



INSTITUTO POLITECNICO NACIONAL

Escuela Superior de Cómputo

INSTITUTO POLITÉCNICO NACIONAL



## PRACTICA 2. Análisis temporal

### Elaborado por

Lara Becerril Bogar Olaf

Calderón Rivera Mario Alberto

Romero De La Cruz Fernando

Juárez Laureano Rafael

### Unidad didáctica

Análisis y diseño de algoritmos

### Profesor

Franco Martínez Edgardo Adrián

A 20 de noviembre de 2022



## -Introducción-

### *Búsqueda Lineal:*

La búsqueda lineal se define como un algoritmo de búsqueda secuencial que comienza en un extremo y recorre cada elemento de una lista hasta encontrar el elemento deseado; de lo contrario, la búsqueda continúa hasta el final del conjunto de datos. Es el algoritmo de búsqueda más fácil.

### *ABB:*

- Un árbol de búsqueda binario es una estructura de datos de árbol binario basada en nodos que tiene las siguientes propiedades:
- El subárbol izquierdo de un nodo contiene solo nodos con claves menores que la clave del nodo.
- El subárbol derecho de un nodo contiene solo nodos con claves mayores que la clave del nodo.

El subárbol izquierdo y derecho también debe ser un árbol de búsqueda binaria. No debe haber nodos duplicados.

### *Búsqueda Binaria:*

La búsqueda binaria es un algoritmo eficiente para encontrar un elemento en una lista ordenada de elementos. Funciona al dividir repetidamente a la mitad la porción de la lista que podría contener al elemento, hasta reducir las ubicaciones posibles a solo una. Usamos la búsqueda binaria en el juego de adivinar en la lección introductoria.

### *Búsqueda exponencial :*

La busca exponencial deja buscar por medio de una lista no delimitada ordenada para un valor de entrada concretado (la llave de busca). El algoritmo consiste en 2 etapas. La primera etapa determina un rango en el que la llave de busca radicaría si estuviera en la lista. En la segunda etapa, se efectúa una busca binaria en ese rango. En la primera etapa, suponiendo que la lista está ordenada en orden ascendiente, el algoritmo busca el primer exponente  $j$ , donde el valor  $2^j$  es más grande que la llave de busca. Este valor,  $2^j$ , se transforma en el tope superior para la busca binaria con el poder precedente de dos,  $2^{j-1}$ , siendo el límite inferior para la busca binaria.

### *Búsqueda de Fibonacci*

En informática , la técnica de búsqueda de Fibonacci es un método de búsqueda de una matriz ordenada utilizando un algoritmo de dividir y conquistar que reduce las posibles ubicaciones con la ayuda de los números de Fibonacci. En comparación con la búsqueda binaria donde la matriz ordenada se divide en dos partes de igual tamaño, una de las cuales se examina más a fondo, la búsqueda de Fibonacci divide la matriz en dos partes que tienen tamaños que son números de Fibonacci consecutivos.



### -Hardware empleado-

- **Procesador:** Intel Core i5 1137G7 a 2.42 GHz
- **Memoria RAM:** 16 GB con 15.8 GB utilizable
- **Almacenamiento:** 250 GB SSD nvme Samsung
- **Tarjeta de video:** Nvidia MX450
- **Sistema operativo utilizado:** Linux – Ubuntu



## -Funciones complejidad-

### Búsqueda Lineal

```
for(int i=0; i<5; i++){ //inicializamos el for para recorrer el arreglo y buscar el numero
    if(numero == array[i]){ // comparamos el numero a buscar con todos los numeros del arreglo
        bandera=1; //prendemos la bandera para indicar que se encontro el numero
        break;
    }else { //en caso de que no se haya encontrado el numero se le asignara el valor de 0
        bandera=0;
    }
}
```

*Mejor caso:*

Se dará cuando nuestro número a buscar está en nuestra primera posición o el primer índice del arreglo

Índice 0 1 2 3 4 5 6  
Valor [10 , 8 , 3 , 1 , 40 , 89 , 28 ] n = 7

Buscamos : 10

A [0] = 10 ----- 1 comparación

$$f(n) = 1 \text{ (Constante)}$$

*Peor caso:*

Se dará si el número que buscamos está en la última posición del arreglo o en definitiva no se encuentra

Índice 0 1 2 3 4 5 6  
Valor [10 , 8 , 3 , 1 , 40 , 89 , 28 ] n = 7

A [0] = 10 ----- 1 comparación

A [1] = 8 ----- 2 comparación

A [2] = 3 ----- 3 comparación

A [3] = 1 ----- 4 comparación Buscamos: 28

A [4] = 40 ----- 5 comparación

A [5] = 89 ----- 6 comparación

A [6] = 28 ----- 28 comparación (True)



Se hicieron 7 comparaciones, por lo tanto podemos observar que serán n operaciones

$$f(n) = n$$

### Caso medio

Para el caso promedio se tienen que contemplar todos los caminos posibles en el algoritmo, entonces analizando que para el primer camino (mejor caso) se hará 1 comparación, para el segundo se harán 2 comparaciones, así hasta llegar al último elemento del arreglo con n comparaciones, además hay que agregar un caso que no se hubiera encontrado dentro del arreglo que igual se tuvieron que hacer n comparaciones y en cuanto a la probabilidad de casos se tienen  $n+1$  (los n caminos por cada elemento del arreglo + 1 donde no se encontrará el elemento a buscar). Por lo tanto, la función que representar es:

$$F(n) = \frac{1}{n+1} (1 + 2 + 3 + 4 + \dots + n + n)$$

$$F(n) = \frac{1}{n+1} \sum_{i=1}^n i + n$$

$$F(n) = \frac{1}{n+1} \left( \frac{n(n+1)}{2} + n \right)$$

Simplificando da:

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

## Árbol Binario de Búsqueda

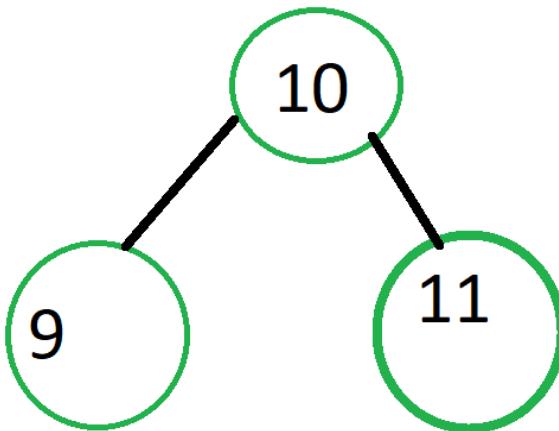
```
void busquedaArbolBinario(struct arbol *a, int numeroBuscado)//recibe una estructura del tipo arbol
{
    struct arbol *aux = a;//Se crea una estructura auxiliar con el fin de recibir la estructura
    int encontrado = 0;//Se crea una variable igual auxiliar para saber si el numero buscado se encontro
    while (aux != NULL && encontrado == 0)//Mientras nuestro auxiliar sea distinto de NULL y el numero no se encontro
    {
        //Se hara lo siguiente
        if (numeroBuscado == aux->numero)//Si el numero que buscamos ya esta en el nodo
        {
            encontrado = 1;//Se asignara 1 a encontrado rompiendo el ciclo ya que ya se encontro
        }
        else if (numeroBuscado < aux->numero)//Si no, el numero es menor al que tenemos
        {
            aux = aux->nodo_izq;//Descartamos el arbol que esta por la derecha
        }
        else if (numeroBuscado > aux->numero)//si no, entonces el numero que buscamos es mayor
        {
            aux = aux->nodo_der;//Entonces movera el auxiliar a la derecha para seguir buscando
        }
    }
}
```

Mejor caso:

Este se da cuando nuestro número está en la raíz del árbol, dando solo 1 comparación

$$F(x) = 1 \text{ (Constante)}$$

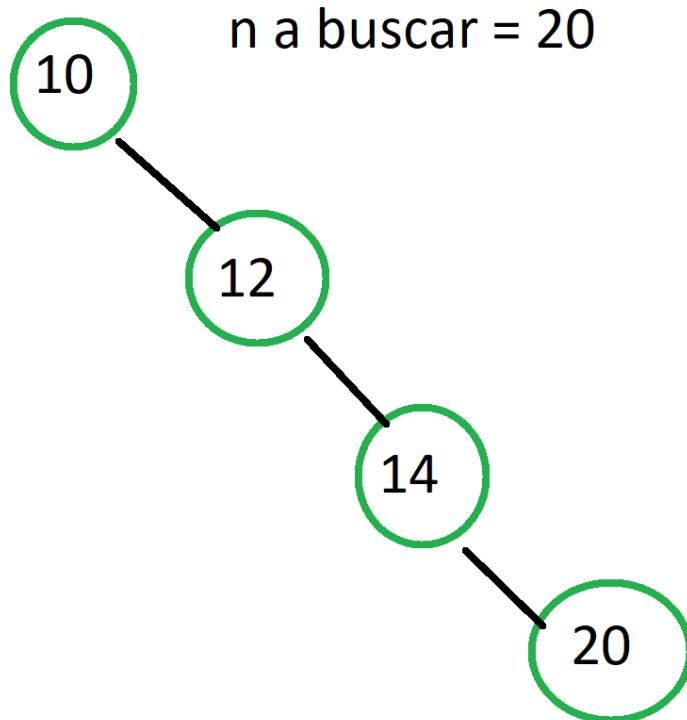
n a buscar = 10



*Peor caso*

Este se da cuando el número no esta o es el más pequeño o el más grande, haciendo que se haga un recorrido completo del árbol haciendo  $n$  comparaciones

$$F(x) = n \text{ (n comparaciones)}$$



*Caso medio:*

No encontramos uno, ya que para nosotros varían demasiado los árboles que no puede haber una receta que sea 100% el caso medio, y esto ¿Por qué?

Pues pensamos que hay infinidad de árboles posibles, a diferencia d ellos otros que solo revisamos el arreglo, aquí podemos variar mucho en los árboles, ya que estos pueden tender solamente hacia un lado, pueden ser completos o no, pueden ser muy pequeños o grandes, etc.

Así que decidimos sumar ambas complejidades y dividirlas entre 2

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

## Búsqueda Binaria

```
void busquedaBinaria(int *A, int inicio, int final, int numeroBuscado)
{
    int encontrado = 0;
    while (encontrado == 0 && inicio <= final)
    {
        int medio = (inicio + final) / 2;
        if (A[medio] == numeroBuscado)
        {
            encontrado = 1;
        }
        else if (A[medio] < numeroBuscado)
        {
            inicio = medio + 1;
        }
        else if (A[medio] > numeroBuscado)
        {
            final = medio - 1;
        }
    }
}
```

Mejor caso:

Este lo encontramos cuando el número buscado esta a la mitad del arreglo ya que realiza solo 1 comparación:

Número buscado: 1  
Índice 0 1 2 3 4 5 6  
Valor [10 , 8 , 3 , 1 , 40 , 89 , 28 ] n = 7

Esto dado por el hecho de que en la búsqueda binaria siempre se buscara primero por la mitad de nuestro arreglo

$$F(n) = 1$$

Pero caso:

Este ocurre cuando nuestro elemento a buscar está en alguna equina, ya sea la izquierda o la derecha de esta manera nuestro arreglo se dividirá en  $\log_2 n$  ya que nuestro arreglo se dividirá entre 2 n veces lo cual nos llevará a hacer la mayor cantidad de operaciones para sacar nuestro arreglo

$$F(n) = \log_2 n$$



Caso medio:

Para el análisis del caso medio se tiene que ver todos los caminos, entonces como caso inicial es el mejor caso, donde solo se hace una comparación, para la siguiente iteración ( $n/2$ ) se harían 2 comparaciones, para  $n/4$  se harían 3, y así sucesivamente hasta llegar al peor caso donde se harían  $\log_2 n$ , además habrá que contar en caso de que el elemento no se encuentre por lo que igual se harían  $\log_2 n$  comparaciones, en la parte de la probabilidad existirían  $\log_2 n$  posibles casos por cada división de la  $n$  más 1 adicional en caso de que no encontrarse.

