

**UNIVERSIDAD TECNOLÓGICA DE SANTIAGO**  
**SISTEMA CORPORATIVO**  
**FACULTAD DE INGENIERÍA Y ARQUITECTURA**  
**CARRERA DE SISTEMAS COMPUTACIONALES**



**ALGORITMOS PARALELOS**

**GRUPO 3**

**Presentado a:**

**Iván Mendoza**

**Presentado por:**

**Sugeiri Torres                      1-16-0736**

**Dinnibel Azcona                      1-16-0788**

**Claudia Báez                      2-16-1116**

**Santiago de los Caballeros,**  
**República Dominicana**

## INDICE

## PAG.

1. Introducción.....	4
2. Descripción del Proyecto.....	5
3. Objetivos.....	5
3. 1. Objetivo General.....	5
3. 2. Objetivos Específicos.....	5
4. Definición de Algoritmos Paralelos.....	5
5. Etapas de los Algoritmos paralelos.....	5
5. 1. Partición.....	5
5. 2. Comunicación.....	6
5. 3. Agrupamiento.....	7
5. 4. Asignación.....	7
6. Técnicas Algorítmicas Paralelas.....	8
6. 1. Divide y Conquistarás.....	8
6. 2. Aleatorización.....	8
6. 3. Técnicas de Puntero Paralelo.....	8
7. Modelos de Algoritmos Paralelos.....	9
7. 1. Paralelismo de datos.....	9
7. 2. Grafo de tareas.....	9
7. 3. Conjunto de trabajadores (work pool).....	9
7. 4. Maestro-esclavo (master-slave).....	9
7. 5. Productor-consumidor (pipeline producer-consumer).....	10
8. Algoritmos de Búsquedas y Ordenamiento (Adjuntar PSeudocódigo y código de cada uno).....	10
8. 1. Búsqueda Secuencial.....	10
8. 2. Búsqueda Binaria.....	11
8. 3. Algoritmo de Ordenamiento de la Burbuja.....	13
8. 4. Quick Sort.....	13

8. 5. Método de Inserción.....	15
9. Programa desarrollado.....	16
9. 1. Explicación de su funcionamiento.....	16
9. 2. Fotos de la aplicación.....	16
9. 3. Link de GitHub y Ejecutable de la aplicación.....	20
9. 4. Resultados (Tiempo en terminar los ordenamientos y búsqueda de cada algoritmo).....	20
10. ¿Qué tanta memoria se consumió este proceso?.....	21
11. ¿Cuál fue el algoritmo que realizó la búsqueda y el ordenamiento más rápido? Explique.....	21
12. Conclusión.....	22
13. Bibliografías.....	23

## 1. Introducción

El presente proyecto tiene como finalidad presentar los diferentes algoritmos de búsqueda y ordenamiento, explicar que son y como se implementa. También explicaremos acerca de que es algoritmo paralelo, sus etapas, técnicas y modelos.

Presentaremos de forma practica varios métodos como el quick sort y el método burbuja. Analizando la duración de los mismos hasta lograr determinar cual es el mas eficiente.

## 2. Descripción del Proyecto

El proyecto se basa en una aplicación que permite ordenar un arreglo y comparar una serie de algoritmos de búsquedas a fin de estudiar su eficiencia y determinar, en base a los tiempos estudiados, cual es el mas rápido.

## 3. Objetivos

### 3. 1.      **Objetivo General**

Analizar los diferentes algoritmos de búsqueda y ordenamiento para determinar su eficiencia.

### 3. 2.      **Objetivos Específicos**

- Implementar varios algoritmos de búsqueda y ordenamiento
- Estudiar el comportamiento de los diferentes algoritmos con respecto a un mismo vector
- Determinar, mediante análisis, cual de los algoritmos es el mas eficiente

## 4. Definición de Algoritmos Paralelos

Es un algoritmo que puede ejecutar multiples instrucciones de forma simultanea en diferentes dispositivos de procesamiento, luego combinar las salidas individuales para producir el resultado final.

## 5. Etapas de los Algoritmos paralelos

### 5. 1.      **Partición**

El cómputo y los datos sobre los cuales se opera se descomponen en tareas. Se ignoran aspectos como el número de procesadores de la máquina a usar y se concentra la atención en explotar oportunidades de paralelismo.

En esta etapa se buscan oportunidades de paralelismo y se trata de subdividir el problema lo más finamente posible, es decir; que la granularidad sea fina.

