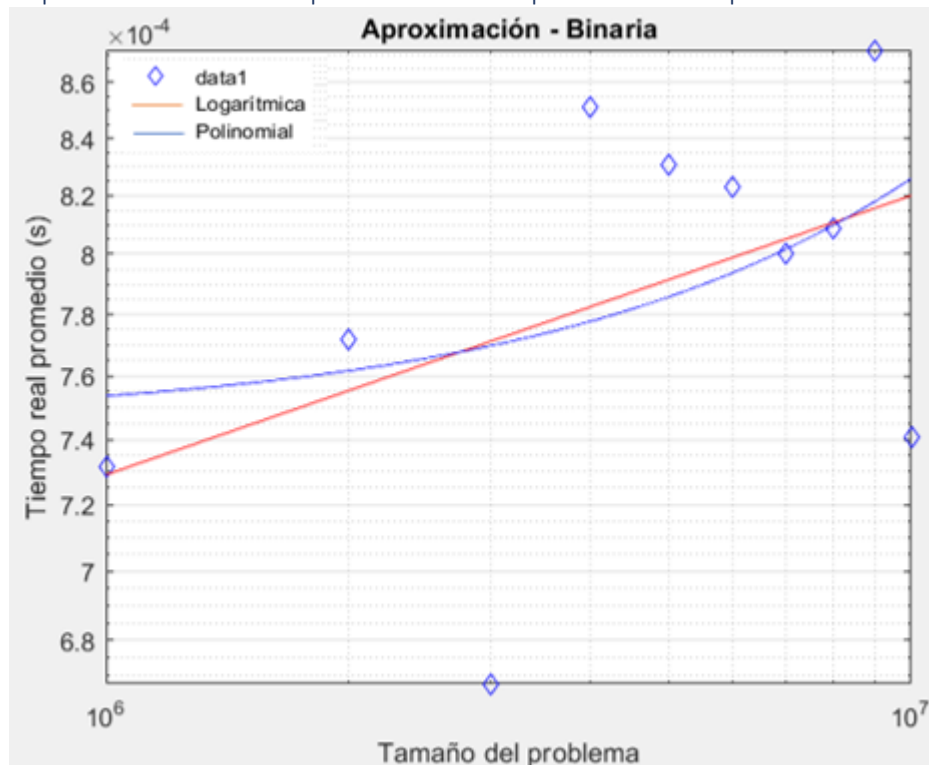
Curva 7.1: Aproximación del comportamiento temporal de la búsqueda lineal



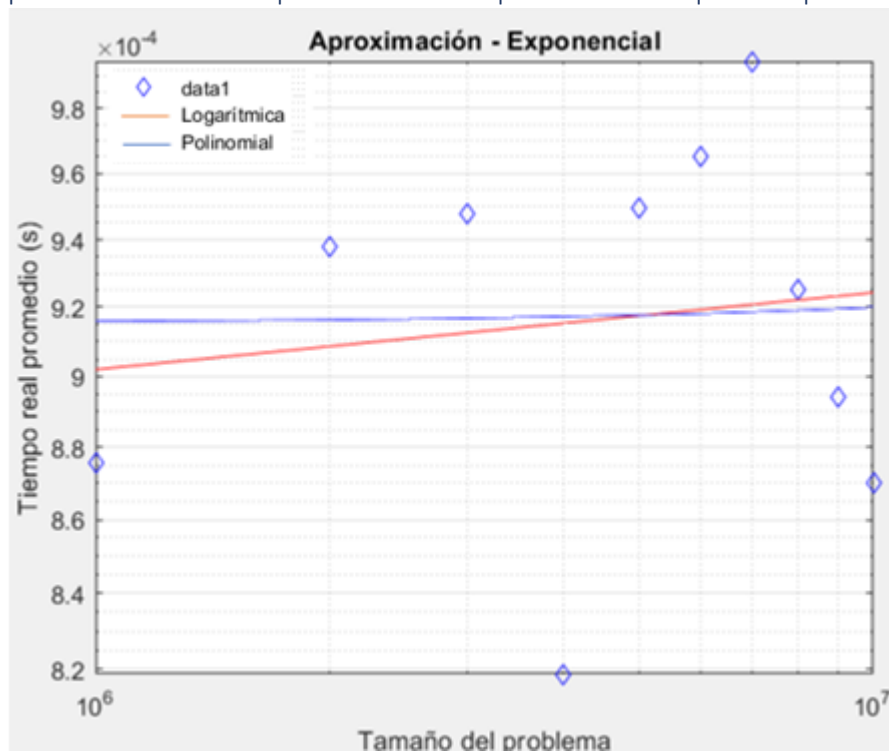
Curva 7.2: Aproximación del comportamiento temporal de la búsqueda en un árbol AVL



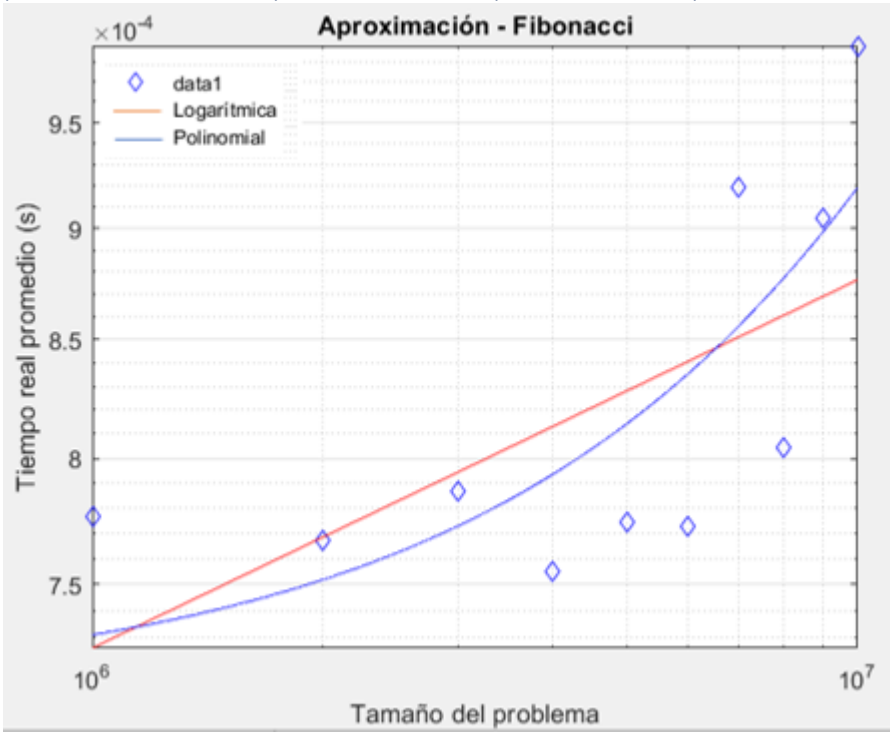
Curva 7.3: Aproximación del comportamiento temporal de la búsqueda binaria