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

$$f_t(n) = \frac{1}{\log_2 n + 1} \sum_{i=1}^{\log_2 n} i$$

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

$$f_t(n) = \frac{(\log_2 n)}{2}$$

### Exponencial:

```

int exponentialSearch(int arr[], int n, int x){
    // primero tomamos en cuenta el caso especial en el que nuestro target
    // este en el primer elemento
    if (arr[0] == x)  MEJOR CASO

    // Encontramos un rango de manera exponencial
    // en base 2 => 2 a la x
    int i = 1;
    while (i < n && arr[i] <= x)
        i = i*2;

    // Hacemos una búsqueda binaria una vez que encontramos el rango
    return binarySearch(arr, i/2, min(i, n-1), x);
}

/*
 * Esta función se encarga de hacer una búsqueda binaria recursiva
 */
int binarySearch(int arr[], int l, int r, int x){
    if (r >= l){
        int mid = l + (r - l)/2;

        // Condicional que verifica si nuestro target está en la mitad
        if (arr[mid] == x)
             return mid;

        // Si el elemento de en medio es menor, entonces buscamos en la izquierda
        if (arr[mid] > x)
            return binarySearch(arr, l, mid-1, x);

        // De otra manera buscamos en la derecha del rango
        return binarySearch(arr, mid+1, r, x);
    }

    // Si el elemento no se encuentra en el arreglo entonces retornamos -1
    return -1;
}

```

PEOR  
CASO

#### Mejor caso

Este se presentara cuando nuestro numero buscado esta en el primer índice del arreglo

Número buscado: 10

Índice 0 1 2 3 4 5 6

Valor [10 , 8 , 3 , 1 , 40 , 89 , 28 ] n = 7

De esta manera se hace solo 1 comparación dando como resultado

$$F(n) = 1 \text{ (Constante)}$$

#### Peor caso

Este es cuando no se encuentra el numero o bien esta al final del arreglo, sin embargo no supimos muy bien como representarlo ya que primero habíamos tomado como su función

$$F(n) = \log_2 n$$

Sin embargo, puede que en ocasiones se salga del rango al ser exponencial y esemos en algún lugar fuera de nuestro arreglo, lo cual vuelve a realizar mas operaciones, lo cual no supimos realmente como representarlo

### Caso medio

Al no saber como realizar el anterior caso, tampoco supimos muy bien como realizar este, pero decidimos tomarlo como una sumatoria de lo que sacamos y dividirlo entre 3 por los 3 caminos que puede tomar

1. 1 comparación por ser el primer elemento del arreglo
2. Entrando en el rango de  $\log_2 n$  de la primera vuelta
3. Y el último caso es donde debe repetirlo en mas ocasiones al salirse del rango

$$F(n) = \frac{\log_2 n + 1}{3}$$

### Fibonacci:

```

int fibonacciSearch(int arr[], int x, int n)
{
    /* Inicializamos los numero de fibonacci */
    int fibMMm2 = 0; // (m-2)'simo
    int fibMMm1 = 1; // (m-1)'simo
    int fibM = fibMMm2 + fibMMm1; // m'simo Fibonacci

    /* Aquí encontramos el menor numero de fibonacci me
     * que es mayor que el target
    while (fibM < n) {
        fibMMm2 = fibMMm1;
        fibMMm1 = fibM;
        fibM = fibMMm2 + fibMMm1;
    }

    // Marca que delimita nuestros elementos descartados
    int offset = -1;

    /* Intentamos cortar los elementos, fibM debe ser 1 ya que
     * el target es menor que el fib(m-2) es valido
    while (fibM > 1) {
        // checar que el fib(m-2) es valido
        int i = min(offset + fibMMm2, n - 1);

        /* Si el target es mayor, cortamos nuestro arreglo
        if (arr[i] < x) {
            fibM = fibMMm1;
            fibMMm1 = fibMMm2;
            fibMMm2 = fibM - fibMMm1;
            offset = i;
        }
        /* Si target es menor cortamos nuestro
        else if (arr[i] > x) {
            fibM = fibMMm2;
            fibMMm1 = fibMMm1 - fibMMm2;
            fibMMm2 = fibM - fibMMm1;
        }
        /* Retornamos el indice del elemento encontrado
        else return i;
    }

    /* Esta condición entra cuando hay 3 elementos
    if (fibMMm1 && arr[offset + 1] == x) {
        printf("encontro");
        return offset + 1;
    }

    /* Si el elemento no se encontro entonces retorna -1;
}

```



## -Registros de tiempo-

N = 1,000,000

| Algoritmos de Búsqueda |             |   |
|------------------------|-------------|---|
| Algoritmo              | Tamaño de n | Tiempo promedio de todas las búsquedas (ns) |
| Búsqueda lineal        | 1,000,000   | 3,386,450.147                               |
| Búsqueda en un ABB     | 1,000,000   | 1985.589                                    |
| Búsqueda Binaria       | 1,000,000   | 2357.601                                    |
| Búsqueda Exponencial   | 1,000,000   | 1879.254                                    |
| Búsqueda Fibonacci     | 1,000,000   | 2503.690                                    |

N = 2,000,000

| Algoritmos de Búsqueda |             |   |
|------------------------|-------------|---|
| Algoritmo              | Tamaño de n | Tiempo promedio de todas las búsquedas (ns) |
| Búsqueda lineal        | 2,000,000   | 5,101,698.25                                |
| Búsqueda en un ABB     | 2,000,000   | 2,425.25                                    |
| Búsqueda Binaria       | 2,000,000   | 2,415.32                                    |
| Búsqueda Exponencial   | 2,000,000   | 2,035.12                                    |
| Búsqueda Fibonacci     | 2,000,000   | 2,956.36                                    |

N = 3,000,000

| Algoritmos de Búsqueda |             |   |
|------------------------|-------------|---|
| Algoritmo              | Tamaño de n | Tiempo promedio de todas las búsquedas (ns) |
| Búsqueda lineal        | 3,000,000   | 7,125,648.21                                |
| Búsqueda en un ABB     | 3,000,000   | 2,458.36                                    |
| Búsqueda Binaria       | 3,000,000   | 2,415.32                                    |
| Búsqueda Exponencial   | 3,000,000   | 2,105.56                                    |
| Búsqueda Fibonacci     | 3,000,000   | 3,154.12                                    |

N = 4,000,000

| Algoritmos de Búsqueda |             |   |
|------------------------|-------------|---|
| Algoritmo              | Tamaño de n | Tiempo promedio de todas las búsquedas (ns) |
| Búsqueda lineal        | 4,000,000   | 10,274,106.32                               |
| Búsqueda en un ABB     | 4,000,000   | 2,347.57                                    |
| Búsqueda Binaria       | 4,000,000   | 2,648.47                                    |
| Búsqueda Exponencial   | 4,000,000   | 2,234.89                                    |
| Búsqueda Fibonacci     | 4,000,000   | 2,989.57                                    |



N = 5,000,000

| Algoritmos de Búsqueda |             |   |
|------------------------|-------------|---|
| Algoritmo              | Tamaño de n | Tiempo promedio de todas las búsquedas (ns) |
| Búsqueda lineal        | 5,000,000   | 13,251,578.25                               |
| Búsqueda en un ABB     | 5,000,000   | 2,584.24                                    |
| Búsqueda Binaria       | 5,000,000   | 2,894.15                                    |
| Búsqueda Exponencial   | 5,000,000   | 2,147.59                                    |
| Búsqueda Fibonacci     | 5,000,000   | 3,142.28                                    |

N = 6,000,000

| Algoritmos de Búsqueda |             |   |
|------------------------|-------------|---|
| Algoritmo              | Tamaño de n | Tiempo promedio de todas las búsquedas (ns) |
| Búsqueda lineal        | 6,000,000   | 15,257,771.47                               |
| Búsqueda en un ABB     | 6,000,000   | 2,587.25                                    |
| Búsqueda Binaria       | 6,000,000   | 3,025.96                                    |
| Búsqueda Exponencial   | 6,000,000   | 2,487.32                                    |
| Búsqueda Fibonacci     | 6,000,000   | 3,227.44                                    |

N = 7,000,000

| Algoritmos de Búsqueda |             |   |
|------------------------|-------------|---|
| Algoritmo              | Tamaño de n | Tiempo promedio de todas las búsquedas (ns) |
| Búsqueda lineal        | 7,000,000   | 17,215,744.55                               |
| Búsqueda en un ABB     | 7,000,000   | 2,154.35                                    |
| Búsqueda Binaria       | 7,000,000   | 3,785.41                                    |
| Búsqueda Exponencial   | 7,000,000   | 2,548.31                                    |
| Búsqueda Fibonacci     | 7,000,000   | 3,216.54                                    |

N = 8,000,000

| Algoritmos de Búsqueda |             |   |
|------------------------|-------------|---|
| Algoritmo              | Tamaño de n | Tiempo promedio de todas las búsquedas (ns) |
| Búsqueda lineal        | 8,000,000   | 19,321,653.22                               |
| Búsqueda en un ABB     | 8,000,000   | 3,654.11                                    |
| Búsqueda Binaria       | 8,000,000   | 3,442.12                                    |
| Búsqueda Exponencial   | 8,000,000   | 2,215.43                                    |
| Búsqueda Fibonacci     | 8,000,000   | 4,023.11                                    |



N = 9,000,000

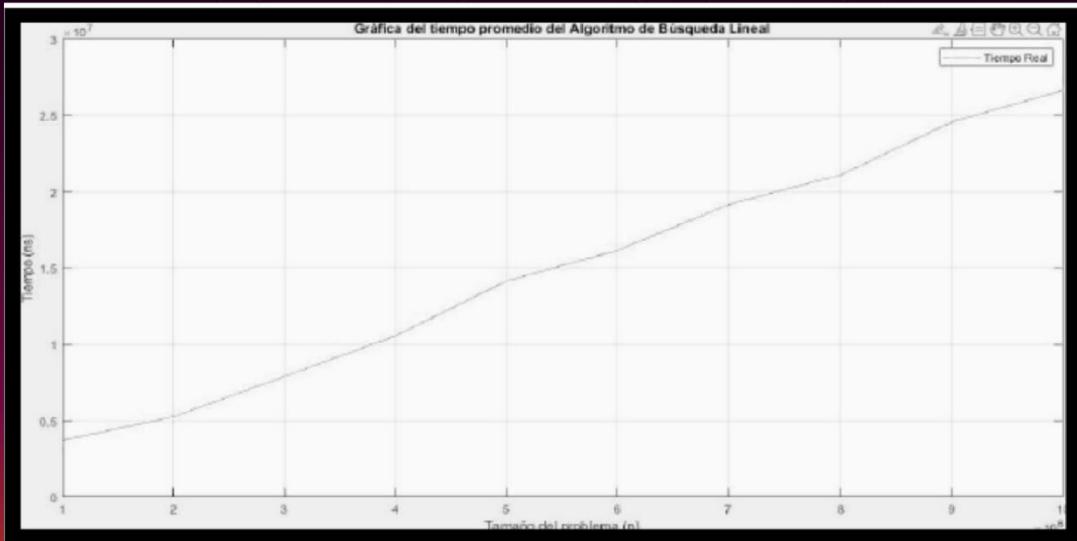
| Algoritmos de Búsqueda |             |   |
|------------------------|-------------|---|
| Algoritmo              | Tamaño de n | Tiempo promedio de todas las búsquedas (ns) |
| Búsqueda lineal        | 9,000,000   | 21,213,452.15                               |
| Búsqueda en un ABB     | 9,000,000   | 3,564.20                                    |
| Búsqueda Binaria       | 9,000,000   | 3,133.20                                    |
| Búsqueda Exponencial   | 9,000,000   | 2,452.01                                    |
| Búsqueda Fibonacci     | 9,000,000   | 3,213.54                                    |

N = 10,000,000

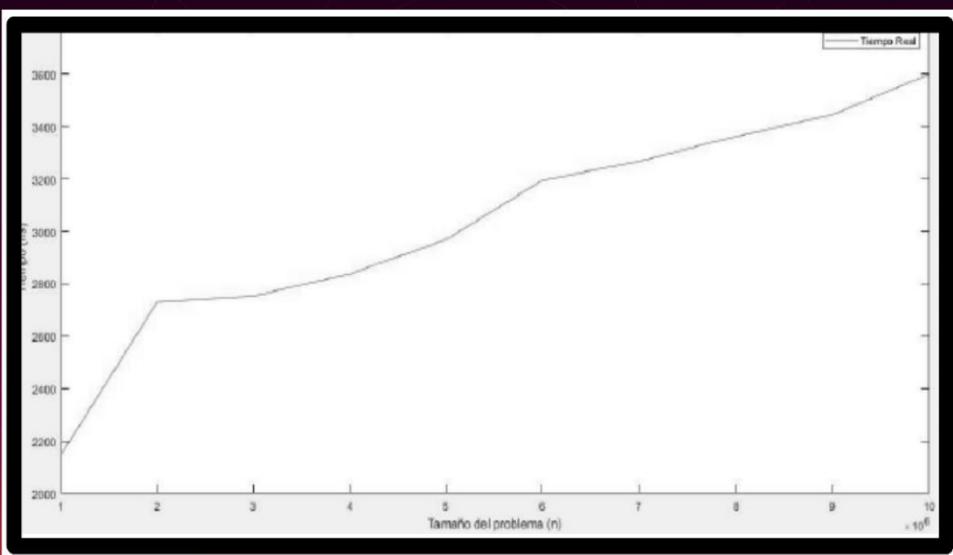
| Algoritmos de Búsqueda |             |   |
|------------------------|-------------|---|
| Algoritmo              | Tamaño de n | Tiempo promedio de todas las búsquedas (ns) |
| Búsqueda lineal        | 10,000,000  | 24,545,314.65                               |
| Búsqueda en un ABB     | 10,000,000  | 3,231.17                                    |
| Búsqueda Binaria       | 10,000,000  | 3,564.68                                    |
| Búsqueda Exponencial   | 10,000,000  | 2,215.41                                    |
| Búsqueda Fibonacci     | 10,000,000  | 5,654.30                                    |