Un grano es una medida del trabajo Computacional a realizar.

Al particionar se deben tener en cuenta los siguientes aspectos:

- El número de tareas debe ser por lo menos un orden de magnitud superior al número de procesadores disponibles para tener flexibilidad en las etapas siguientes.
- Hay que evitar cálculos y almacenamientos redundantes; de lo contrario el algoritmo puede ser no extensible a problemas más grandes.
- Hay que tratar de que las tareas sean de tamaños equivalentes ya que facilita el balanceo de la carga de los procesadores.
- Considere alternativas de paralelismo en esta etapa ya que pueden flexibilizar etapas subsecuentes.
- El número de tareas debe ser proporcional al tamaño del problema. Así, se podrá resolver problemas más grandes cuando se tenga más procesadores. Es decir, que el algoritmo sea escalable.

## 5. 2. **Comunicación**

En esta etapa se determina la comunicación requerida para coordinar las tareas y se definen estructuras y algoritmos de comunicación.

La comunicación requerida por un algoritmo puede ser definida en dos fases.

- Primero se definen los canales que conectan las tareas que requieren datos con las que los poseen.
- Segundo se especifica la información o mensajes que deben ser enviado y recibidos en estos canales.

En la etapa de comunicación hay que tener en cuenta los siguientes aspectos:

- Todas las tareas deben efectuar aproximadamente el mismo número de operaciones de comunicación. Si esto no se da, es muy probable que el algoritmo no sea extensible a problemas mayores ya que habrán cuellos de botella.
- La comunicación entre tareas debe ser tan pequeña como sea posible.
- Las operaciones de comunicación deben poder proceder concurrentemente.
- Los cálculos de diferentes tareas deben poder proceder concurrentemente.

### 5. 3. **Agrupamiento**

El resultado de las dos etapas anteriores es evaluado en términos de eficiencia y costos de implementación. De ser necesario, se agrupan tareas pequeñas en tareas más grandes.

Esta etapa va de lo abstracto a lo concreto y se revisa el algoritmo obtenido tratando de considerar si es útil agrupar tareas y si vale la pena replicar datos y/o cálculos de acuerdo a la plataforma seleccionada.

Mediante la agrupación de tareas se puede reducir la cantidad de datos a enviar y así reducir el número de mensajes a transmitir y por ende el costo de comunicación.

En la etapa de agrupación se debe tener en cuenta los siguientes aspectos:

- Chequear si la agrupación redujo los costos de comunicación.
- Si se han replicado cálculos y/o datos, se debe verificar que los beneficios son superiores a los costos.
- Se debe verificar que las tareas resultantes tengan costos de cómputo y comunicación similares.
- Hay que revisar si el número de tareas es extensible con el tamaño del problema.
- Si el agrupamiento ha reducido las oportunidades de ejecución concurrente, se debe verificar que aun hay suficiente concurrencia y posiblemente considerar diseños alternativos.
- Analizar si es posible reducir aun más el número de tareas sin introducir desbalances de cargas o reducir la extensibilidad.

### 5. 4. **Asignación**

Cada tarea es asignada a un procesador tratando de maximizar la utilización de los procesadores y de reducir el costo de comunicación.

La asignación puede ser:

- Estática: una tarea es asignada a un procesador desde su inicio hasta su fin
- Dinámica: una tarea puede ser migrada durante su ejecución. Esto puede agregar un costo adicional

Aquí se trata de establecer el mayor número posible de tareas con la intención de explorar al máximo las oportunidades de paralelismo.

En esta etapa se determina en que procesador se ejecutará cada tarea

## 6. Técnicas Algorítmicas Paralelas

### 6. 1. **Divide y Conquistarás**

El algoritmo divide un problema a resolver en subproblemas que son más fáciles de resolver que el problema original, resuelve subproblemas, combina soluciones de subproblemas y encuentra soluciones a las causas del problema.

El modelo divide y vencerás mejora la modularidad del programa y, a menudo, da como resultado algoritmos simples y eficientes. Por lo tanto, ha demostrado ser una herramienta poderosa para los diseñadores de algoritmos secuenciales.

### 6. 2. **Aleatorización**

Los números aleatorios se utilizan en algoritmos paralelos para permitir que el procesador tome decisiones locales. Esto tiene una alta probabilidad de que la decisión general sea adecuada. Aquí analizamos tres usos del caso.