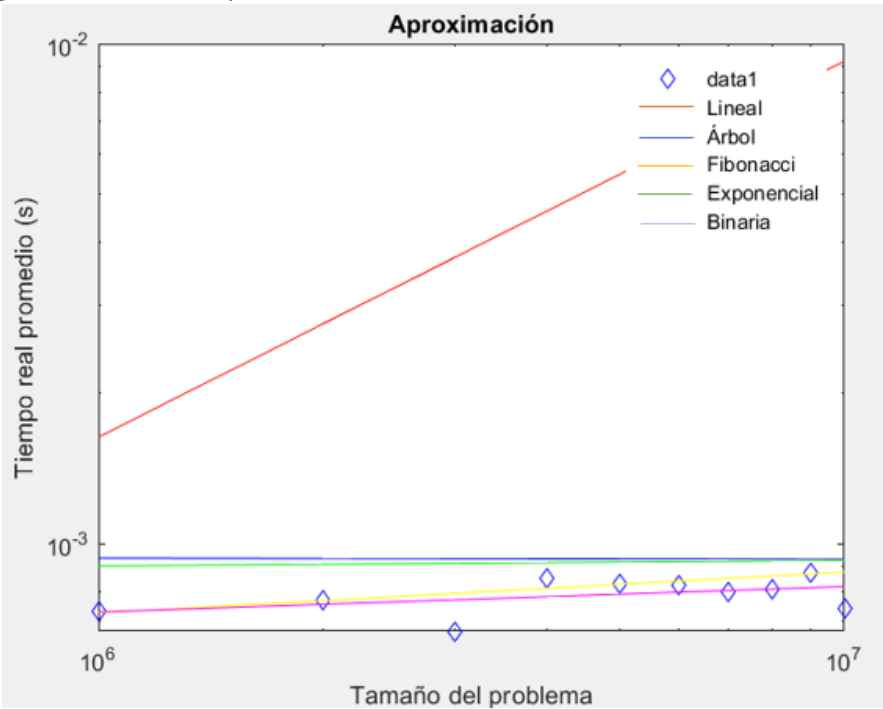
Curva 7.4 Aproximación del comportamiento temporal de la búsqueda exponencial



Curva 7.5 Aproximación del comportamiento temporal de la búsqueda de Fibonacci



Punto 8: Gráfica comparativa de las aproximaciones del comportamiento temporal de los 5 algoritmos de búsqueda



Punto 9: Constante multiplicativa

| Algoritmo | Peor caso | Prueba con el peor caso (tiempo en segundos) | Función teórica del peor caso (Cantidad de operaciones) | Constante multiplicativa (segundos/operación) |
|-----------|-----------------|--|---|---|
| Lineal | n = 10 millones | 2.425.02 | 10000000 | 2.425.02 |
| | No ordenados | | | |
| | x = 61396 | | | |

| | | | | |
|--------------------|---|----------|-------------|-----------------|
| Binaria | n = 10 millones | 3.10E-06 | 93.01398666 | 3.33E-08 |
| | Ordenados | | | |
| | x = 61396 | | | |
| | Un elemento que no existe en los 10 millones de números | | | |
| Exponencial | n = 10 millones | 1.91E-06 | 186.0279733 | 1.03E-08 |
| | Ordenados | | | |
| | x = 2147483428 | | | |
| | Un elemento que no existe pero de existir estaría en el último segmento del arreglo | | | |
| Fibonacci | n = 10 millones | 9.54E-07 | 1168.225101 | 8.16E-10 |
| | Ordenados | | | |
| | x = 2147483428 | | | |
| | Un elemento que no existe pero de existir estaría en el último segmento del arreglo | | | |

Punto 10: Determinación de los tiempos de búsqueda para diferentes tamaños del problema con base en la función ajustada del peor caso

| Algoritmo | Función ajustada para el peor caso | Constante multiplicativa | Valor de n | Tiempo estimado de búsqueda (segundos) |
|-------------|------------------------------------|--------------------------|------------|--|
| Lineal | 50000000 | 2.43E-09 | 50000000 | 1.22E-01 |
| | 100000000 | | 100000000 | 2.43E-01 |
| | 500000000 | | 500000000 | 1.22E+00 |
| | 1000000000 | | 1000000000 | 2.43E+00 |
| | 5000000000 | | 5000000000 | 1.22E+01 |
| Binaria | 102.301699 | 3.33E-08 | 50000000 | 3.41E-06 |
| | 106.301699 | | 100000000 | 3.54E-06 |
| | 115.5894114 | | 500000000 | 3.85E-06 |
| | 119.5894114 | | 1000000000 | 3.985E-06 |
| | 128.8771238 | | 5000000000 | 4.29448E-06 |
| Árbol AVL | 127.8771238 | 1.48E-07 | 50000000 | 1.88802E-05 |
| | 132.8771238 | | 100000000 | 1.96184E-05 |
| | 144.4867643 | | 500000000 | 2.13325E-05 |
| | 149.4867643 | | 1000000000 | 2.20707E-05 |
| | 161.0964047 | | 5000000000 | 2.37848E-05 |
| Exponencial | 204.6033981 | 1.03E-08 | 50000000 | 2.0978E-06 |
| | 212.6033981 | | 100000000 | 2.17983E-06 |
| | 231.1788228 | | 500000000 | 2.37028E-06 |
| | 239.1788228 | | 1000000000 | 2.45231E-06 |
| | 257.7542476 | | 5000000000 | 2.64276E-06 |
| | 1279.328709 | 8.16E-10 | 50000000 | 1.04437E-06 |

Punto 11: Mejoras de los algoritmos usando Threads

Tabla 11: Comparativa de los tiempos promedios de los algoritmos en sus versiones originales y en sus versiones con hilos.