## -Graficas de tiempo promedio-

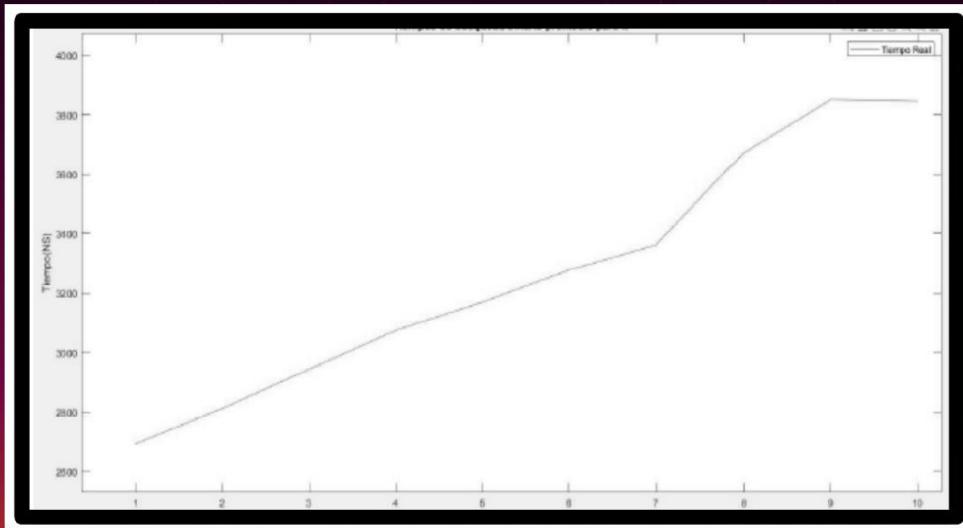
### Búsqueda Lineal



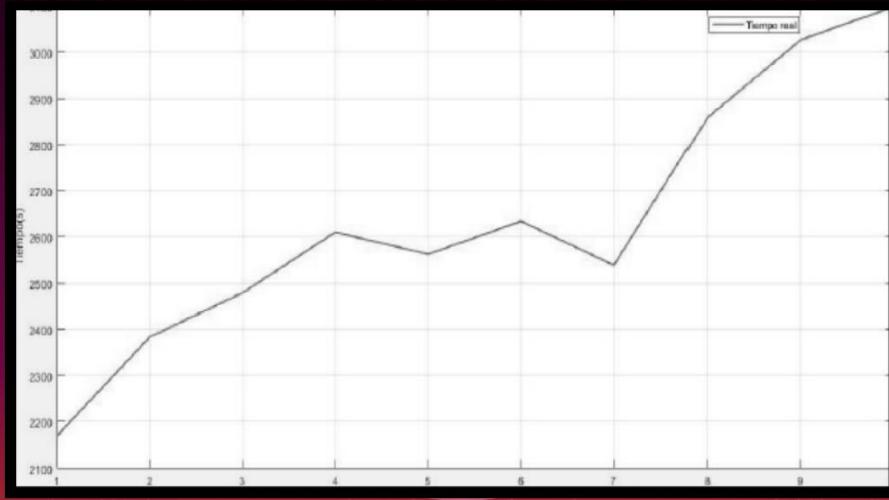
### Búsqueda en ABB



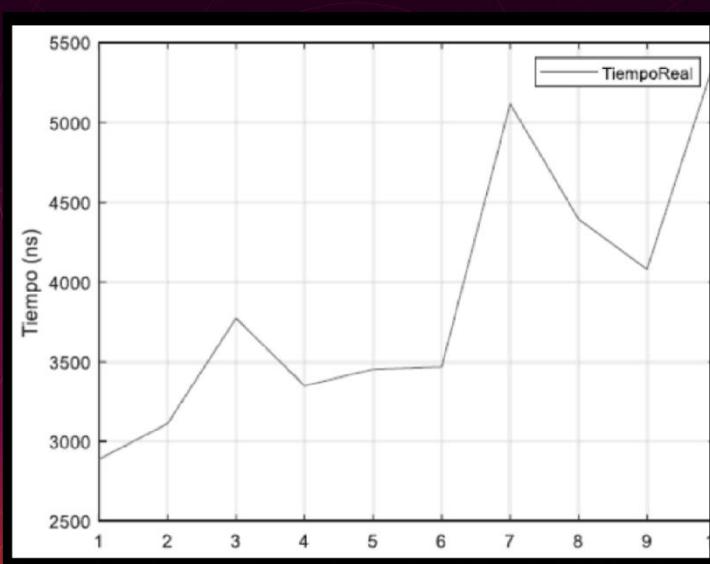
## Búsqueda Binaria o dicotómica



## Búsqueda Exponencial

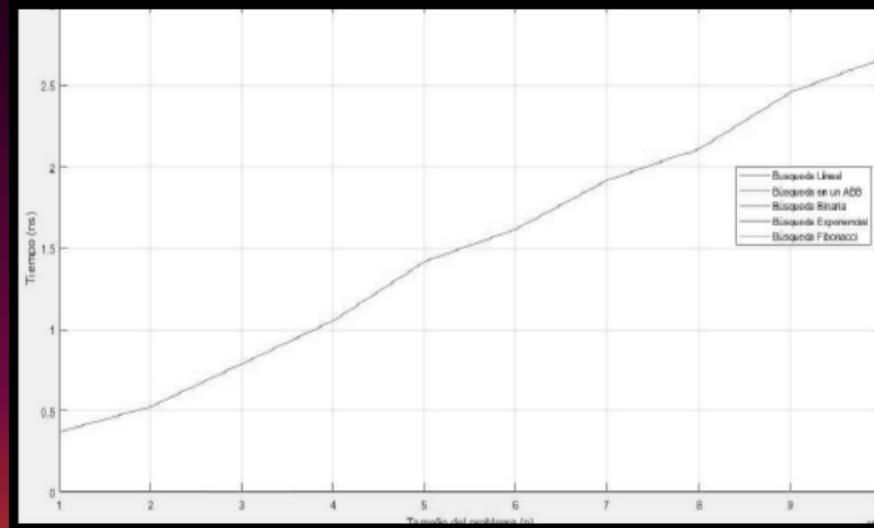


## Búsqueda Fibonacci

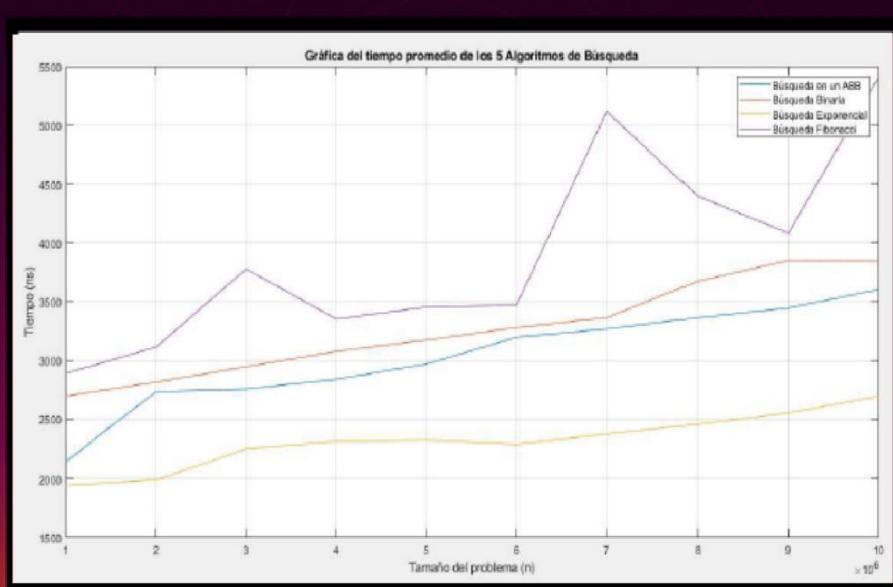


### -Grafica comparativa-

Con Búsqueda lineal

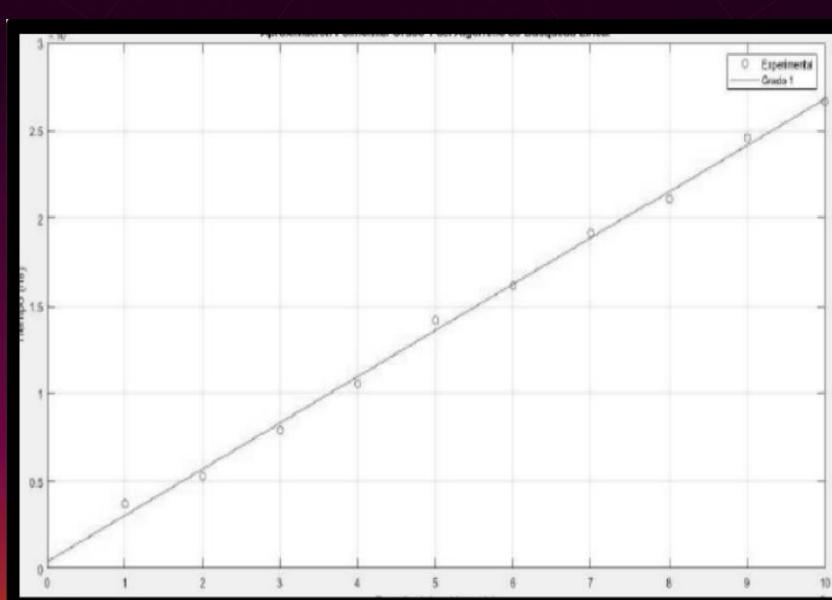


Sin Búsqueda lineal

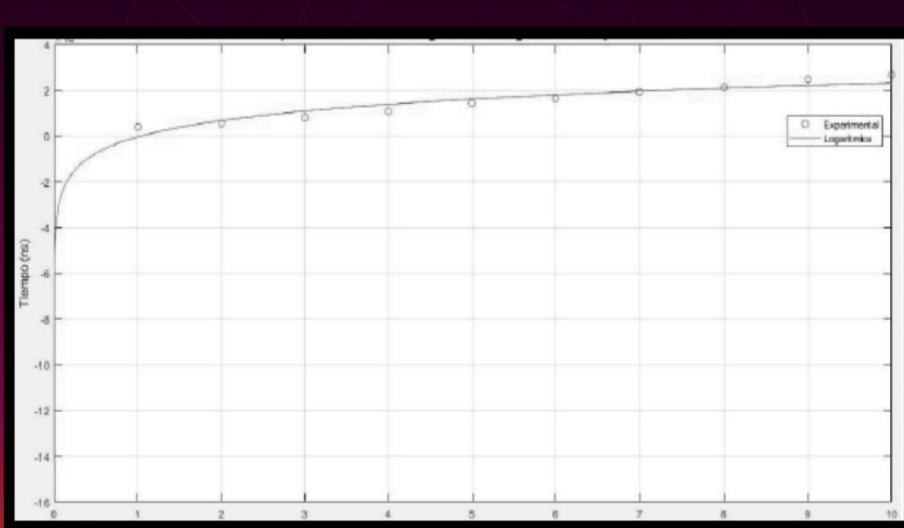


### -Graficas de aproximación-

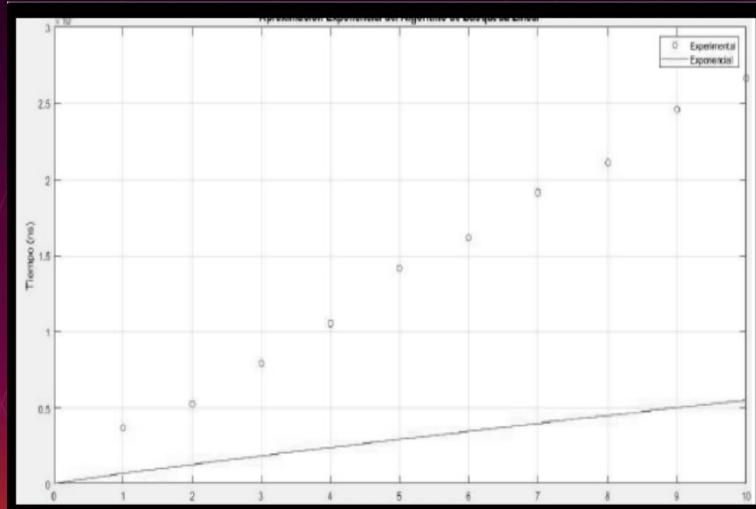
#### Búsqueda lineal (Polinomial)



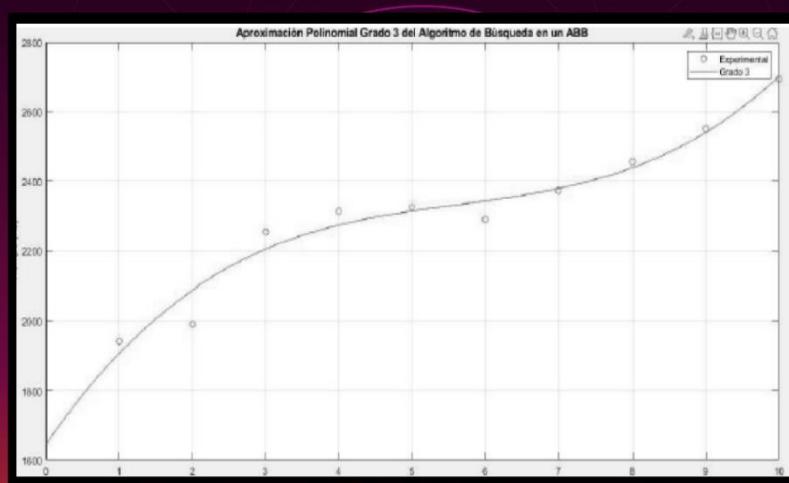
#### Búsqueda lineal (Logarítmica)



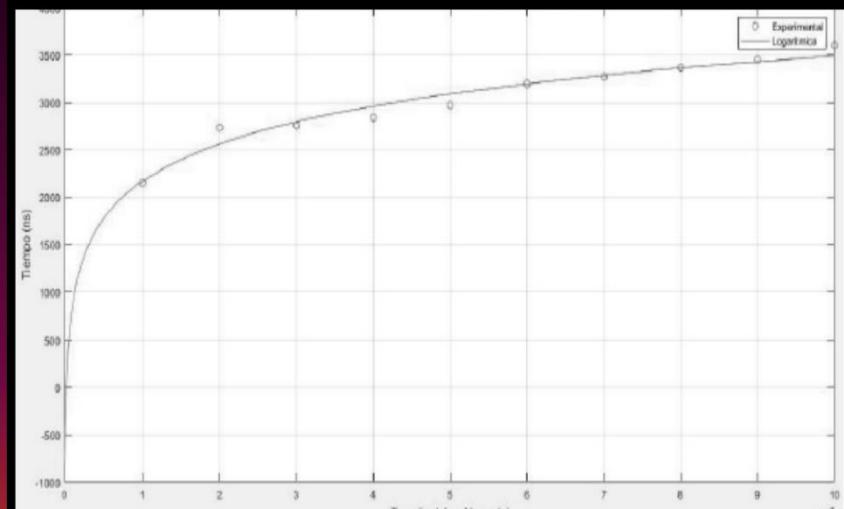
## Búsqueda lineal (Exponencial)