### 6. 3. **Técnicas de Puntero Paralelo**

#### 6. 3. 1. 1. **Recorrido de Euler**

es un camino a través de un gráfico en el que cada arco se recorre solo una vez. En los gráficos no dirigidos, cada borde suele sustituirse por dos bordes opuestos. El camino del árbol no se dirige alrededor del árbol, moviendo cada borde dos veces, una hacia abajo y otra hacia arriba. Al mantener una estructura conectada que representa la ruta de Euler a través del árbol, puede calcular muchas funciones en el árbol, como el tamaño de cada subárbol. Esta técnica utiliza trabajo lineal y profundidad paralela y es independiente de la profundidad del árbol. Las instrucciones de Euler se utilizan a menudo para reemplazar las instrucciones de árbol estándar, como las instrucciones de profundidad.

#### 6. 3. 2. **Contracción del grafo**

el tamaño de un gráfico conservando parte de la estructura original. Por lo general, después de realizar una operación de reducción en un gráfico, el problema se resuelve de forma recursiva en el gráfico reducido. La solución al problema del gráfico de contrato se utiliza para formar la solución final. Por ejemplo, una forma de dividir un gráfico en componentes conectados es combinar primero varios vértices en vértices adyacentes para contraer el gráfico y luego encontrar los componentes relevantes del gráfico. Contrae y



eventualmente cancela la operación de contracción. Se pueden solucionar muchos problemas encogiendo el eje. En ese caso, esta técnica se conoce como contracción del eje.

### 6. 3. 3. **Descomposición del oído**

en dividir los bordes del gráfico en un conjunto ordenado de caminos. El primer segmento es el bucle y los otros segmentos se denominan orejas. El punto final de cada oreja se fija en la primera línea. Una vez que se encuentra el decaimiento del gráfico en el oído, no es difícil determinar si los dos bordes se encuentran en un ciclo común. Esta información se puede utilizar en algoritmos para determinar dos conexiones, tres conexiones, cuatro conexiones y planitud. La caries auricular paralela se puede encontrar utilizando trabajo lineal y profundidad de registro, independientemente de la estructura del trazado. Por lo tanto, esta técnica se puede utilizar para reemplazar la técnica de secuenciación estándar resolviendo estos problemas y buscando la profundidad primero.

## 7. **Modelos de Algoritmos Paralelos**

### 7. 1. **Paralelismo de datos**

Aquí Las tareas son mapeadas de manera cuasi-estática. Las operaciones paralelas son muy similares. Además de esto, necesita poca sincronización y asignación.

### 7. 2. **Grafo de tareas**

Generalmente se usa cuando tenemos grandes volúmenes de datos. Y Usualmente se mapean estáticamente.

### 7. 3. **Conjunto de trabajadores (work pool)**

Mapeo dinámico de tareas a procesos. Es muy útil para el balance de carga. En este el mapeo puede ser centralizado o descentralizado.

### 7. 4. **Maestro-esclavo (master-slave)**

Un maestro genera trabajo y los esclavos ejecutan. Ayuda a resolver problemas de cuellos de botella.

## 7. 5. Productor-consumidor (pipeline producer-consumer)

Una secuencia de datos pasa a través de una serie de procesos. Aquí cada proceso realiza una tarea y todos los procesos actúan a la vez formando un pipeline.