| Algoritmo | Tamaño de n | Tiempo real promedio (versión sin hilos) | Tiempo real promedio (versión con hilos) | Mejora |
|-------------|-------------|--|--|-------------|
| Lineal | 1,000,000 | 1.80E-03 | 1.50E-03 | 120% |
| | 2,000,000 | 0.002637243 | 0.002069747 | 1.274186024 |
| | 3,000,000 | 0.003495455 | 0.00237478 | 147% |
| | 4,000,000 | 0.004341638 | 0.002592182 | 1.674896987 |
| | 5,000,000 | 0.005462074 | 0.003014588 | 181% |
| | 6,000,000 | 0.006128943 | 0.003287613 | 1.864252951 |
| | 7,000,000 | 0.006999063 | 0.003718114 | 188% |
| | 8,000,000 | 0.007558894 | 0.004095924 | 1.845467431 |
| | 9,000,000 | 0.008976912 | 0.004257715 | 211% |
| | 10,000,000 | 0.010112357 | 0.004619992 | 2.188825786 |
| Árbol AVL | 1,000,000 | 0.000922227 | 0.000592613 | 156% |
| | 2,000,000 | 0.000970972 | 0.000636089 | 1.526471636 |
| | 3,000,000 | 0.000833464 | 0.000702667 | 119% |
| | 4,000,000 | 0.000962472 | 0.000673866 | 1.428283329 |
| | 5,000,000 | 0.001023459 | 0.000696993 | 147% |
| | 6,000,000 | 0.000965285 | 0.000968516 | 0.99666441 |
| | 7,000,000 | 0.000909293 | 0.000972354 | 94% |
| | 8,000,000 | 0.000920165 | 0.000964212 | 0.954317294 |
| | 9,000,000 | 0.000876212 | 0.000992858 | 88% |
| | 10,000,000 | 0.000935459 | 0.000729108 | 1.283018868 |
| Binaria | 1,000,000 | 0.000731754 | 0.001406002 | 52% |
| | 2,000,000 | 0.000771832 | 0.001408029 | 0.548165332 |
| | 3,000,000 | 0.000667727 | 0.00151751 | 44% |
| | 4,000,000 | 0.000851357 | 0.001625562 | 0.523730951 |
| | 5,000,000 | 0.000830579 | 0.001684773 | 49% |
| | 6,000,000 | 0.000823128 | 0.001693416 | 0.486075718 |
| | 7,000,000 | 0.000800455 | 0.001778293 | 45% |
| | 8,000,000 | 0.000808811 | 0.001663244 | 0.486285415 |
| | 9,000,000 | 0.000871813 | 0.001711988 | 51% |
| | 10,000,000 | 0.000740504 | 0.001701307 | 0.435256033 |
| Exponencial | 1,000,000 | 0.000875664 | 0.000928593 | 94% |
| | 2,000,000 | 0.000937891 | 0.00100863 | 0.929866446 |
| | 3,000,000 | 0.000947976 | 0.001077998 | 88% |
| | 4,000,000 | 0.000818825 | 0.001156211 | 0.708196721 |
| | 5,000,000 | 0.000949478 | 0.001119518 | 85% |
| | 6,000,000 | 0.000965202 | 0.001174223 | 0.821991655 |
| | 7,000,000 | 0.000994802 | 0.001099193 | 91% |
| | 8,000,000 | 0.000925291 | 0.001213753 | 0.762338313 |
| | 9,000,000 | 0.000894165 | 0.001037359 | 86% |
| | 10,000,000 | 0.000870275 | 0.001005411 | 0.865591653 |
| | 1.000.000 | 0.000776434 | 0.000973928 | 80% |

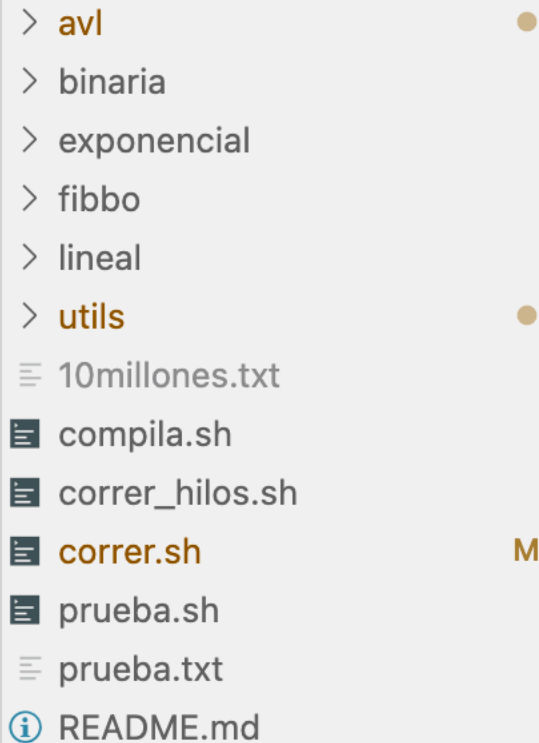
Punto 12: Cuestionario

1. ¿Cuál de los 5 algoritmos es el más fácil de implementar?
El algoritmo lineal.
2. ¿Cuál de los 5 algoritmos es el más difícil de implementar?
El árbol AVL.
3. ¿Cuál de los 5 algoritmos fue el más difícil de implementar en su variante con hilos?
Árboles, binario.
4. ¿Cuál de los 5 algoritmos resultó ser más rápido en su variante con hilos? ¿Por qué?
EL lineal, debido a que se crean varias instancias con la misma tarea, por lo que se divide la tarea entre el numero de hilos que se ejecuten.
5. ¿Cuál de los 5 algoritmos no representó una ventaja en su variante con hilos? ¿Por qué?
El árbol binario, porque por definición siempre se descartaba al mitad.
6. ¿Cuál algoritmo tiene menor complejidad temporal?
El algoritmo del árbol binario y el fibbonacci.
7. ¿Cuál algoritmo tiene mayor complejidad temporal?
El lineal.
8. ¿El comportamiento experimental de los algoritmos era el esperado? ¿Por qué?
Sí, porque los métodos teoricos nos arrojaban eso.
9. ¿Sus resultados experimentales difieren mucho de los análisis teóricos que realizó?
¿A qué se debe?
No mucho, si graficábamos 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?
Solo en el lineal, se aproximaba, pero no lo logró, por el costo computacional que genera crear los hilos.
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?
Sí, 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 realizara el análisis teórico de un algoritmo antes de implementarlo, podrías asegurar cuál es el mejor?
Sí.
14. ¿Qué tan difícil fue realizar el análisis teórico de cada algoritmo?

- Yash Varyani. (2020). Fibonacci Search. 20/11/2020, de Geeks4Geeks
Sitio web: <https://www.geeksforgeeks.org/fibonacci-search/>

Anexos.

Estructura de la práctica.



A screenshot of a file explorer window showing the project structure. The files are listed in a tree view. The 'avl' directory is expanded, showing its contents. The 'utils' directory is also expanded. The '10millones.txt' file is selected. The 'compila.sh', 'correr_hilos.sh', 'correr.sh', 'prueba.sh', 'prueba.txt', and 'README.md' files are listed at the bottom. The 'correr.sh' file is highlighted with a yellow background. The 'README.md' file has an information icon next to it.