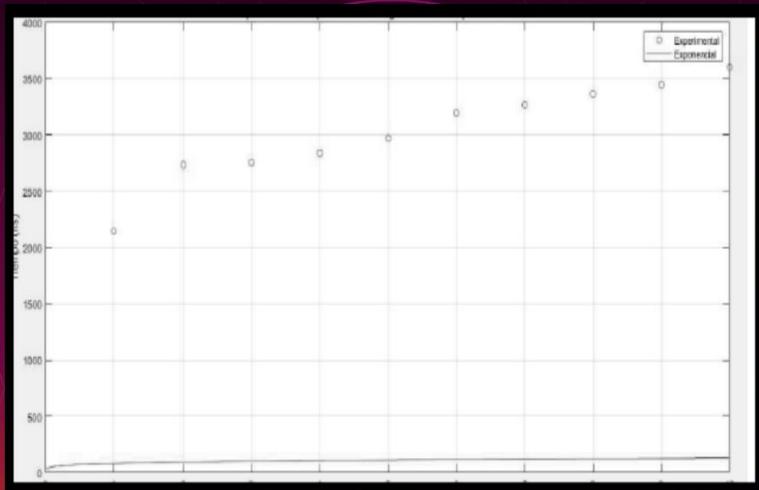
## Búsqueda ABB (Polinomial)



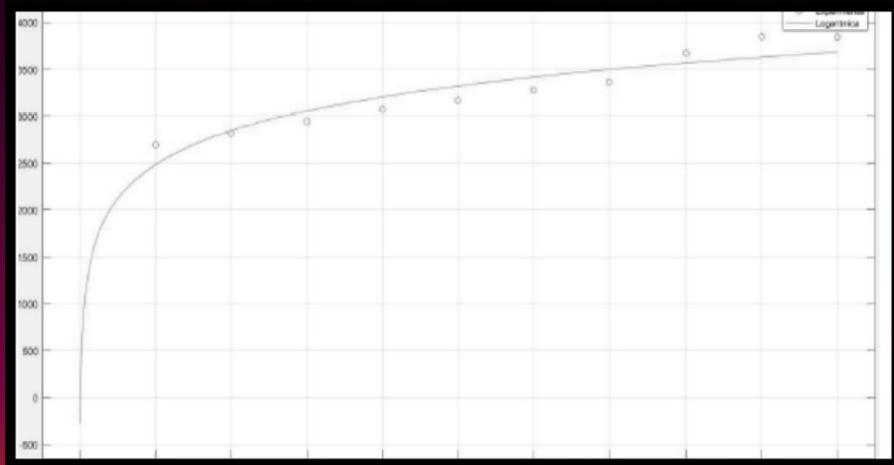
## Búsqueda ABB (Logarítmica)



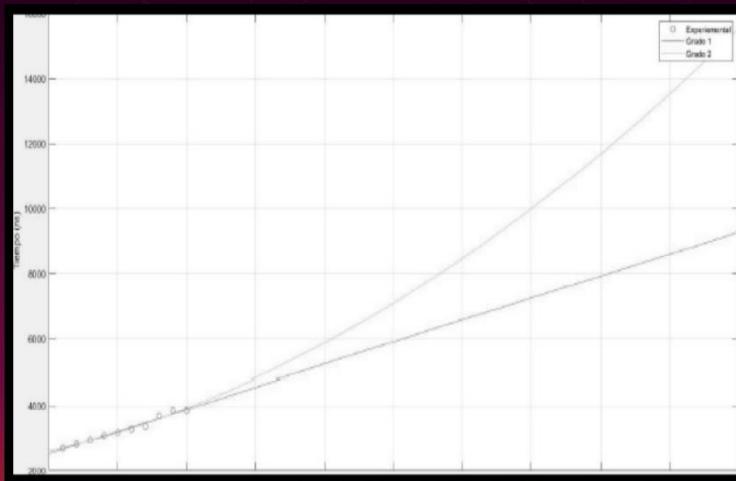
## Búsqueda ABB (Exponencial)



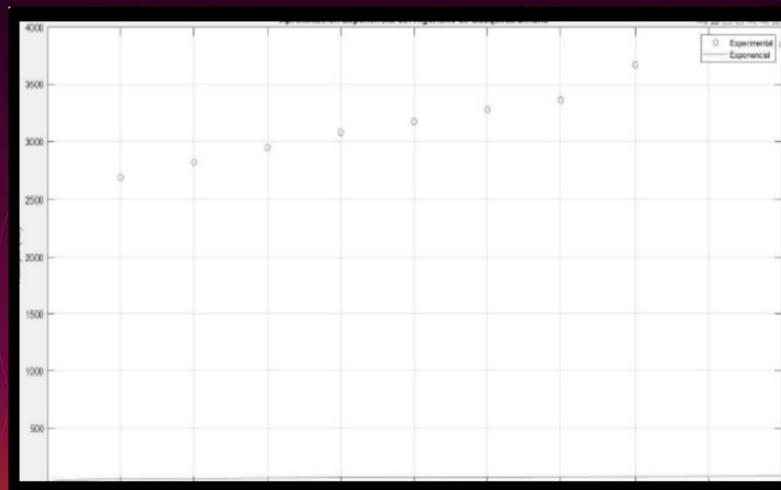
## Búsqueda Binaria (Polinomial)



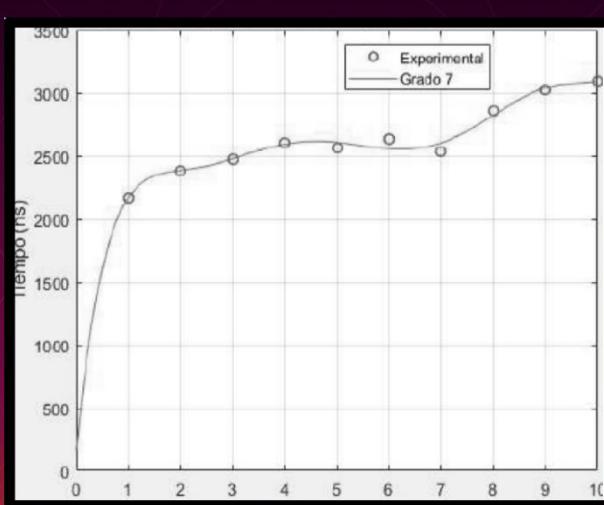
## Búsqueda Binaria (Logarítmica)



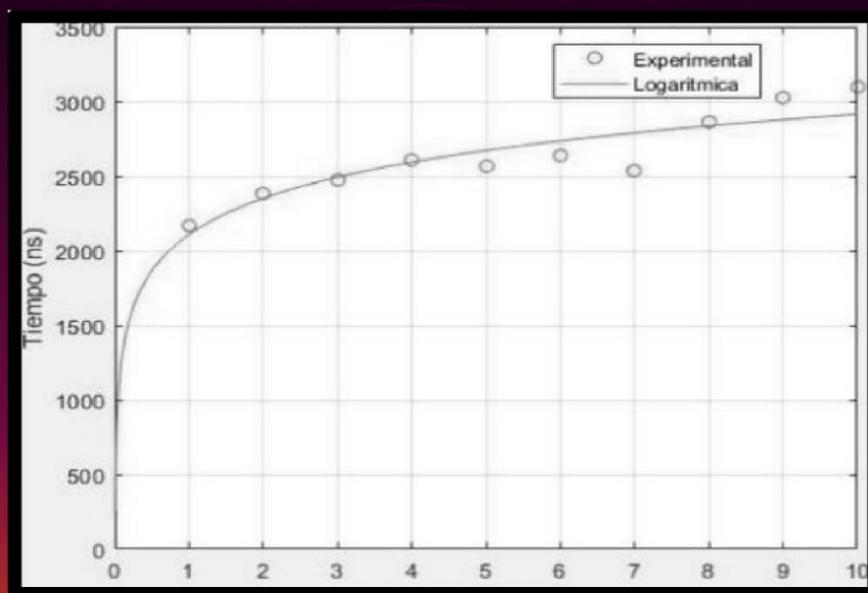
## Búsqueda Binaria (Exponencial)



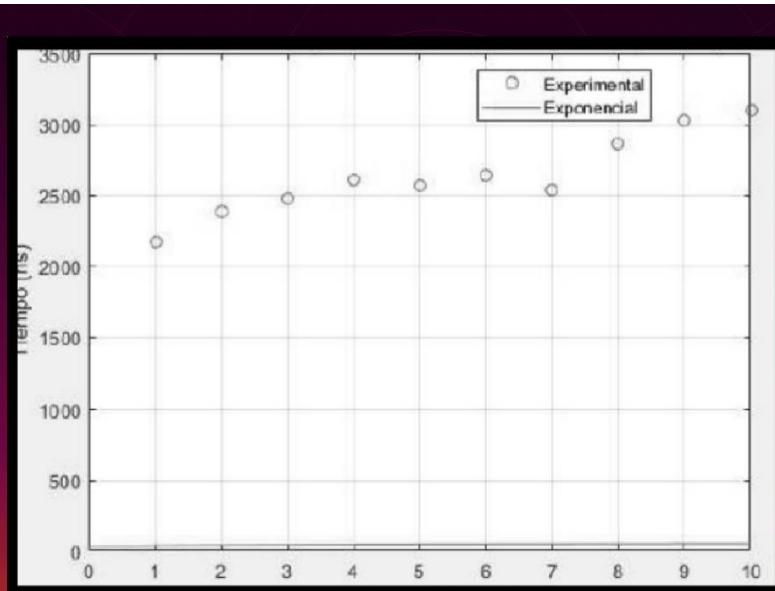
## Búsqueda Exponencial (Polinomial)



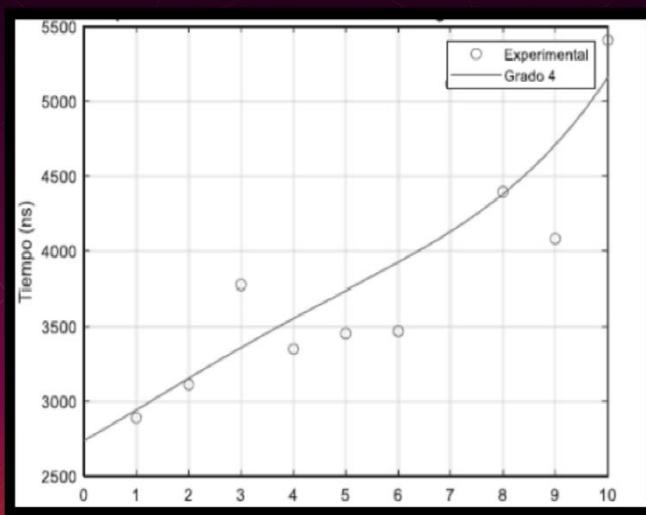
## Búsqueda Exponencial (Logarítmica)



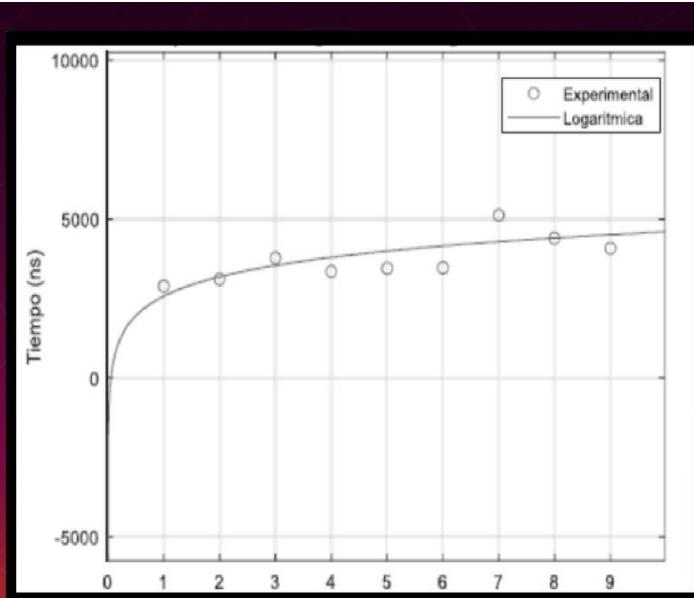
## Búsqueda Exponencial (Exponencial)