## 8. Algoritmos de Búsquedas y Ordenamiento (Adjuntar Pseudocódigo y código de cada uno)

### 8. 1. Búsqueda Secuencial

Consiste en tomar un dato clave que identifica al elemento que se busca y hacer un recorrido a través de todo el arreglo comparando el dato de referencia con el dato de cada posición.

```
int[] ii_arreglo = new int[15] { 9, 15, 1, 3, 98, 23, 76, 7, 29, 67, 4, 45, 87, 34, 72 };

int Tamaño = ii_arreglo.Length;
int Posicion = 0;

while (Posicion < Tamaño)
{
    if (ii_arreglo[Posicion] == num)
    {
        encontrado_bs = "Si";
        stopwatch_bs.Stop();
        arreglo_bs = ii_arreglo;
        return ms;
    }
    else
    {
        Posicion++;
    }
}

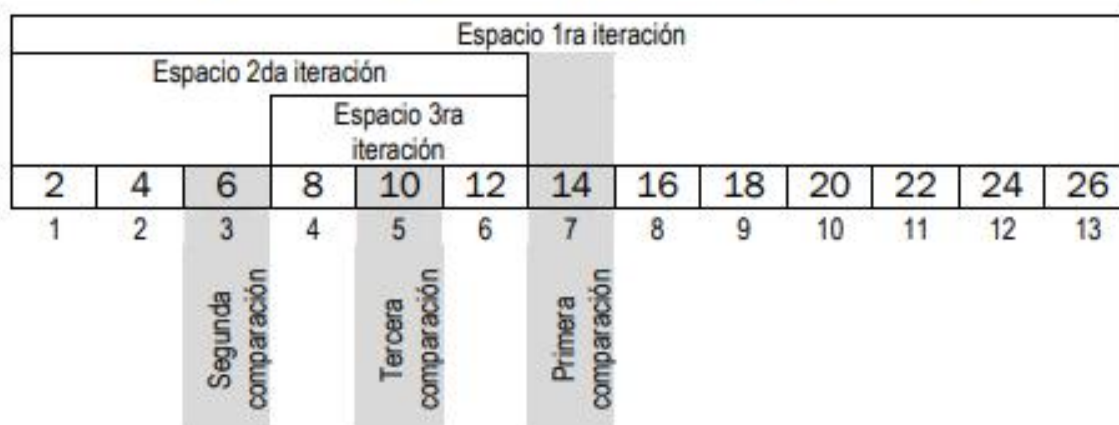
encontrado_bb = "No";
```

## 8. 2. Búsqueda Binaria

Es más eficiente que la búsqueda secuencial pero sólo se puede aplicar sobre vectores o listas de datos ordenados.

En la búsqueda binaria no se hace un recorrido de principio a fin, sino que se delimita progresivamente el espacio de búsqueda hasta llegar al elemento buscado. La primera comparación se hace con el elemento de la mitad del arreglo, si aquel no es el dato buscado, se decide si buscar en la mitad inferior o en la mitad superior según la clave sea menor o mayor del elemento de la mitad. Se toma como espacio de búsqueda la mitad del vector que corresponda y se procede de igual forma, se compara con el elemento del centro, si ese no es el que se busca, se toma un nuevo espacio de búsqueda correspondiente a la mitad inferior o superior del espacio anterior, se compara nuevamente con el elemento del centro, y así sucesivamente hasta que se encuentre el elemento o el espacio de búsqueda se haya reducido un elemento.

Figura 114. Diagrama de flujo del algoritmo de búsqueda lineal



```

int[] ii_arreglo = new int[15] { 9, 15, 1, 3, 98, 23, 76, 7, 29, 67, 4, 45, 87, 34, 72 };

int t;
for (int a = 1; a < ii_arreglo.Length; a++)
    for (int b = ii_arreglo.Length - 1; b >= a; b--)
    {
        if (ii_arreglo[b - 1] > ii_arreglo[b])
        {
            t = ii_arreglo[b - 1];
            ii_arreglo[b - 1] = ii_arreglo[b];
            ii_arreglo[b] = t;
        }
    }

int l = 0, h = 14;
int m = 0;
bool found = false;
while (l <= h && found == false)
{
    m = (l + h) / 2;
    if (ii_arreglo[m] == num)
        found = true;
    if (ii_arreglo[m] > num)
        h = m - 1;
    else
        l = m + 1;
}
if (found == false)
    encontrado_bb = "No";
else
    encontrado_bb = "Si";

```

### 8. 3.     **Algoritmo de Ordenamiento de la Burbuja**

La idea consiste en realizar varios recorridos secuenciales en el arreglo intercambiando los elementos adyacentes que estén desordenados. en la primera iteración se lleva el máximo a su posición definitiva al final del arreglo. En la segunda iteración se lleva el segundo máximo a su posición definitiva. Y así continúa hasta ordenar todo el arreglo.

```
int[] ii_arreglo = new int[15] { 9, 15, 1, 3, 98, 23, 76, 7, 29, 67, 4, 45, 87, 34, 72 };  
  
int t;  
for (int a = 1; a < ii_arreglo.Length; a++)  
    for (int b = ii_arreglo.Length - 1; b >= a; b--)  
    {  
        if (ii_arreglo[b - 1] > ii_arreglo[b])  
        {  
            t = ii_arreglo[b - 1];  
            ii_arreglo[b - 1] = ii_arreglo[b];  
            ii_arreglo[b] = t;  
        }  
    }  
}
```

### 8. 4.     **Quick Sort**

El algoritmo trabaja de la siguiente forma:

Elegir un elemento del conjunto de elementos a ordenar, al que llamaremos pivote. Re-situar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.