- > avl
- > binaria
- > exponencial
- > fibbo
- > lineal
- > utils
- ≡ 10millones.txt
- ≡ compila.sh
- ≡ correr_hilos.sh
- ≡ correr.sh
- ≡ prueba.sh
- ≡ prueba.txt
- ⓘ README.md

avl/avl.c

```
#include<stdlib.h>
#include"avl.h"
```

```
AVL iniAVL(void)
{
    return NULL;
}
```

```
AVL roteIzq(AVL a)
{
    AVL temp = a -> der;
    a -> der = temp -> izq;
    temp -> izq = a;
    return temp;
}
```

```
AVL roteDer(AVL a)
{
    AVL temp = a -> izq;
    a -> izq = temp -> der;
```

```

AVL roteIzqDer(AVL a)
{
    a -> izq = roteIzq(a -> izq);
    return roteDer(a);
}

AVL balanceaDer(AVL a)
{
    if(a -> der -> balan == DER)
    {
        a -> balan = a -> der -> balan = BAL;
        a = roteIzq(a);
    }
    else
    {
        switch(a -> der -> izq -> balan)
        {
            case IZQ:
            {
                a -> balan = BAL;
                a -> der -> balan = DER;
                break;
            }
            case BAL:
            {
                a -> balan = a -> der -> balan = BAL;
                break;
            }
            case DER:
            {
                a -> balan = IZQ;
                a -> der -> balan = BAL;
                break;
            }
        }
        a -> der -> izq -> balan = BAL;
        a = roteDerIzq(a);
    }
    return a;
}

AVL balanceaIzq(AVL a)
{
    if(a -> izq -> balan == IZQ)
    {
        a -> balan = a -> izq -> balan = BAL;
        a = roteDer(a);
    }
    else
    {
        switch(a -> izq -> der -> balan)
        {
            case DER:

```

```

        {
            a -> balan = DER;
            a -> izq -> balan = BAL;
            break;
        }
    }
    a -> izq -> der -> balan = BAL;
    a = roteIzqDer(a);
}
return a;
}

```

AVL insAVL(AVL a, TipoAVL elem)

```

{
    AVL p = (AVL) malloc(sizeof(TAVL));
    int masAlto;
    p -> izq = p -> der = NULL;
    p -> info = elem;
    p -> balan = BAL;
    return insertar(a, p, &masAlto);
}

```

AVL insertar(AVL a, struct NodoAVL* p, int* masAlto)

```

{
    if(a == NULL)
    {
        *masAlto = TRUE;
        a = p;
    }
    else if(a -> info > p -> info)
    {
        a -> izq = insertar(a -> izq, p, masAlto);
        if(*masAlto)
            switch(a -> balan)
            {
                case IZQ:
                {
                    *masAlto = FALSE;
                    a = balanceaIzq(a);
                    break;
                }
                case BAL:
                {
                    a -> balan = IZQ;
                    break;
                }
                case DER:
                {
                    *masAlto = FALSE;
                    a -> balan = BAL;
                    break;
                }
            }
    }
}

```

```

        break;
    }
    case BAL:
    {
        a -> balan = DER;
        break;
    }
    case DER:
    {
        *masAlto = FALSE;
        a = balanceaDer(a);
    }
    }
}

return a;
}

int estaAVL(AVL a, TipoAVL elem)
{
    while(a != NULL && a -> info != elem)
        a = (elem < (a -> info)) ? a -> izq : a -> der;
    return a != NULL;
}

void destruirAVL(AVL a)
{
    if(a != NULL)
    {
        destruirAVL(a -> izq);
        destruirAVL(a -> der);
        free(a);
    }
}

```

avl/avl.h

```

#pragma once
#include "TipoAVL.h"

#define FALSE 0
#define TRUE 1

#define IZQ 1
#define BAL 0
#define DER -1

struct NodoAVL
{
    TipoAVL info;
    struct NodoAVL* izq;
    struct NodoAVL* der;
    int balan;
}

```

```

/* Rutina para la rotación a la izquierda (no recalcula el factor de balance) */
AVL roteIzq(AVL a);

/* Rutina para la rotación a la derecha (no recalcula el factor de balance) */
AVL roteDer(AVL a);

/* Rutina para la doble rotación derecha-izquierda (no recalcula el factor de balance) */
AVL roteDerIzq(AVL a);

/* Rutina para la doble rotación izquierda-derecha (no recalcula el factor de balance) */
AVL roteIzqDer(AVL a);

/* Rutina para balancer el subárbol derecho de un árbol AVL. Acutaliza los indicadores que lo requieran. */
AVL balanceaDer(AVL a);

/* Rutina para balancer el subárbol izquierdo de un árbol AVL. Acutaliza los indicadores que lo requieran. */
AVL balanceaIzq(AVL a);

/* Rutina que reibe un árbol AVL y un nodo con el elemento que se quiere agregar, y retorna funcionalmente un árbol AVL con el nuevo nodo agregado a la estructura inicial. Informa, en un parámetro por referencia, si la altura del árbol resultante es mayor que la altura del árbol inicial.*/
AVL insertar(AVL a, struct NodoAVL* p, int* masAlto);

// ----- Modificadoras -----
/* FUNCIÓN:      Adiciona un elemento a árbol AVL.
*/
AVL insAVL(AVL a, TipoAVL elem);

/* FUNCIÓN: Elimina un elemento de un árbol AVL.
*/
//AVL elimAVL(AVL a, TipoAVL elem);

// ----- Analizadora -----
/* FUNCIÓN:      Informa si un elemento se encuentra en el árbol AVL.
*/
int estaAVL(AVL a, TipoAVL elem);

// ----- Destructora -----
void destruirAVL(AVL a);

```

avl/TipoAVL.h

```
#pragma once
```

```
typedef int TipoAVL;
```

avl/main.c

```

#include <stdlib.h>
#include "avl.h"
#include "../utils/util.h"
#include "../utils/tiempo.h"

AVL carga_avl(AVL arbol, int *arreglo, int n);

/*
FUNCIÓN: main(int argc, const char **argv)
DESCRIPCIÓN: Main del programa
RECIBE: int argc (número de argumentos recibidos), const char **argv (argumentos
recibidos)
DEVUELVE: int 0
OBSERVACIONES: puede recibir el número n como argumento, si no se lo recibe lo
solicita en la hora de ejecución
*/

// MAIN
int main(int argc, const char **argv)
{
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    int i;
    int *arreglo;
    AVL arbol = iniAVL();

    int n = 0; // cantidad de números
    int x = 0; // número a buscar
    if (argc >= 2)
    {
        n = atoi(argv[1]);
        if (argc > 2)
        {
            x = atoi(argv[2]);
        }
        else
        {
            x = obtener_n();
        }
    }
    else
    {
        n = obtener_n();
        x = obtener_n();
    }

    arreglo = leer_archivo(arreglo, n);
    arbol = carga_avl(arbol, arreglo, n);

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

    if(estaAVL(arbol, x))
        printf("ESTÁ!!");
}

```



```

        return 0;
    }

/* FUNCIÓN: carga_arbol_binario_busqueda(Arbol arbol, int *apt_arreglo, int n)
 * DESCRIPCIÓN: carga el arbol binario de búsqueda con los números del arreglo
 * RECIBE: AVL arbol (arbol binario vacío e inicializado), int *apt_arreglo, int
n (total de números)
 * DEVUELVE: void.
 * */

AVL carga_avl(AVL arbol, int *arreglo, int n)
{
    int i;

    for (i = 0; i < n; i++)
    {
        arbol = insAVL(arbol, arreglo[i]);
    }

    return arbol;
}