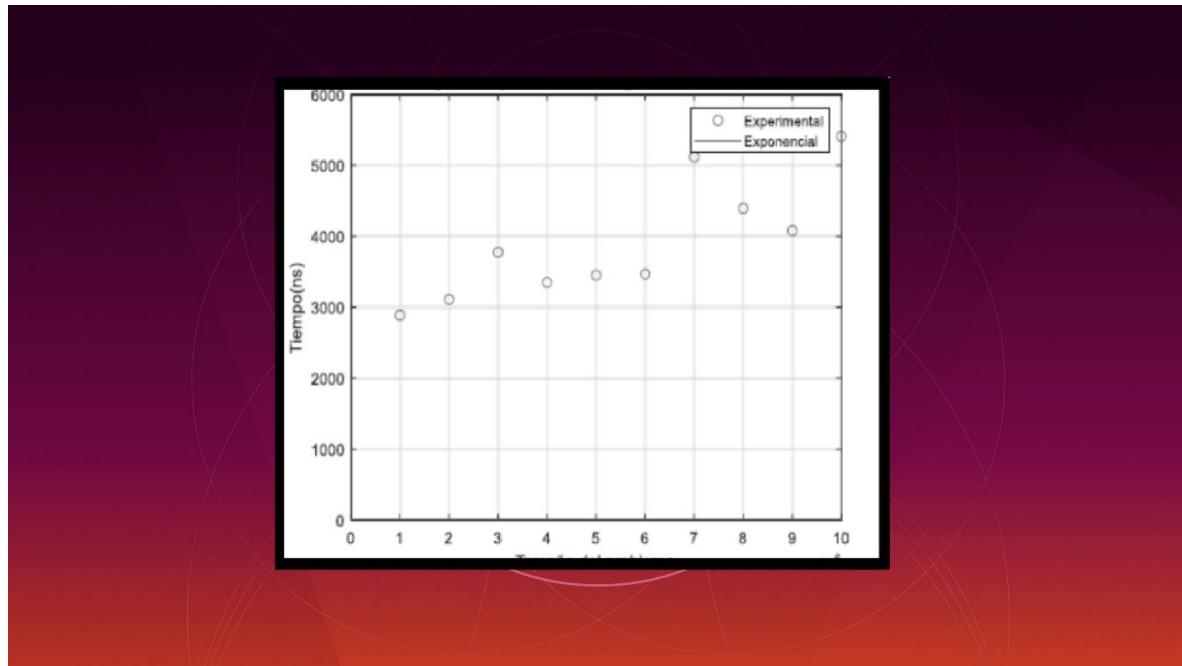
## Búsqueda Fibonacci (Polinomial)



## Búsqueda Fibonacci (Logarítmica)



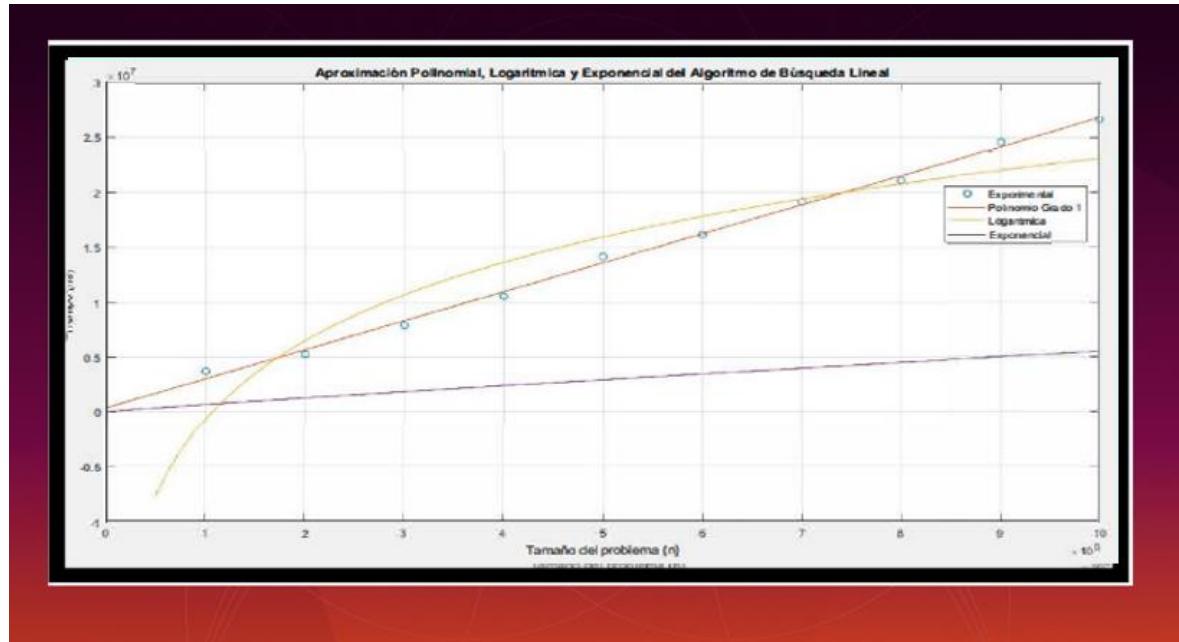
## Búsqueda Fibonacci (Exponencial)



## -Comparativa de las aproximaciones-

El cálculo de las aproximaciones no se acerca a la parte experimental de todas las búsquedas, ya que depende mucho de las pruebas y el momento de estar corriendo los scripts, hicimos lo que pudimos, pero realmente con el tiempo que se tuvo, nos tardamos más en comprender hilos y los algoritmos y pudimos correr solo una vez

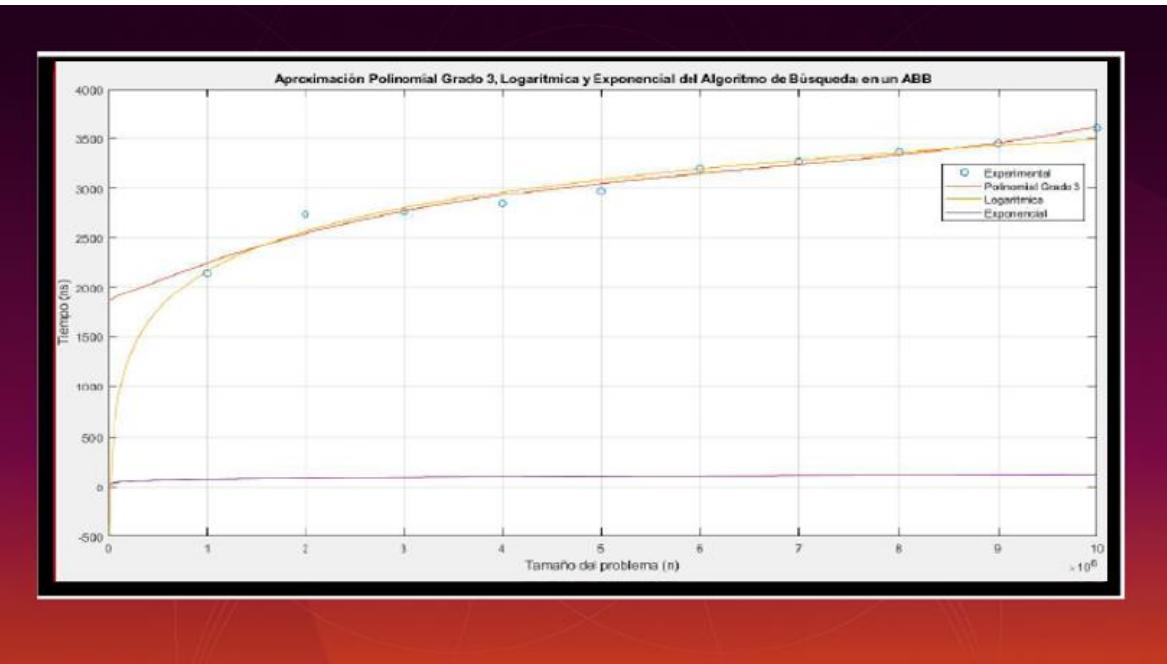
### Búsqueda Lineal



En la búsqueda lineal la función más aproximada fue la aproximación Polinomial de grado 1, ya que, se tiene que para la búsqueda lineal se tiene un crecimiento lineal por lo que crecerá cada vez con el tamaño del problema.

$$y = (2.6402)n + 3.9297 \times 10^5$$

## Búsqueda ABB



La búsqueda en un árbol binario de búsqueda la mejor aproximación que se tuvo fue la logarítmica, ya que, la exponencial ni siquiera se llegó a acercar y la lineal era muy plana, por lo que la logarítmica se aproximaba un poco mejor a lo experimental por la tendencia de las pruebas ya que poco a poco se iban estabilizando en un rango.

$$y = (399.5017)\log_2 n - 5.8030 \times 10^3$$

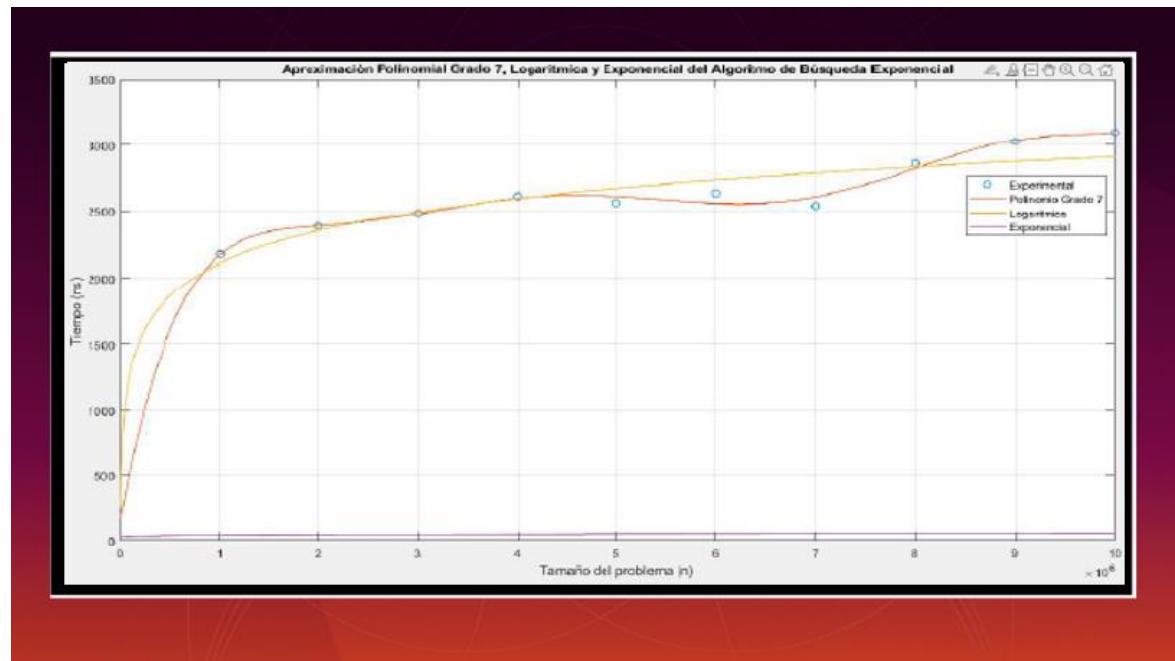
## Búsqueda Binaria



Se puede observar que las funciones que más se acercan a los puntos experimentales son de función logarítmico y Polinomial. En el caso de la Polinomial se acercaba el grado 2, ya que, pasaba en general por todos los puntos o la mayoría, por otro lado, la logarítmica sigue un poco la tendencia de lo experimental. La aproximación exponencial y grado 1 no se aproximaban a lo experimentado.

$$y = (360.7081)\log_2 n - 4.705 \times 10^3$$

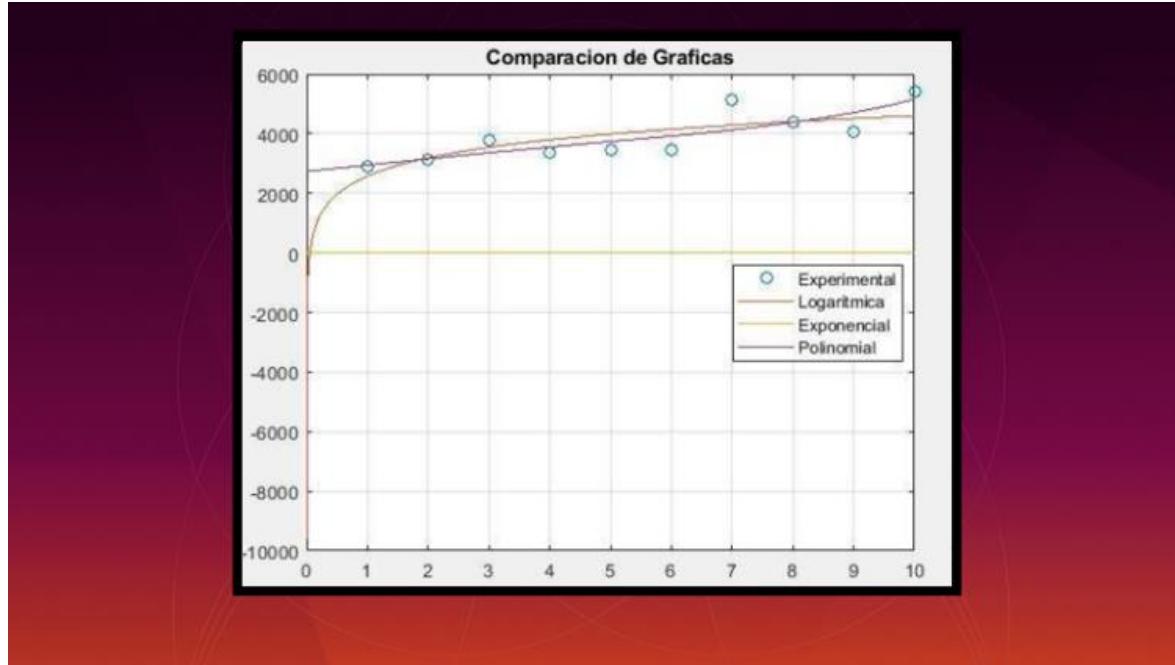
## Búsqueda Exponencial



Se puede observar que las funciones que más se acercan a los puntos experimentales es lo función polinomial y lo función logarítmico. Sin embargo, poro aproximarse más la función polinomial, tiene que ser de grado 7, yo que, los tiempos promedios no crecen tan rápido, la función que mejor se aproxima es lo función logarítmico.

$$\begin{aligned}y = & (157.3563)n^7 + (0.0043)n^6 - (3.3374 \times 10^{-9})n^5 + (1.3534 \times 10^{-15})n^4 \\& - (3.0026 \times 10^{-22})n^3 + (3.6623 \times 10^{-29})n^2 - (2.2979 \times 10^{-36})n \\& + 5.7879 \times 10^{-44}\end{aligned}$$

## Búsqueda Fibonacci

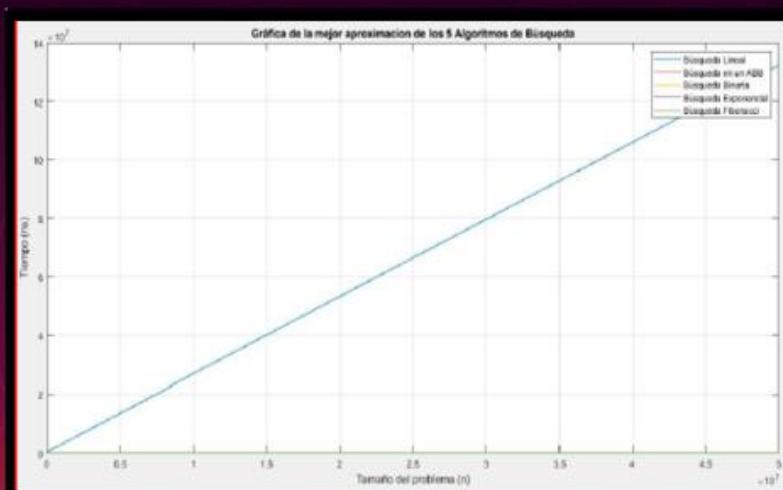


Se puede observar que las funciones que más se acercan a los puntos experimentales son de función logarítmica y Polinomial. En el caso de la logarítmica, esta fue la que más se acercó al comportamiento de la curva experimental.