La lista queda separada en dos sub listas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.

Repetir este proceso de forma recursiva para cada sub lista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

```

private int[] QuickSort(int[] ii_arreglo, int primero, int ultimo)
{
    int i, j, central;
    double pivote;
    central = (primero + ultimo) / 2;
    pivote = ii_arreglo[central];
    i = primero;
    j = ultimo;
    do
    {
        while (ii_arreglo[i] < pivote) i++;
        while (ii_arreglo[j] > pivote) j--;
        if (i <= j)
        {
            int temp;
            temp = ii_arreglo[i];
            ii_arreglo[i] = ii_arreglo[j];
            ii_arreglo[j] = temp;
            i++;
            j--;
        }
    } while (i <= j);

    if (primero < j)
    {
        ii_arreglo= QuickSort(ii_arreglo, primero, j);
    }
    if (i < ultimo)
    {
        ii_arreglo =QuickSort(ii_arreglo, i, ultimo);
    }
    return ii_arreglo;
}

```

## 8.5. Método de Inserción

Es un algoritmo de fácil aplicación que permite el ordenamiento de una lista. Su funcionamiento consiste en el recorrido por la lista seleccionando en cada iteración un valor como clave y compararlo con el resto insertándolo en el lugar correspondiente.

```
int[] ii_arreglo = new int[15] { 9, 15, 1, 3, 98, 23, 76, 7, 29, 67, 4, 45, 87, 34, 72 };

    int auxili;
    int j;
    for (int i = 0; i < ii_arreglo.Length; i++)
    {
        auxili = ii_arreglo[i];
        j = i - 1;
        while (j >= 0 && ii_arreglo[j] > auxili)
        {
            ii_arreglo[j + 1] = ii_arreglo[j];
            j--;
        }
        ii_arreglo[j + 1] = auxili;
    }
```

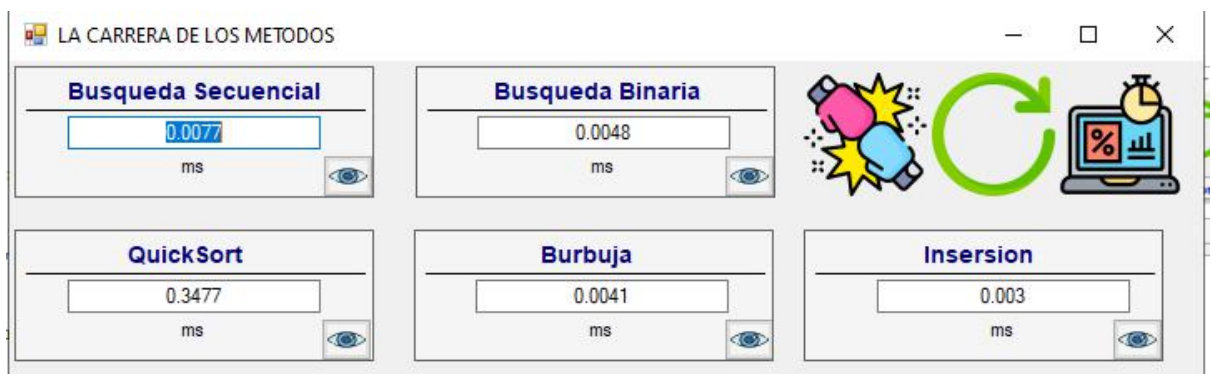
## 9. Programa desarrollado

### 9.1. Explicación de su funcionamiento

La aplicación consta de 5 apartados donde se presentara el tiempo que tarda en terminar de correr cada algoritmo por separado de forma paralela, este tiempo se visualizara en mili-segundos.

Tiene 3 botones, uno que inicia la ejecución de los algoritmos tipo carrera, otro que permite ver el listado de las distintas corridas ejecutadas, por ultimo uno que reinicia las tablas.