```

avl/main_threads.c

```

/*
    AUTORES (C) 2020:
        Hernández López Moises
        Herrera Merino Roxana Angélica
        Jiménez Delgado Luis Diego
    VERSIÓN: 1.0
    DESCRIPCIÓN: Implementación del algoritmo de búsqueda en un árbol AVL.
*/

#include <stdio.h>
#include <stdlib.h>
#include "avl.h"
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <pthread.h>
#include "../utils/util.h"
#include "../utils/tiempo.h"

AVL carga_avl(AVL arbol, int *arreglo, int n);
void busqueda(AVL a);

typedef struct informacion
{
    AVL arbol;
} informacion;

```

```

// MAIN
int main(int argc, const char **argv)
{
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    int i;
    int *arreglo;
    AVL arbol = iniAVL();

    int n = 0; // cantidad de números
    x = 0; // número a buscar
    if (argc >= 2)
    {
        n = atoi(argv[1]);
        if (argc > 2)
        {
            x = atoi(argv[2]);
        }
        else
        {
            x = obtener_n();
        }
    }
    else
    {
        n = obtener_n();
        x = obtener_n();
    }

    arreglo = leer_archivo(arreglo, n);
    arbol = carga_avl(arbol, arreglo, n);

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

    /*
    if(estaAVL(arbol, x))
        printf("Si se encontro el elemento %d en el arbol\n", x);
    else
        printf("No se encontro el elemento %d en el arbol\n", x);
    */
    uswtime(&utime1, &stime1, &wtime1);

    imprimir_tiempos(utime0, stime0, wtime0, utime1, stime1, wtime1);

    imprimir_arreglo(arreglo, n);

    destruirAVL(arbol);

    return 0;
}

```

```

        if (a != NULL)
        {
            posicion = 1;
        }
    }

void *thread_process(void *datos)
{
    informacion *info = datos;
    busqueda_arbol(info->arbol, x);
    pthread_exit(0);
}

void busqueda(AVL a)
{
    pthread_t thread_izq;
    pthread_t thread_der;
    informacion *info_izq = malloc(sizeof(info_izq));
    informacion *info_der = malloc(sizeof(info_der));

    info_izq->arbol = a->izq;
    info_der->arbol = a->der;

    int status, i, *exit_code, aux;

    if (a->info == x)
    {
        posicion = 1;
        return;
    }

    status = pthread_create(&thread_izq, NULL, thread_process, info_izq);
    if (status)
    {
        printf("\nError en thread %i\n", status);
        exit(-1);
    }

    status = pthread_create(&thread_der, NULL, thread_process, info_der);
    if (status)
    {
        printf("\nError en thread %i\n", status);
        exit(-1);
    }
    pthread_join(thread_izq, (void **)&exit_code);
    pthread_join(thread_der, (void **)&exit_code);
}

/* FUNCIÓN: carga_arbol_binario_busqueda(ArbinOr arbol, int *apt_arreglo, int n)
 * DESCRIPCIÓN: carga el arbol binario de búsqueda con los números del arreglo
 * RECIBE: AVL arbol (arbol binario vacío e inicializado), int *apt_arreglo, int
n (total de números)
 * DEVUELVE: void.

```

```

        return arbol;
    }

```

Binaria/main.c

```

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "../utils/util.h"
#include "../utils/tiempo.h"

int binaria(int *arr, int x, int n);

int main(int argc, const char **argv)
{
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    int n = 0;
    int x = 0;
    int *arr;
    int pos = -1;
    if (argc >= 2)
    {
        n = atoi(argv[1]);
        if (argc > 2)
        {
            x = atoi(argv[2]);
        }
        else
        {
            x = obtener_n();
        }
    }
    else
    {
        n = obtener_n();
        x = obtener_n();
    }
    arr = leer_archivo(arr, n);
    uswtime(&utime0, &stime0, &wtime0);
    pos = binaria(arr, n, x);
    uswtime(&utime1, &stime1, &wtime1);
    imprimir_resultado(pos);
    imprimir_tiempos(utime0, stime0, wtime0, utime1, stime1, wtime1);
    return 0;
}

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

    while (anterior <= siguiente)

```

```
}
```

Binaria/main_threads.c

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

#include <math.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <pthread.h>

#include "../utils/util.h"
#include "../utils/tiempo.h"
#define MAXTHREADS 8

typedef struct informacion
{
    int inicio;
    int fin;
} informacion;

int *arr;
int posicion = -1;
int x = 0;

int create_threads(int x, int n);
int calc_hilos(int n);

int main(int argc, const char **argv)
{
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    int n = 0;
    int pos = -1;
    if (argc >= 2)
    {
        n = atoi(argv[1]);
        if (argc > 2)
        {
            x = atoi(argv[2]);
        }
        else
        {
            x = obtener_n();
        }
    }
    else
    {
        n = obtener_n();
        x = obtener_n();
    }
    arr = leer_archivo(arr, n);
```

```

int anterior = info->inicio;
int siguiente = (info->fin) - 1;
int centro;

while (anterior <= siguiente && posicion == -1)
{
    centro = anterior + (siguiente - anterior) / 2;
    if (arr[centro] == x)
        posicion = centro;
    if (arr[centro] < x)
        anterior = centro + 1;
    else
        siguiente = centro - 1;
}
pthread_exit(0);
}

int create_threads(int x, int n)
{
    int hilos = calc_hilos(n);
    int actual = n / hilos;
    pthread_t threads[hilos];
    informacion *infos[hilos];
    int status, i, *exit_code;
    for (i = 0; i < (hilos - 1); ++i)
    {
        infos[i] = malloc(sizeof(*infos[i]));
        infos[i]->inicio = (i * actual);
        infos[i]->fin = ((i + 1) * actual - 1);
    }

    infos[hilos - 1] = malloc(sizeof(*infos[hilos]));
    infos[hilos - 1]->inicio = (i * actual);
    infos[hilos - 1]->fin = n;
    for (i = 0; i < hilos; ++i)
    {

        status = pthread_create(&threads[i], NULL, thread_process, infos[i]);
        if (status)
        {
            printf("\nError en thread %i\n", status);
            exit(-1);
        }
    }
    for (i = 0; i < hilos; i++)
    {
        pthread_join(threads[i], (void **)&exit_code);
    }
    return 0;
}

int calc_hilos(int n)
{

```

```

#include "../utils/tiempo.h"

int exponencial(int *arr, int x, int n);

int main(int argc, const char **argv)
{
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    int n = 0;
    int x = 0;
    int *arr;
    int pos = -1;
    if (argc >= 2)
    {
        n = atoi(argv[1]);
        if (argc > 2)
        {
            x = atoi(argv[2]);
        }
        else
        {
            x = obtener_n();
        }
    }
    else
    {
        n = obtener_n();
        x = obtener_n();
    }
    arr = leer_archivo(arr, n);
    uswtime(&utime0, &stime0, &wtime0);
    pos = exponencial(arr, x, n);
    uswtime(&utime1, &stime1, &wtime1);
    imprimir_resultado(pos);
    imprimir_tiempos(utime0, stime0, wtime0, utime1, stime1, wtime1);
    return 0;
}

int exponencial(int *arr, int x, int n)
{
    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)

```