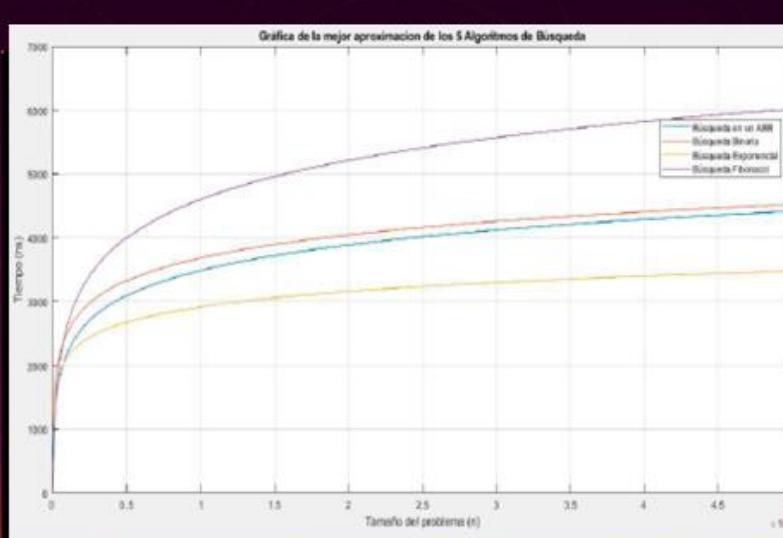
$$y = (880.28) \log n - 9.58 \times 10^3$$

## -Comparativa de todas las comparativas de los algoritmos-

Con Búsqueda Lineal

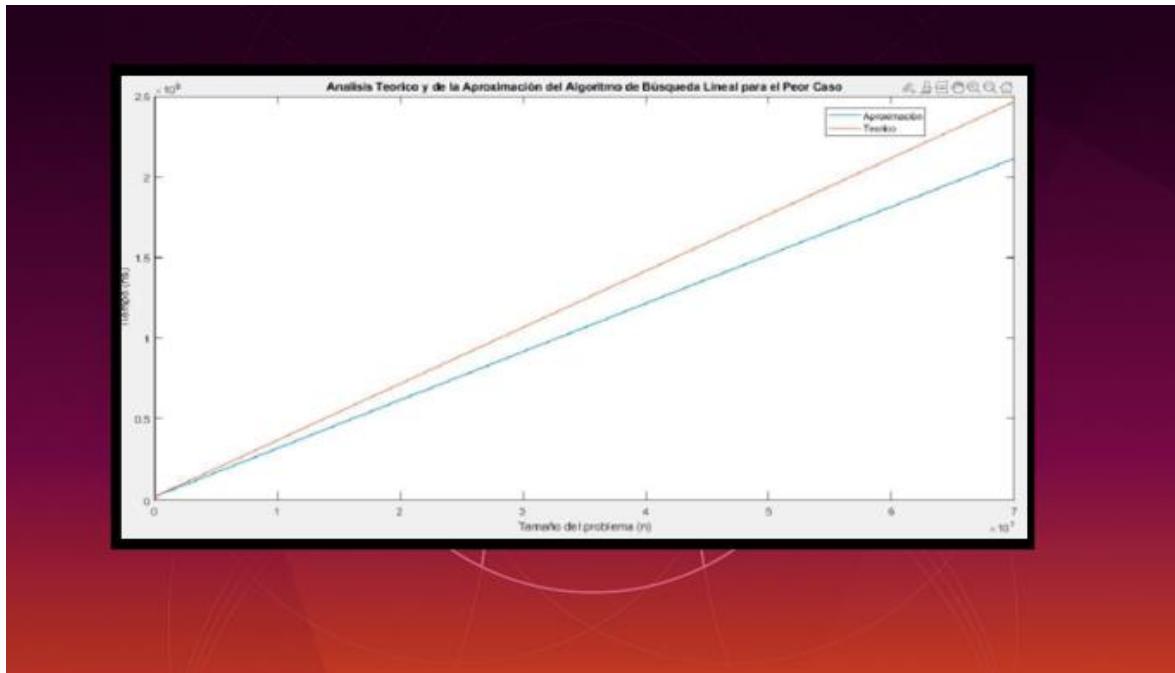


Sin Búsqueda Lineal



## -Constante Multiplicativa-

### Búsqueda Lineal

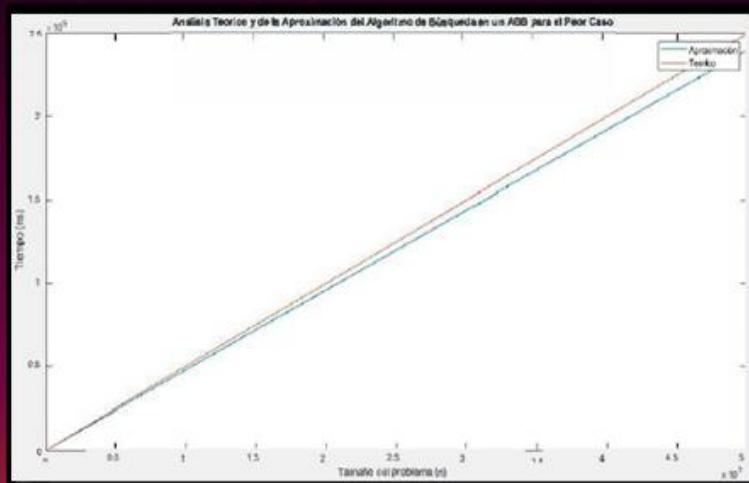


función con la constante

$$y = 3.5n$$

Poro lo búsqueda lineal se eligió la constante siguiendo como base el coeficiente del término lineal de lo aproximación obtenido, yo que, se vio que entre el rango de 3 a 4 se iba aproximando, por lo que se pensó irse con valores decimales para tener un mayor acercamiento o lo porte de lo aproximación del peor coso.

## Búsqueda ABB

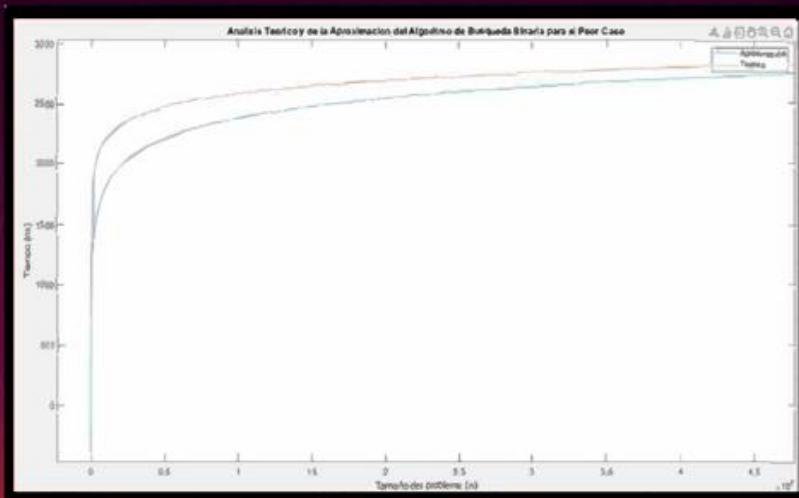


Función con la constante

$$y = 0.005n$$

Para la elección de la constante para tratar de aproximar la función teórica a la obtenida por medio de aproximaciones de Matlab se consiguió por prueba y error basándonos más en el coeficiente que tiene la aproximación de Matlab y con ellos irnos aproximando hasta que se logró ir muy pegada la función teórica para poder complementarla.

## Búsqueda Binaria

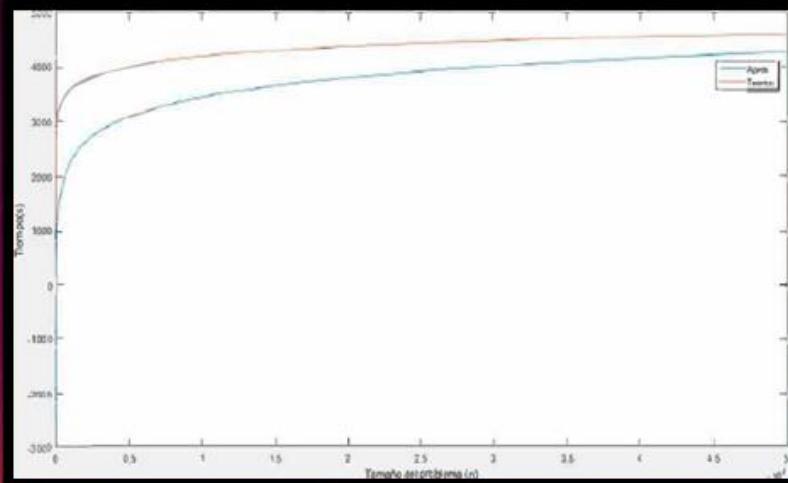


Función con la constante

$$y = 111 \log_2 n$$

Se eligió esta constante, a partir de prueba y error, primero empezamos dándole valores del 1 al SO, sin embargo, la función no crecía tanto y quedaba por debajo de la aproximación, subimos hasta 100 y un poco más donde ya se quedaban aproximadas.

## Búsqueda Exponencial

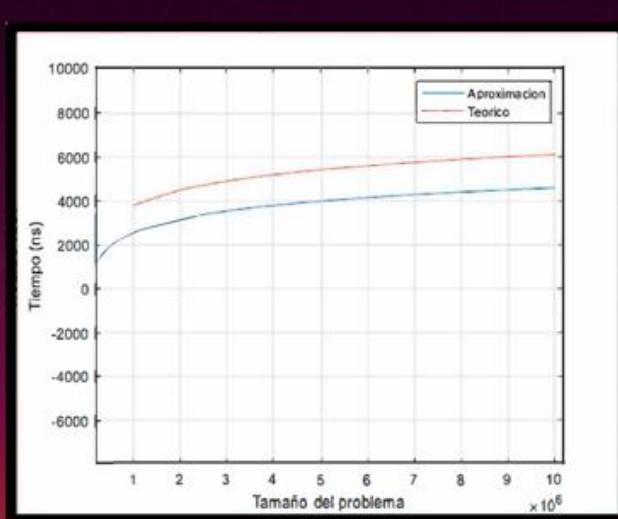


Función con la constante

$$y = 90(\log_2 n + \log_2 \frac{n}{2} + 1)$$

Se eligió esta constante, a partir de prueba y error. Al principio se probó utilizando una constante igual o 500, sin embargo, la función teoría creció demasiado rápido y quedaba muy por encima de la función obtenida a partir de las pruebas del peor caso. Posteriormente se eligió una constante igual a 50, y aunque está por debajo de la función obtenida, ya estaba más próxima. En algún momento se había optado por una constante igual a 73, ya que, estaba muy próxima a la otra función, pero al aumentar el rango a mayor a 50 millones, se observó que realmente no se aproximaba. Finalmente, se eligió una constante igual a 90, ya que en el rango de hasta 50 millones, se podía notar que ya las dos funciones se aproximaban mucho. Es importante mencionar que entre mayor sea el rango, mayor debería ser la constante para que las dos funciones se aproximen, pero por cuestiones de cómputo, ya no se puede comprobar.

## Búsqueda Fibonacci



Función con la constante

$$y = (700) \log n$$

Para la búsqueda Fibonacci se aproximamos la función teórica a la obtenida por aproximaciones de Matlab, haciendo prueba y error asignamos valores hasta que se aproxima lo mejor posible.