### 9.2. Fotos de la aplicación





Ver\_Arreglo

### ARREGLO ORIGINAL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
**	9	15	1	3	98	23	76	7	29	67	4	45	87	34	72

### RESULTADO

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
*	9	15	1	3	98	23	76	7	29	67	4	45	87	34	72

B.SECUENCIAL Numero Buscado: 67 Encontrado: SI

Ver\_Arreglo

### ARREGLO ORIGINAL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
**	9	15	1	3	98	23	76	7	29	67	4	45	87	34	72

### RESULTADO

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
*	1	3	4	7	9	15	23	29	34	45	67	72	76	87	98

B.BINARIA Numero Buscado: 67 Encontrado: SI

Ver\_Arreglo

**ARREGLO ORIGINAL**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
▶*	9	15	1	3	98	23	76	7	29	67	4	45	87	34	72

**RESULTADO**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
*	1	3	4	7	9	15	23	29	34	45	67	72	76	87	98

QUICKSORT

Ver\_Arreglo

**ARREGLO ORIGINAL**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
▶*	9	15	1	3	98	23	76	7	29	67	4	45	87	34	72

**RESULTADO**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
*	1	3	4	7	9	15	23	29	34	45	67	72	76	87	98

BURBUJA

Ver\_Arreglo

### ARREGLO ORIGINAL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
▶▶	9	15	1	3	98	23	76	7	29	67	4	45	87	34	72

### RESULTADO

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
*	1	3	4	7	9	15	23	29	34	45	67	72	76	87	98

INSERION

Ver\_Average

### TIEMPOS

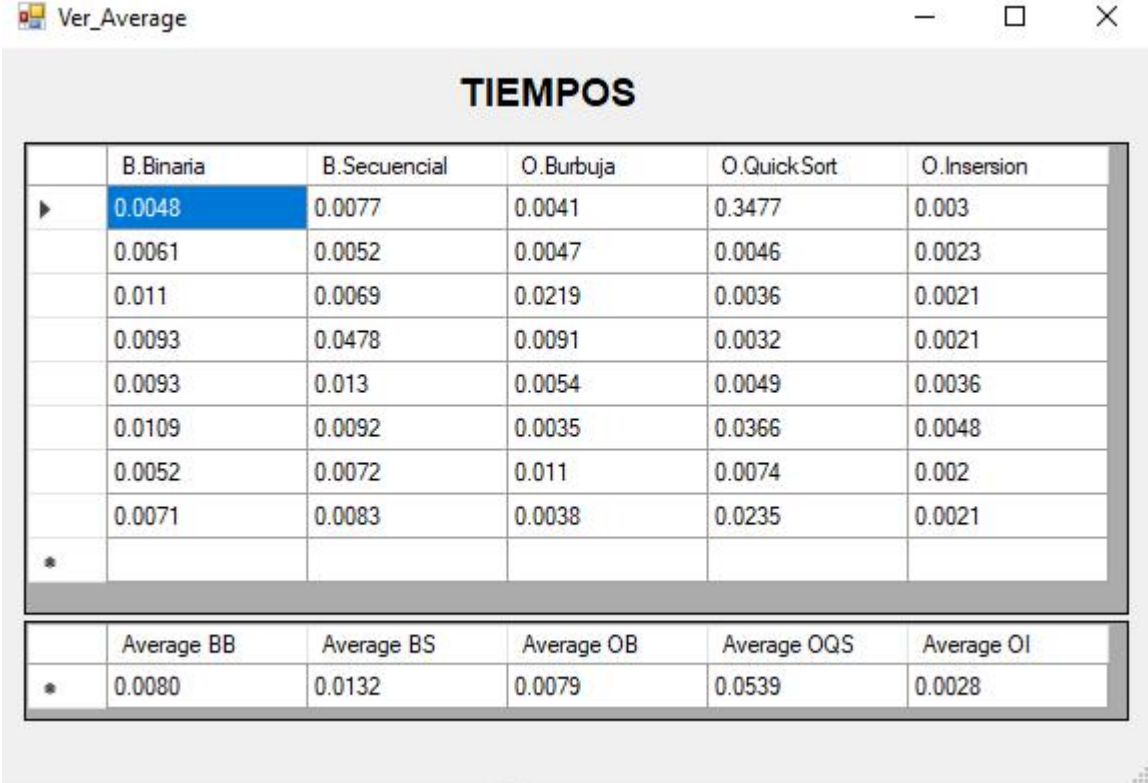
	B.Binaria	B.Secuencial	O.Burbuja	O.Quick Sort	O.Inserion
▶	0.0048	0.0077	0.0041	0.3477	0.003
	0.0061	0.0052	0.0047	0.0046	0.0023
	0.011	0.0069	0.0219	0.0036	0.0021
	0.0093	0.0478	0.0091	0.0032	0.0021
	0.0093	0.013	0.0054	0.0049	0.0036
	0.0109	0.0092	0.0035	0.0366	0.0048
	0.0052	0.0072	0.011	0.0074	0.002
	0.0071	0.0083	0.0038	0.0235	0.0021
*					
	Average BB	Average BS	Average OB	Average OQS	Average OI
*	0.0080	0.0132	0.0079	0.0539	0.0028