```
}
```

Exponencial/main_threads.c

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

#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <pthread.h>

#include "../utils/util.h"
#include "../utils/tiempo.h"
#define MAXTHREADS 4

typedef struct informacion
{
    int inicio;
    int fin;
} informacion;

int *arr;
int posicion = -1;
int x = 0;

int create_threads(int x, int n);
int calc_hilos(int n);

int main(int argc, const char **argv)
{
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    int n = 0;
    int pos = -1;
    if (argc >= 2)
    {
        n = atoi(argv[1]);
        if (argc > 2)
        {
            x = atoi(argv[2]);
        }
        else
        {
            x = obtener_n();
        }
    }
    else
    {
        n = obtener_n();
        x = obtener_n();
    }
    arr = leer_archivo(arr, n);
    uswtime(&utime0, &stime0, &wtime0);
    create_threads(x, n);
```



```

int siguiente = (info->fin) - 1;
int centro;

while (anterior <= siguiente && posicion == -1)
{
    centro = anterior + (siguiente - anterior) / 2;
    if (arr[centro] == x)
        posicion = centro;
    if (arr[centro] < x)
        anterior = centro + 1;
    else
        siguiente = centro - 1;
}
pthread_exit(0);
}

int create_threads(int x, int n)
{
    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 elementos = (siguiente - anterior) + 1;
    int hilos = calc_hilos(elementos);
    int actual = elementos / hilos;
    pthread_t threads[hilos];
    informacion *infos[hilos];
    int status, *exit_code;
    for (i = 0; i < (hilos - 1); ++i)
    {
        infos[i] = malloc(sizeof(*infos[i]));
        infos[i]->inicio = anterior + (i * actual);
        infos[i]->fin = actual + ((i + 1) * actual);
    }

    infos[hilos - 1] = malloc(sizeof(*infos[hilos]));
    infos[hilos - 1]->inicio = anterior + (i * actual);
    infos[hilos - 1]->fin = siguiente;
    for (i = 0; i < hilos; ++i)
    {
        status = pthread_create(&threads[i], NULL, thread_process, infos[i]);
        if (status)
        {
            printf("\nError en thread %i\n", status);
        }
    }
}

```

```

{
    if (n == 1)
        return 1;
    int module = (n % MAXTHREADS);
    return (module == 0) ? MAXTHREADS : module;
}

```

Fibbo/main.c

```

#include <stdio.h>
#include <stdlib.h>
#include "../utils/util.h"
#include "../utils/tiempo.h"

int fibonacci(int *arr, int x, int n);
int obtener_menor(int x, int y);

int main(int argc, const char **argv)
{
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    int n = 0;
    int x = 0;
    int *arr;
    int pos = -1;
    if (argc >= 2)
    {
        n = atoi(argv[1]);
        if (argc > 2)
        {
            x = atoi(argv[2]);
        }
        else
        {
            x = obtener_n();
        }
    }
    else
    {
        n = obtener_n();
        x = obtener_n();
    }
    arr = leer_archivo(arr, n);
    uswtime(&utime0, &stime0, &wtime0);
    pos = fibonacci(arr, x, n);
    uswtime(&utime1, &stime1, &wtime1);
    imprimir_resultado(pos);
    imprimir_tiempos(utime0, stime0, wtime0, utime1, stime1, wtime1);
    return 0;
}

int fibonacci(int *arr, int x, int n)
{
    int ultimo_fibo = 1;

```

```

while (siguiente_fibo > 1)
{
    int i = obtener_menor(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;
}

```

Fibbo/main_threads.c

```

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

#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <pthread.h>

#include "../utils/util.h"
#include "../utils/tiempo.h"

typedef struct informacion
{
    int inicio;
    int fin;
    int medio;
    int tope;
} informacion;

void fibonacci();
int obtener_menor(int x, int y);

```

```

        n = atoi(argv[1]);
        if (argc > 2)
        {
            x = atoi(argv[2]);
        }
        else
        {
            x = obtener_n();
        }
    }
    else
    {
        n = obtener_n();
        x = obtener_n();
    }
    arr = leer_archivo(arr, n);
    uswtime(&utime0, &stime0, &wtime0);
    fibonacci(arr, x, n);
    uswtime(&utime1, &stime1, &wtime1);
    imprimir_resultado(posicion);
    imprimir_tiempos(utime0, stime0, wtime0, utime1, stime1, wtime1);
    return 0;
}

```

```

void *thread_process(void *datos)
{
    informacion *info = datos;

    int ultimo_fibo = info->inicio;
    int penultimo_fibo = info->medio;
    int siguiente_fibo = info->fin;

    int limite = 0;

    while (siguiente_fibo > info->tope && posicion == -1)
    {
        int i = obtener_menor(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;
            }
        }
    }
}

```

```

void fibonacci()
{
    int hilos = 2;

    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;
    }

    pthread_t threads[hilos];
    informacion *infos[hilos];
    int status, i, *exit_code, aux;
    int actual = n / hilos;
    for (i = 0; i < hilos; ++i)
    {
        infos[i] = malloc(sizeof(*infos[i]));
        infos[i]->inicio = ultimo_fibo;
        infos[i]->medio = penultimo_fibo;
        infos[i]->fin = siguiente_fibo;

        siguiente_fibo = penultimo_fibo;
        ultimo_fibo = ultimo_fibo - penultimo_fibo;
        penultimo_fibo = siguiente_fibo - ultimo_fibo;

        infos[i]->tope = 1;

        status = pthread_create(&threads[i], NULL, thread_process, infos[i]);
        if (status)
        {
            printf("\nError en thread %i\n", status);
            exit(-1);
        }
    }
    for (i = 0; i < hilos; i++)
    {
        pthread_join(threads[i], (void **)&exit_code);
    }
}

```

Lineal/main.c

```

#include <stdio.h>
#include <stdlib.h>
#include "../utils/util.h"
#include "../utils/tiempo.h"

int lineal(int *arr, int x, int n);