### -Evaluación de funciones en diferentes n-

Búsqueda lineal

**n=50,000,000**

CO = 3.5;

x1 = 50000000;

y1 = CO.\*(x1)

**Tiempo Real (ns) = 175000000**

**n=100,000,000**

CO = 3.5;

x2= 100000000;

y2 = CO.\*(x2)

**Tiempo Real (ns) = 350000000**

**n=500,000,000**

CO = 3.5;

x3= 500000000

y3 = CO.\*(x3)

**Tiempo Real (ns) = 1.7500e9**

**n=1,000,000,000**

CO = 3.5;

**x4= 1000000000**

y4 = CO.\*(x4)

**Tiempo Real (ns) = 3.5000e9**

**n=5,000,000,000**

CO = 3.5;

x5= 5000000000

y5 = CO.\*(x5)

**Tiempo Real (ns) = 1.7500e10**



Búsqueda en ABB

**n=50,000,000**

CO = 0.005;

x1 = 50000000;

y1 = CO.\*(x1)

**Tiempo Real (ns) = 250000**

**n=100,000,000**

CO = 0.005;

x2= 100000000;

y2 = CO.\*(x2)

**Tiempo Real (ns) = 500000**

**n=500,000,000**

CO = 0.005;

x3= 500000000;

y3 = CO.\*(x3)

**Tiempo Real (ns) = 2500000**

**n=1,000,000,000**

CO = 0.005;

x4= 1000000000;

y4 = CO.\*(x4)

**Tiempo Real (ns) = 5000000**

**n=5,000,000,000**

CO = 0.005;

x5= 5000000000;

y5 = CO.\*(x5)

**Tiempo Real (ns) = 25000000**



*Búsqueda binaria*

**n=50,000,000**

x = 50000000;

co = 111;

yT = co.\*(log2(x))

**Tiempo Real (ns)**

yT = 2.8389e+03

**n=100,000,000**

x=100000000;

co = 111;

yT = co.\*(log2(x))

**Tiempo Real (ns)**

yT = 2.9499e+03

**n=500,000,000**

x = 500,000,000;

co = 111;

yT = co.\*(log2(x))

**Tiempo Real (ns)**

yT = 3.2076e+03

**n=1,000,000,000**

x = 1000000000

co = 111;

yT = co.\*(log2(x))

**Tiempo Real (ns)**

yT = 3.318e+03

**n=5,000,000,000**

x = 5000000000;

co = 111;

yT = co.\*(log2(x))

**Tiempo Real (ns)**

yT = 3.5763e+03



### Búsqueda Exponencial

**n=50,000,000**

CO = 90;  
x1 = 50000000;

$$yT1 = CO.*(\log2(x1)+\log2(x1./2)+1)$$

**Tiempo Real (ns) = 4603.6**

**n=100,000,000**

CO = 90;

x2 = 100000000;

$$yT2 = CO.*(\log2(x2) +\log2(x2./2)+1)$$

**Tiempo Real (ns) = 4783.6**

**n=500,000,000**

CO = 90;

x3 = 500000000;

$$yT3 = CO.*(\log2(x3) +\log2(x3./2)+1)$$

**Tiempo Real (ns) = 5201.5**

**n=1,000,000,000**

CO = 90;

x4 = 1000000000;

$$yT4 = CO.*(\log2(x4) +\log2(x4./2)+1)$$

**Tiempo Real (ns) = 5381.5**

**n=5,000,000,000**

CO = 90;

x5 = 5000000000;

$$yT5 = CO.*(\log2(x5) +\log2(x5./2)+1)$$

**Tiempo Real (ns) = 5799.5**



### Búsqueda Fibonacci

**n=50,000,000**

CO = 80;  
x1 = 50000000;  
yT1 = CO.\*(log2(x1))  
**Tiempo Real (ns) = 5389.2**

**n=100,000,000**

CO = 80;  
x2 = 100000000;  
yT2 = CO.\*(log2(x2))  
**Tiempo Real (ns) = 6400**

**n=500,000,000**

CO = 80;  
x3 = 500000000;  
yT3 = CO.\*(log2(x3))  
**Tiempo Real (ns) = 6959.17**

**n=1,000,000,000**

CO = 80;  
x4 = 1000000000;  
yT4 = CO.\*(log2(x4))  
**Tiempo Real (ns) = 7200**

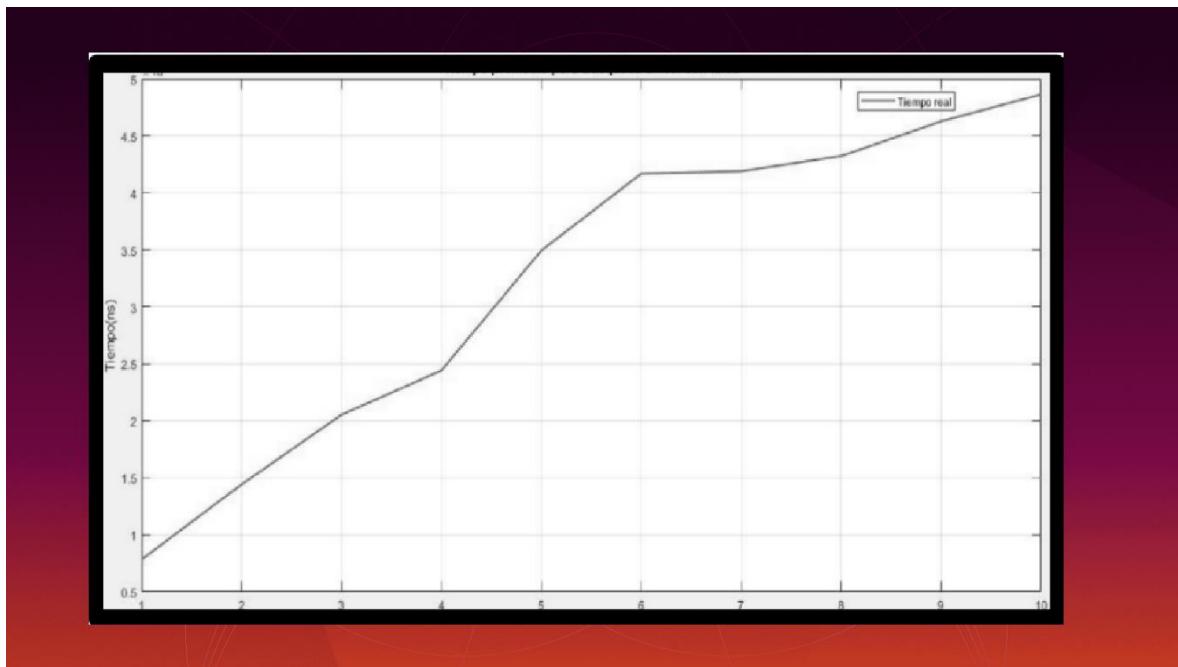
**n=5,000,000,000**

CO = 80;  
x5 = 5000000000;  
yT5 = CO.\*(log2(x5))  
**Tiempo Real (ns) = 7759.1**

### -Tablas comparativas con hilos-

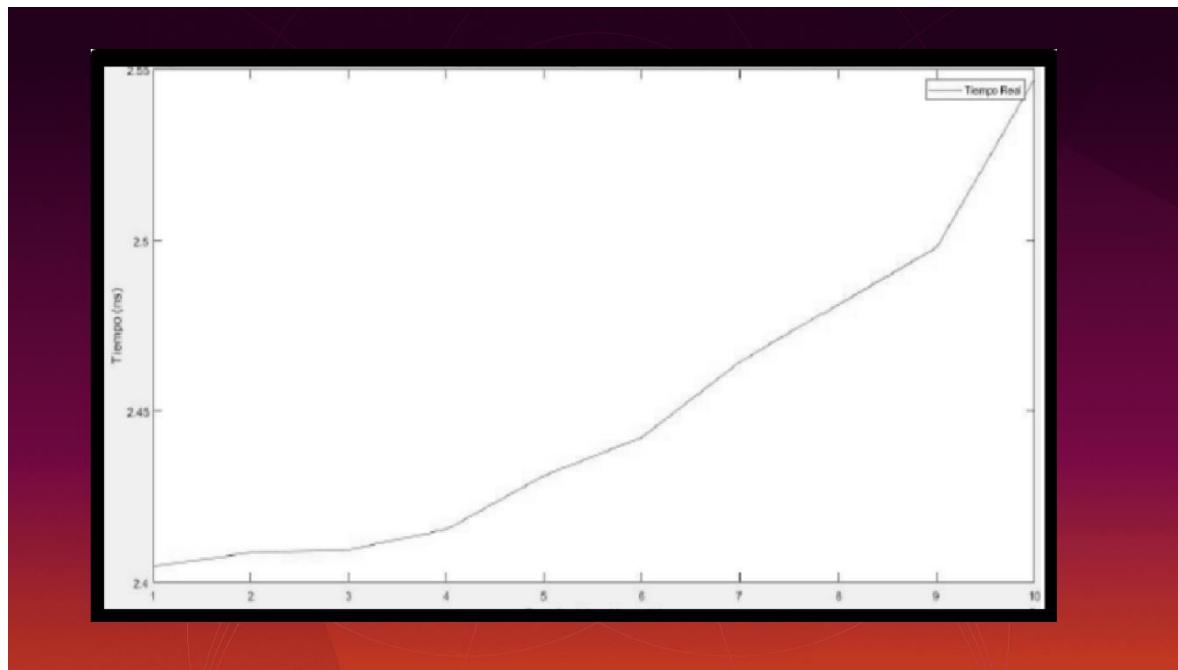
#### Búsqueda Lineal

| Búsqueda lineal |                            |                          |            |
|-----------------|----------------------------|--------------------------|------------|
| Tamaño de n     | Tiempo Real<br>(Sin hilos) | Tiempo real<br>Con hilos | Mejora (%) |
| 1,000,000       | 3,546,445.54               | 7,654,354.25             | 43.31      |
| 2,000,000       | 5,654,554.21               | 15,465,454.16            | 32.65      |
| 3,000,000       | 7,231,545.33               | 20,465,465.54            | 38.25      |
| 4,000,000       | 10,321,654.56              | 24,455,441.54            | 40.12      |
| 5,000,000       | 14,354,654.51              | 34,687,463.51            | 41.25      |
| 6,000,000       | 15,534,554.41              | 40,214,654.41            | 38.10      |
| 7,000,000       | 18,431,322.21              | 43,213,543.25            | 43.25      |
| 8,000,000       | 20,547,865.44              | 44,654,346.43            | 45.21      |
| 9,000,000       | 24,324,645.34              | 45,456,746.49            | 50.36      |
| 10,000,000      | 25,654,645.,654            | 48,465,466.54            | 52.21      |



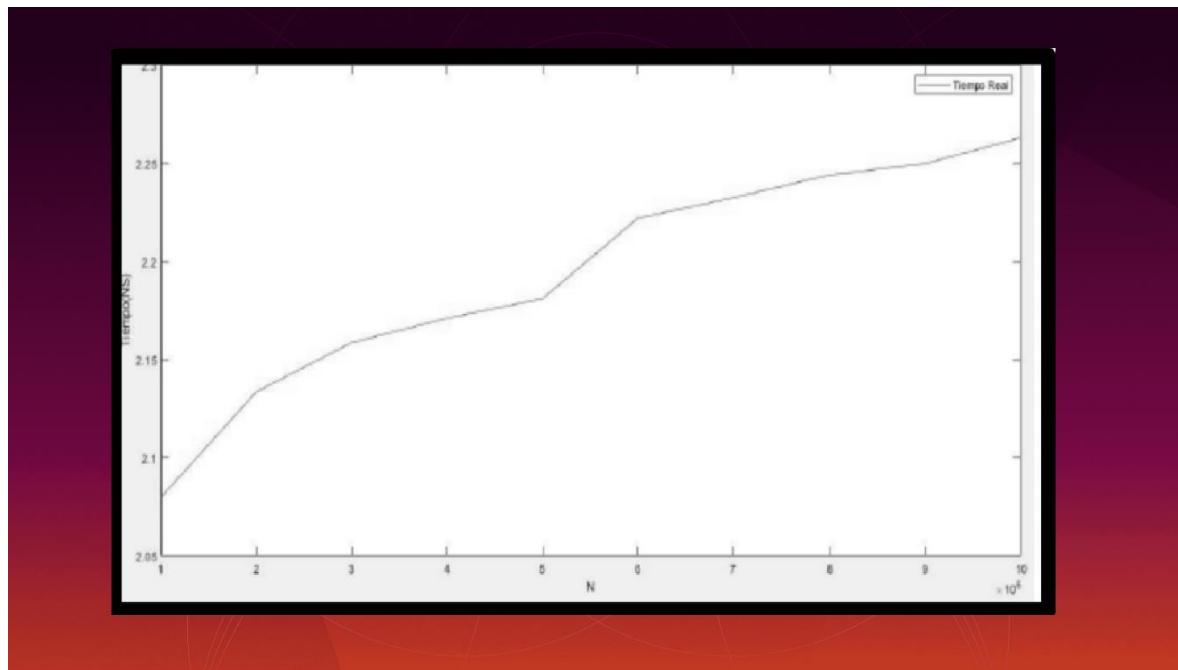
## Búsqueda ABB

| Búsqueda ABB      |                            |                          |            |
|-------------------|----------------------------|--------------------------|------------|
| Tamaño de n       | Tiempo Real<br>(Sin hilos) | Tiempo real<br>Con hilos | Mejora (%) |
| <b>1,000,000</b>  | 2,145.745                  | 240,457.058              | 0.89       |
| <b>2,000,000</b>  | 2,729.87                   | 240,862.37               | 1.13       |
| <b>3,000,000</b>  | 2,753.72                   | 240,945.816              | 1.14       |
| <b>4,000,000</b>  | 2,837.165                  | 241,541.862              | 1.17       |
| <b>5,000,000</b>  | 2,968.29                   | 243,115.425              | 1.25       |
| <b>6,000,000</b>  | 3,194.785                  | 244,235.992              | 1.32       |
| <b>7,000,000</b>  | 3,266.315                  | 246,436.234              | 1.35       |
| <b>8,000,000</b>  | 3,361.685                  | 248,122.215              | 1.40       |
| <b>9,000,000</b>  | 3,445.125                  | 249,787.373              | 1.42       |
| <b>10,000,000</b> | 3,600.11                   | 254,710.415              | 1.45       |