### 9.3. Link de GitHub y Ejecutable de la aplicación

[https://github.com/sugeiri/PF\\_AlgoritmosParalelos.git](https://github.com/sugeiri/PF_AlgoritmosParalelos.git)

Link de video de Youtube: <https://youtu.be/FzFfqUETHRs>

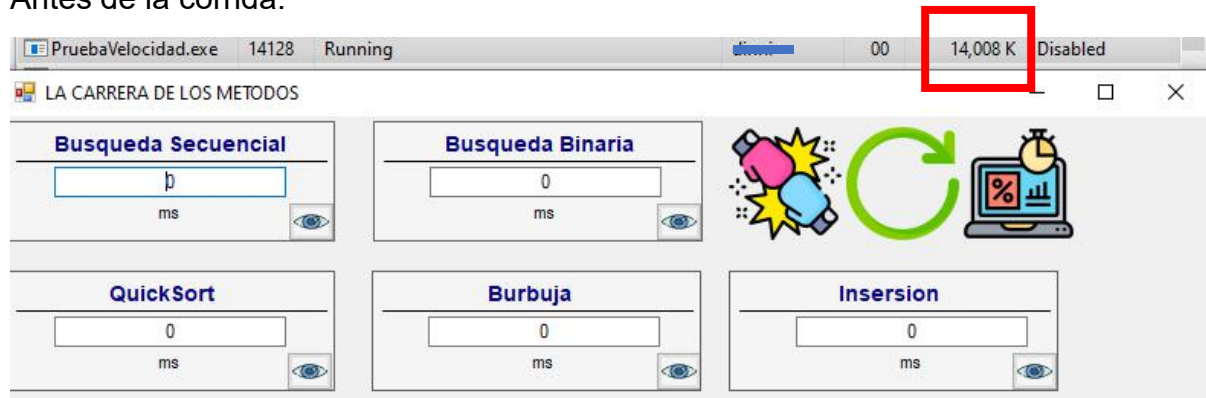
### 9.4. Resultados (Tiempo en terminar los ordenamientos y búsqueda de cada algoritmo)



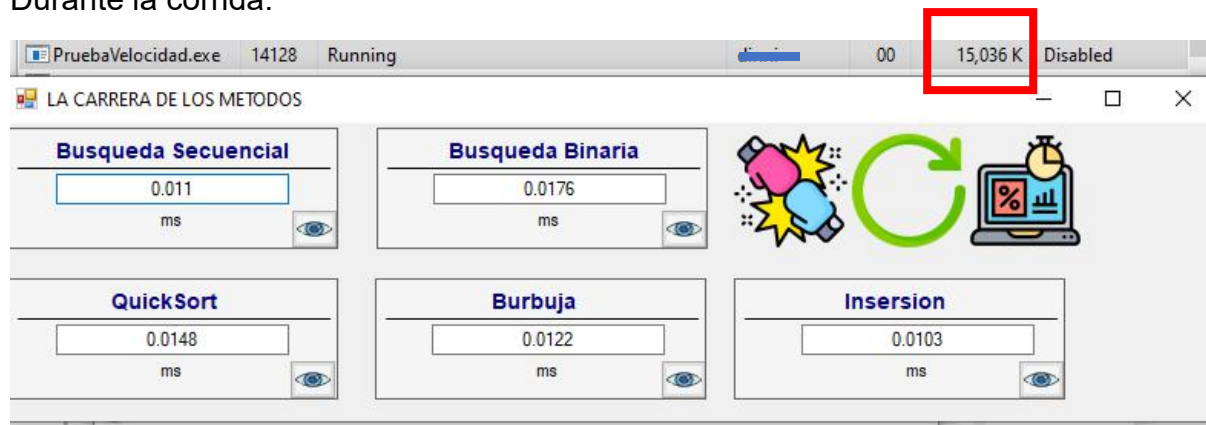
	B.Binaria	B.Secuencial	O.Burbuja	O.QuickSort	O.Insersion
▶	0.0048	0.0077	0.0041	0.3477	0.003
	0.0061	0.0052	0.0047	0.0046	0.0023
	0.011	0.0069	0.0219	0.0036	0.0021
	0.0093	0.0478	0.0091	0.0032	0.0021
	0.0093	0.013	0.0054	0.0049	0.0036
	0.0109	0.0092	0.0035	0.0366	0.0048
	0.0052	0.0072	0.011	0.0074	0.002
	0.0071	0.0083	0.0038	0.0235	0.0021
*					
	Average BB	Average BS	Average OB	Average OQS	Average OI
*	0.0080	0.0132	0.0079	0.0539	0.0028