```

```

        if (argc > 2)
        {
            x = atoi(argv[2]);
        }
        else
        {
            x = obtener_n();
        }
    }
    else
    {
        n = obtener_n();
        x = obtener_n();
    }
    arr = leer_archivo(arr, n);
    uswtime(&utime0, &stime0, &wtime0);
    pos = lineal(arr, x, n);
    uswtime(&utime1, &stime1, &wtime1);
    imprimir_resultado(pos);
    imprimir_tiempos(utime0, stime0, wtime0, utime1, stime1, wtime1);
    return 0;
}

int lineal(int *arr, int x, int n)
{
    int i = 0;

    while (i < n)
    {
        if (arr[i] == x)
            return i;
        i++;
    }
    return -1;
}

```

Lineal/main_threads.c

```

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

#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <pthread.h>

#include "../utils/util.h"
#include "../utils/tiempo.h"
#define MAXTHREADS 8

typedef struct informacion
{
    int inicio;

```

```

{
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    int n = 0;
    int pos = -1;
    if (argc >= 2)
    {
        n = atoi(argv[1]);
        if (argc > 2)
        {
            x = atoi(argv[2]);
        }
        else
        {
            x = obtener_n();
        }
    }
    else
    {
        n = obtener_n();
        x = obtener_n();
    }
    arr = leer_archivo(arr, n);
    uswtime(&utime0, &stime0, &wtime0);
    create_threads(x, n);
    uswtime(&utime1, &stime1, &wtime1);
    imprimir_resultado(posicion);
    imprimir_tiempos(utime0, stime0, wtime0, utime1, stime1, wtime1);
    return 0;
}

```

```

void *thread_process(void *datos)
{
    informacion *info = datos;
    int i = info->inicio;
    while (i < info->fin && posicion == -1)
    {
        if (arr[i] == x)
        {
            posicion = i;
        }
        i++;
    }
    pthread_exit(0);
}

```

```

int create_threads(int x, int n)
{
    int hilos = calc_hilos(n);
    int actual = n / hilos;
    pthread_t threads[hilos];
    informacion *infos[hilos];
    int status, i, *exit_code;
    for (i = 0; i < (hilos - 1); ++i)
    {

```

```

        status = pthread_create(&threads[i], NULL, thread_process, infos[i]);
        if (status)
        {
            printf("\nError en thread %i\n", status);
            exit(-1);
        }
    }
    for (i = 0; i < hilos; i++)
    {
        pthread_join(threads[i], (void **)&exit_code);
    }
    return 0;
}

```

```

int calc_hilos(int n)
{
    if (n < MAXTHREADS)
        return 1;
    int module = (n % MAXTHREADS);
    if (module == 0)
        return MAXTHREADS;
    return module;
}

```

Utils/tiempo.c

```

//*****
//TIEMPO.C
//*****
//*****
//M. EN C. EDGARDO ADRIÁN FRANCO MARTÍNEZ
//Curso: Análisis de algoritmos
//(C) Enero 2013
//ESCOM-IPN
//Ejemplo de medición de tiempo en C y recepción de parametros en C bajo UNIX
//Compilación de la libreria: "gcc -c tiempo.c " (Generación del código objeto)
//*****

```

```

//*****
//Librerias incluidas
//*****
#include <sys/resource.h>
#include <sys/time.h>
#include "tiempo.h"

```

```

//*****
//uswtime (Definición)
//*****
//Descripción: Función que almacena en las variables referenciadas
//el tiempo de CPU, de E/S y Total actual del proceso actual.
//
//Recibe: Variables de tipo doble para almacenar los tiempos actuales
//

```



```

    *usertime = (double) buffer.ru_utime.tv_sec + 1.0e-6 *
buffer.ru_utime.tv_usec;
    *systemtime = (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;
}

```

/*En Unix, se dispone de temporizadores ejecutables (en concreto time) que nos proporcionan medidas de los tiempos de ejecución de programas. Estos temporizadores nos proporcionan tres medidas de tiempo:

- * real: Tiempo real que se ha tardado desde que se lanzó el programa a ejecutarse hasta que el programa finalizó y proporcionó los resultados.
- * user: Tiempo que la CPU se ha dedicado exclusivamente a la computación del programa.
- * sys: Tiempo que la CPU se ha dedicado a dar servicio al sistema operativo por necesidades del programa (por ejemplo para llamadas al sistema para efectuar I/O).

El tiempo real también suele recibir el nombre de elapsed time o wall time. Algunos temporizadores también proporcionan el porcentaje de tiempo que la CPU se ha dedicado al programa. Este porcentaje viene dado por la relación entre el tiempo de CPU (user + sys) y el tiempo real, y da una idea de lo cargado que se hallaba el sistema en el momento de la ejecución del programa.

El grave inconveniente de los temporizadores ejecutables es que no son capaces de proporcionar medidas de tiempo de ejecución de segmentos de código. Para ello, hemos de invocar en nuestros propios programas a un conjunto de temporizadores disponibles en la mayor parte de las librerías de C de Unix, que serán los que nos proporcionen medidas sobre los tiempos de ejecución de trozos discretos de código.

En nuestras prácticas vamos a emplear una función que actúe de temporizador y que nos proporcione los tiempos de CPU (user, sys) y el tiempo real. En concreto, vamos a emplear el procedimiento uswtime listado a continuación.

Este procedimiento en realidad invoca a dos funciones de Unix: getrusage y gettimeofday. La primera de ellas nos proporciona el tiempo de CPU, tanto de usuario como de sistema, mientras que la segunda nos proporciona el tiempo real (wall time). Estas dos funciones son las que disponen de mayor resolución de todos los temporizadores disponibles en Unix.

Modo de Empleo:

La función uswtime se puede emplear para medir los tiempos de ejecución de determinados segmentos de código en nuestros programas. De forma esquemática, el empleo de esta función constaría de los siguientes pasos:

- 1.- Invocar a uswtime para fijar el instante a partir del cual se va a medir el tiempo.

4.- Calcular los tiempos de ejecución como la diferencia entre la primera y segunda

invocación a `uswtime`:

```
real:   wtime1 - wtime0
user:   utime1 - utime0
sys :   stime1 - stime0
```

El porcentaje de tiempo dedicado a la ejecución de ese segmento de código

vendría dado por la relación CPU/Wall:

$$\text{CPU/Wall} = (\text{user} + \text{sys}) / \text{real} \times 100 \text{ \%}$$

Utils/tiempo.h

```

//*****
//TIEMPO.H
//*****
//*****
//M. EN C. EDGARDO ADRIÁN FRANCO MARTÍNEZ
//Curso: Análisis de algoritmos
//(C) Enero 2013
//ESCOM-IPN
//Ejemplo de medición de tiempo en C y recepción de parametros en C bajo UNIX
//Compilación de la libreria: "gcc -c tiempo.c " (Generación del código objeto)
//*****

//*****
//uswtime (Declaración)
//*****
//Descripción: Función que almacena en las variables referenciadas
//el tiempo de CPU, de E/S y Total actual del proceso actual.
//
//Recibe: Variables de tipo doble para almacenar los tiempos actuales
//Devuelve:
//*****
void uswtime(double *usertime, double *systime, double *walltime);
/*
Modo de Empleo:
La función uswtime se puede emplear para medir los tiempos de ejecución de
determinados segmentos de código en nuestros programas. De forma esquemática, el
empleo de esta función constaría de los siguientes pasos:
```

1.- Invocar a `uswtime` para fijar el instante a partir del cual se va a medir el tiempo.