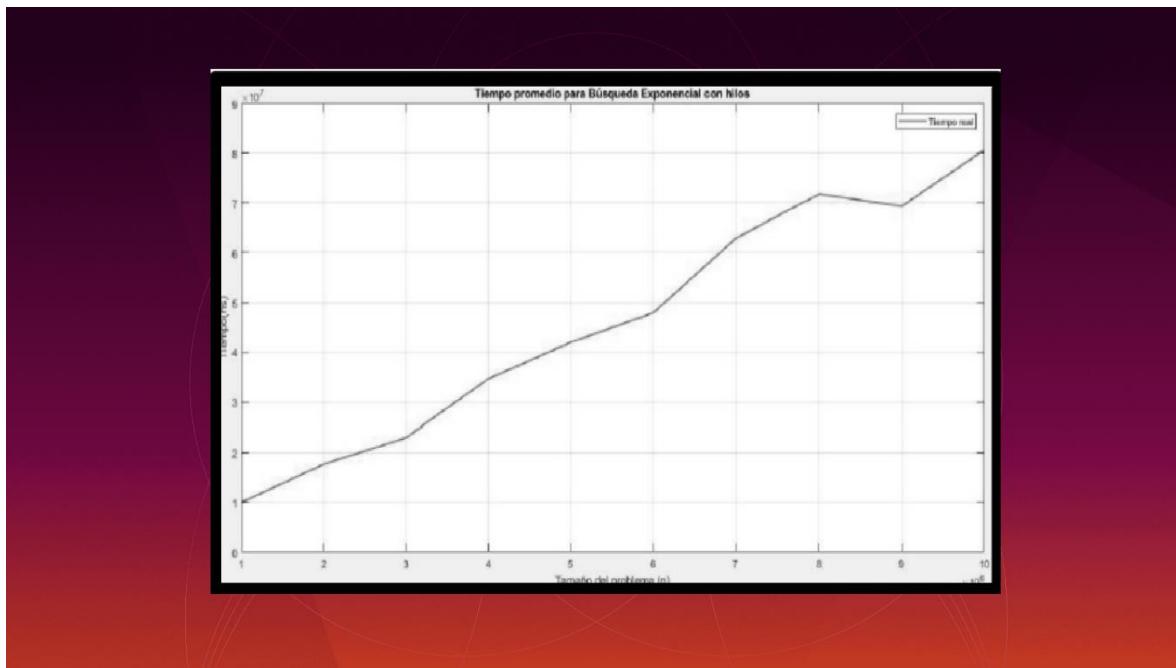
## Búsqueda Binaria

| Búsqueda Binaria  |                            |                          |            |
|-------------------|----------------------------|--------------------------|------------|
| Tamaño de n       | Tiempo Real<br>(Sin hilos) | Tiempo real<br>Con hilos | Mejora (%) |
| <b>1,000,000</b>  | 2,181.514                  | 207,974.925              | 0.104      |
| <b>2,000,000</b>  | 2,300.730                  | 213,350.21               | 0.106      |
| <b>3,000,000</b>  | 2,396.085                  | 215,857.69               | 0.111      |
| <b>4,000,000</b>  | 2,598.744                  | 217,101.245              | 0.118      |
| <b>5,000,000</b>  | 2,610.654                  | 218,g101.245             | 0.119      |
| <b>6,000,000</b>  | 2,622.584                  | 222,202.34               | 0.120      |
| <b>7,000,000</b>  | 2,658.330                  | 223,235.09               | 0.126      |
| <b>8,000,000</b>  | 2,908.680                  | 224,396.06               | 0.130      |
| <b>9,000,000</b>  | 3,135.180                  | 224,987.915              | 0.132      |
| <b>10,000,000</b> | 3,325.920                  | 226,301.78               | 0.134      |



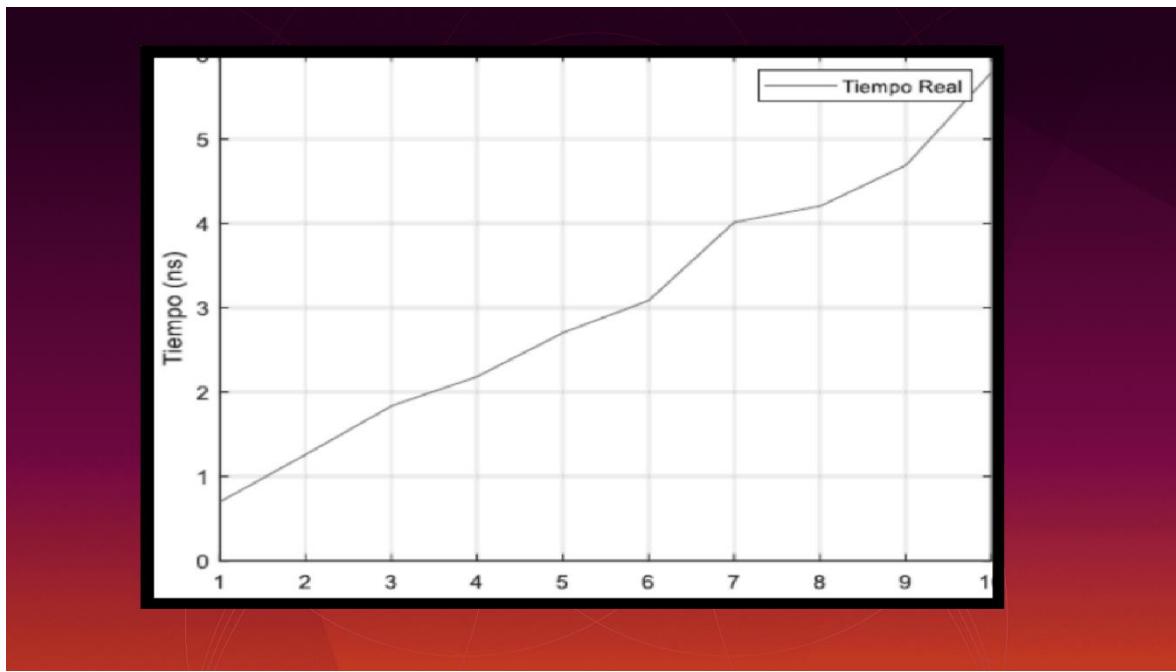
## Búsqueda Exponencial

| Búsqueda Exponencial |                            |                          |            |
|----------------------|----------------------------|--------------------------|------------|
| Tamaño de n          | Tiempo Real<br>(Sin hilos) | Tiempo real<br>Con hilos | Mejora (%) |
| <b>1,000,000</b>     | 2,169.609                  | 1,984,326.36             | 0.109      |
| <b>2,000,000</b>     | 2384.186                   | 17,652,988.4             | 0.014      |
| <b>3,000,000</b>     | 2,479.553                  | 22,977,149.5             | 0.011      |
| <b>4,000,000</b>     | 2,610.683                  | 34,635,853.8             | 0.008      |
| <b>5,000,000</b>     | 2,563.000                  | 41,953,326.9             | 0.006      |
| <b>6,000,000</b>     | 2,634.525                  | 47,915,995.1             | 0.005      |
| <b>7,000,000</b>     | 2,539.158                  | 62,853,944.3             | 0.004      |
| <b>8,000,000</b>     | 2,861.023                  | 71,788,680.6             | 0.004      |
| <b>9,000,000</b>     | 3,027.916                  | 69,3S8,505.5             | 0.005      |
| <b>10,000,000</b>    | 3,099.442                  | 80,645,668.5             | 0.003      |



## Búsqueda Fibonacci

| Búsqueda Fibonacci |                            |                          |            |
|--------------------|----------------------------|--------------------------|------------|
| Tamaño de n        | Tiempo Real<br>(Sin hilos) | Tiempo real<br>Con hilos | Mejora (%) |
| <b>1,000,000</b>   | 2,888.82                   | 6,987,204.2              | 0.041      |
| <b>2,000,000</b>   | 3,111.35                   | 12,592,266.6             | 0.024      |
| <b>3,000,000</b>   | 3,775.2                    | 18,358,270.3             | 0.020      |
| <b>4,000,000</b>   | 3,349.8                    | 21,847,069.7             | 0.015      |
| <b>5,000,000</b>   | 3,453.1                    | 27,036,419.8             | 0.012      |
| <b>6,000,000</b>   | 3,469.5                    | 30,841,698.5             | 0.011      |
| <b>7,000,000</b>   | 5,118.6                    | 40,138,096.2             | 0.012      |
| <b>8,000,000</b>   | 4,394.8                    | 42,064,031.3             | 0.010      |
| <b>9,000,000</b>   | 4,080.9                    | 46,875,268.4             | 0.008      |
| <b>10,000,000</b>  | 5,408.1                    | 57,912,767.7             | 0.009      |





## -Cuestionario-

**1. ¿Cuál de los 5 algoritmos es más fácil de implementar?**

Búsqueda Lineal

**2. ¿Cuál de los 5 algoritmos es el más difícil de implementar?**

Búsqueda en un Árbol Binario de Búsqueda y Búsqueda Fibonacci

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

Búsqueda en un Árbol Binario de Búsqueda

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

Ninguno, en todos se empeoraba las versiones con hilos porque cada vez que dividíamos el arreglo original en subarreglos para que cada hilo buscara en esos subarreglos se producían peores casos básicamente.

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

Ninguno presentó alguna ventaja, tenemos 2 opciones:

- Es por la complejidad de estos, dando casos bastante dispares desde el inicio, pero más en cuanto más crecía el número
- Porque al dividir el arreglo en 2, uno de ellos SIEMPRE daría un peor caso, en ese caso nos pareció que o bien eran lo mismo o bien era peor ya que podía encontrarlo en el inicio y nosotros al dividirlo agarrábamos una mitad completa y la mitad de la otra, dándonos tiempos horribles

**6. ¿Cuál algoritmo tiene menor complejidad temporal?**

Búsqueda de Fibonacci

**7. ¿Cuál algoritmo tiene mayor complejidad temporal?**

Búsqueda Lineal

**8. ¿El comportamiento experimental de los algoritmos era el esperado? ¿Por qué?**

Sin duda alguna, tomamos en cuenta como se dividiría el arreglo con base a las búsquedas y era obvio que el lineal sería el peor al no ser dividido y agarrarlo completo, sin embargo aun teníamos dudas si Fibonacci o Exponencial serán los mejores, ya que agarran trozos grandes del arreglo para buscar lo cual facilita encontrar el número más rápido, pero al final ganó Fibonacci para nuestra sorpresa, nosotros pensábamos que lo haría exponencial



**9. ¿Sus resultados experimentales difieren mucho de los análisis teóricos que realizo? ¿A qué se debe?**

Si, ya que el teórico tomamos matemáticas muy perfectas, lo cual nos da casos bastante adecuados, pero con la experimental se hacen cosas muy variantes, además, a veces la computadora podía alejarse un poco o cosas que afectaran directamente

**10. ¿En la versión con hilos, usar e hilos dividió el tiempo en c? ¿lo hizo c veces más rápido?**

No, al contrario, hizo peores los tiempos

**11. ¿Cuál es el % de mejora que tiene cada uno de los algoritmos en su variante con hilos? ¿Es lo que esperabas? ¿Por qué?**

El mejor fue el rango de 40% – 50% en el lineal de ahí en fuera fueron 0 – 1 % en Binaria y exponencial y los demás eran bajísimos con 0.004%.

Según nuestro pensar, usábamos hilos para mejorar los tiempos, sin embargo, luego de argumentar y debatir mucho (Por que pensábamos que habíamos hecho mal la práctica) nos dimos cuenta de que era algo muy probable que empeorara los tiempos al dividir arreglos, a que SIEMPRE uno daría peor caso

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

Si, fue en una laptop Lenovo ideapad 5, con un procesador Intel core i5 de 11va generación, 16 GB de memoria ram a 3200 MHz.

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

Medianamente, mínimo el peor nos parece que sí, ya que tomaremos en cuenta lo peor de lo peor

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

Bastante, realmente algunos no los entendimos del todo y tuvimos que revisar otras fuentes para realizar 1 trabajo, pero por cuenta propia no logramos hacer todos ya que nos confundíamos mucho

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

Manejar muy bien scripts

Sabes analizar los algoritmos, tener un buen desempeño para realizar las operaciones

Saber analizar el por qué lo hilos funcionarían o no



## -Referencias-

1. GeeksforGeeks. (2022a, marzo 14). *Indexed Sequential Search*.  
<https://www.geeksforgeeks.org/indexed-sequential-search/?ref=gcse>
2. GeeksforGeeks. (2022b, agosto 26). *Binary Search Tree / Set 1 (Search and Insertion)*. <https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/?ref=gcse>
3. GeeksforGeeks. (2022c, septiembre 23). *Linear Search Algorithm*.  
<https://www.geeksforgeeks.org/linear-search/>
4. GeeksforGeeks. (2022d, octubre 13). *Binary Search*.  
<https://www.geeksforgeeks.org/binary-search/?ref=gcse>
5. GeeksforGeeks. (2022e, octubre 26). *Fibonacci Search*.  
<https://www.geeksforgeeks.org/fibonacci-search/?ref=gcse>