## 10. ¿Qué tanta memoria se consumió este proceso?

Antes de la corrida:



Durante la corrida:



## 11. ¿Cuál fue el algoritmo que realizo la búsqueda y el ordenamiento más rápido? Explique.

El método de búsqueda más rápido que hubo fue el de búsqueda binaria, debido a que comienza por comparar el elemento del medio del arreglo con el valor buscado. Si el valor buscado es igual al elemento del medio, su posición en el arreglo es retornada. Si el valor buscado es menor o mayor que el elemento del medio, la búsqueda continua en la primera o segunda mitad, respectivamente, dejando la otra mitad fuera de consideración.

El método de ordenamiento más rápido que hubo fue el de Quicksort, debido a que usa menos métodos para ejercer la función de ordenamiento ya que usa la técnica de dividir la cantidad de elementos existentes y comienza a organizarlos partiendo de uno en específico.

## 12. Conclusión

Según los datos obtenidos en esta investigación se concluyó que todos los métodos, tanto de búsqueda como de ordenamiento, son usados en el ambiente que se requiera, así como también cual de ellos por renglón fue el más rápido entre los testeados.

De igual forma, los resultados obtenidos en las comparaciones realizadas demuestran que en el renglón de método de búsqueda más rápido el ganador fue el de Búsqueda Binaria que por su procedimiento demuestra que sigue siendo de los principales en usarse y de los que le ahorra tiempo al usuario en madera de agilizar otros procesos que interactúen cuando se esté implementando la búsqueda.

Para el método de ordenamiento, el más rápido fue Quicksort, que utilizando particionamiento entre los elementos ya guardados, usa como punto de inicio para empezar a ordenarlos.

## 13. Bibliografías

[https://es.wikipedia.org/wiki/Algoritmo\\_paralelo#:~:text=En%20las%20ciencias%20de%20la,y%20obtener%20el%20resultado%20correcto.](https://es.wikipedia.org/wiki/Algoritmo_paralelo#:~:text=En%20las%20ciencias%20de%20la,y%20obtener%20el%20resultado%20correcto.)

<https://www.hebergementwebs.com/tutorial-sobre-el-algoritmo-paralelo/algoritmo-paralelo-guia-rapida>

<https://www.inf.utfsm.cl/~noell/IWI-131-p1/Tema8b.pdf>

<http://biolab.uspceu.com/aotero/recursos/docencia/TEMA%208.pdf>

[http://webdelprofesor.ula.ve/ingenieria/gilberto/paralela/08\\_DisenioDeAlgoritmosParalelos.pdf](http://webdelprofesor.ula.ve/ingenieria/gilberto/paralela/08_DisenioDeAlgoritmosParalelos.pdf)

<https://www.frbb.utn.edu.ar/hpc/lib/exe/fetch.php?media=2016-03-disenno-algoritmos-paralelos.pdf>

Grama, A. Karypis, G., Kumar, V. Gupta, A., 2003. Introduction to Parallel Computing, 2nd ed., Addison Wesley,

Mohcine, J., Contassot-Vivier, S. Couturir, R., 2007. Parallel Iterative Algorithms: From Sequential to Grid Computing, Chapman and Hall/CRC Numerical Analysis and Scientific Computation Series.

Correa, R., de Castro, I., Fiallos, M., Gomez L.F. (Editors), 2002. Models for Parallel and Distributed Computation: Theory, Algorithmic Techniques and Applications (Applied Optimization), Kluwer Academic Pu.