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

2.- Ejecutar el código cuyo tiempo de ejecución se desea medir.

3.- Invocar a `uswtime` para establecer el instante en el cual finaliza la medición

El porcentaje de tiempo dedicado a la ejecución de ese segmento de código

vendría dado por la relación CPU/Wall:

$$\text{CPU/Wall} = (\text{user} + \text{sys}) / \text{real} \times 100 \text{ \%}$$

Utils/util.c

```
/*
    AUTORES (C) 2020:
        Hernández López Moises
        Herrera Merino Roxana Angélica
        Jiménez Delgado Luis Diego
    VERSIÓN: 1.0
    DESCRIPCIÓN: Implementación del algoritmo burbuja optimizado
*/

#include <stdio.h>
#include <stdlib.h>
#include "util.h"

/*
    FUNCIÓN: imprimir_arreglo(int *apt_arreglo, int n)
    DESCRIPCIÓN: funcion para imprimir arreglo de enteros
    RECIBE: int *apt_arreglo(el arreglo a imprimir), int n(el tamaño del arreglo)
    DEVUELVE: void
    OBSERVACIONES: desactivado por motivos prácticos
*/
void imprimir_arreglo(int *apt_arreglo, int n)
{
    int i;
    for (i = 0; i < n; i++)
    {
        //printf("%d \n", apt_arreglo[i]);
    }
}

/*
    FUNCIÓN: int obtener_n()
    DESCRIPCIÓN: funcion para solicitar una n al usuario
    RECIBE: void
    DEVUELVE: int (el número ingresado)
    OBSERVACIONES:
*/
int obtener_n()
{
    int n;
    scanf("%d", &n);
    return n;
}

/*
```

```

        for (i = 0; i < n; i++)
            scanf("%d", (apt_arreglo + i));

        return apt_arreglo;
    }

    /*
    FUNCIÓN: imprimir_tiempos(double utime0, double stime0, double wtime0, double
    utime1, double stime1, double wtime1)
    DESCRIPCIÓN: funcion para imprimir los tiempos totales
    RECIBE: double utime0, double stime0, double wtime0, double utime1, double
    stime1, double wtime1
    DEVUELVE: void
    OBSERVACIONES: leer time.h para más información
    */
    void imprimir_tiempos(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 acciones de E/S) %.10f s\n", stime1 - stime0);
        printf("CPU/Wall %.10f %% \n", 100.0 * (utime1 - utime0 + stime1 - stime0)
        / (wtime1 - wtime0));
        printf("\n");

        //Mostrar los tiempos en formato exponencial
        printf("\n");

        printf("real (Tiempo total) %.10e s\n", wtime1 - wtime0);
        printf("user (Tiempo de procesamiento en CPU) %.10e s\n", utime1 - utime0);
        printf("sys (Tiempo en acciones de E/S) %.10e s\n", stime1 - stime0);
        printf("CPU/Wall %.10f %% \n", 100.0 * (utime1 - utime0 + stime1 - stime0)
        / (wtime1 - wtime0));
        printf("\n");
        */
        printf("%.10e\n", wtime1 - wtime0);

    }

    void imprimir_resultado(int res)
    {
        if (res == -1)
        {
            printf("NO\n");
        }
        else
        {
            printf("SI\n");
        }
    }

```

```

        Hernández López Moises
        Herrera Merino Roxana Angélica
        Jiménez Delgado Luis Diego
    VERSIÓN: 1.0
    DESCRIPCIÓN: Implementación del algoritmo burbuja optimizado
*/

#include <stdio.h>

void imprimir_arreglo(int *apt_arreglo, int n);

int obtener_n();

void imprimir_resultado(int res);

int *leer_archivo(int *apt_arreglo, int n);

void imprimir_tiempos(double utime0, double stime0, double wtime0, double utime1,
double stime1, double wtime1);

int obtener_menor(int x, int y);

```

compila.sh

```

gcc ./avl/avl.c ./avl/main.c ./utils/util.c ./utils/tiempo.c -o ./avl/a.out
gcc ./avl/main_threads.c ./avl/avl.c ./utils/util.c ./utils/tiempo.c -o
./avl/e.out

gcc ./fibbo/main.c ./utils/util.c ./utils/tiempo.c -o ./fibbo/a.out
gcc ./lineal/main.c ./utils/util.c ./utils/tiempo.c -o ./lineal/a.out
gcc ./exponencial/main.c ./utils/util.c ./utils/tiempo.c -o ./exponencial/a.out
gcc ./binaria/main.c ./utils/util.c ./utils/tiempo.c -o ./binaria/a.out
gcc ./lineal/main_threads.c ./utils/util.c ./utils/tiempo.c -o ./lineal/e.out
gcc ./binaria/main_threads.c ./utils/util.c ./utils/tiempo.c -o ./binaria/e.out
gcc ./exponencial/main_threads.c ./utils/util.c ./utils/tiempo.c -o
./exponencial/e.out
gcc ./fibbo/main_threads.c ./utils/util.c ./utils/tiempo.c -o ./fibbo/e.out

```

correr_hilos.sh

```

x_array=(322486 14700764 3128036 6337399 61396 10393545 2147445644 1295390003
450057883 187645041 1980098116 152503 5000 1493283650 214826 1843349527
1360839354 2109248666 2147470852 0)
n_array=(1000000 2000000 3000000 4000000 5000000 6000000 7000000 8000000 9000000
10000000)

```

```

mkdir "$1/resultados"
echo "" > "$1/resultados/resultados_hilos.txt"
for n in ${n_array[*]}
do
    echo "VALOR DE N : $n" >> "$1/resultados/resultados_hilos.txt"
    for x in ${x_array[*]}

```

```
x_array=(322486 14700764 3128036 6337399 61396 10393545 2147445644 1295390003
450057883 187645041 1980098116 152503 5000 1493283650 214826 1843349527
1360839354 2109248666 2147470852 0)
n_array=(1000000 2000000 3000000 4000000 5000000 6000000 7000000 8000000 9000000
10000000)
```

```
mkdir "./$1/resultados"
echo "" > "./$1/resultados/resultados_e.txt"
for n in ${n_array[*]}
do
    echo "VALOR DE N : $n" >> "./$1/resultados/resultados_e.txt"
    for x in ${x_array[*]}
    do
        echo "n:$n x:$x"
        ./a.out "$n" "$x" < 10millones.txt >>
        "./$1/resultados/resultados_e.txt"
    done
done
```