UNIVERSIDAD TECNOLÓGICA DE SANTIAGO
(UTESA)
Área de Arquitectura e Ingeniería
Carrera de Ingeniería en Sistemas Computacionales.

ALGORITMOS PARALELOS

Tarea Semana 10

PRESENTADO POR:

José Rodolfo Morel.	1-16-0328.
Rudelvi A. Valenzuela	1-17-1005
Brayan Isaac Mancebo.	2-16-0375.

ASESOR:

Ing. Iván Mendoza.

GRUPO:
INF-025-001

Santiago De Los Caballeros
República Dominicana
Agosto, 2021.

ÍNDICE

Introducción 3

1. Descripción del Proyecto 4
2. Objetivos 4
    - **a.** Objetivo General 5
    - **b.** Objetivos Específicos 5
3. Definición de Algoritmos Paralelos 5
4. Etapas de los Algoritmos paralelos 6
    - **a.** Partición 6
    - **b.** Comunicación 7
    - **c.** Agrupamiento 8
    - **d.** Asignación 9
5. Técnicas Algorítmicas Paralelas 10
6. Modelos de Algoritmos Paralelos 11
7. Algoritmos de Búsquedas y Ordenamiento 13
    - **a.** Búsqueda Secuencial 13
    - **b.** Búsqueda Binaria 15
    - **c.** Algoritmo de Ordenamiento de la Burbuja 16
    - **d.** Quick Sort 18
    - **e.** Método de Inserción. 2
8. Programa desarrollado 23
    - **a.** Explicación de su funcionamiento 23
    - **b.** Link de Github y Ejecutable de la aplicación 27
    - **c.** Resultados 27
    - **d.** ¿Qué tanta memoria se consumió este proceso? 28
9. ¿Cuál fue el algoritmo que realizó la búsqueda y el ordenamiento más rápido? Explique.

Conclusión 29

Bibliografías **29**

Introducción

El procesamiento en paralelo, es una técnica que favorece considerablemente el desarrollo de software, mediante la optimización de procesos a través de la reducción de recursos y tiempo de procesamiento.

En esta práctica se analizan diferentes algoritmos de búsqueda y ordenamiento, para determinar cuales presentan mejores resultados en diferentes escenarios y en consecuencia poder elegir el mejor según nuestro propósito. Para lograr estos objetivos se desarrolló un programa con una interfaz amigable e intuitiva para el usuario.

1. Descripción del Proyecto

Actualmente existen diferentes algoritmos, y su procesamiento en paralelo mejora significativamente su rendimiento. En este proyecto se presentan diferentes algoritmos de búsqueda dentro de una determinada estructura de datos en diferentes escenarios, reflejando las ventajas ofrecidas entre uno y otro.

2. Objetivos

a. Objetivo General

- Comparar el rendimiento de diferentes algoritmos.

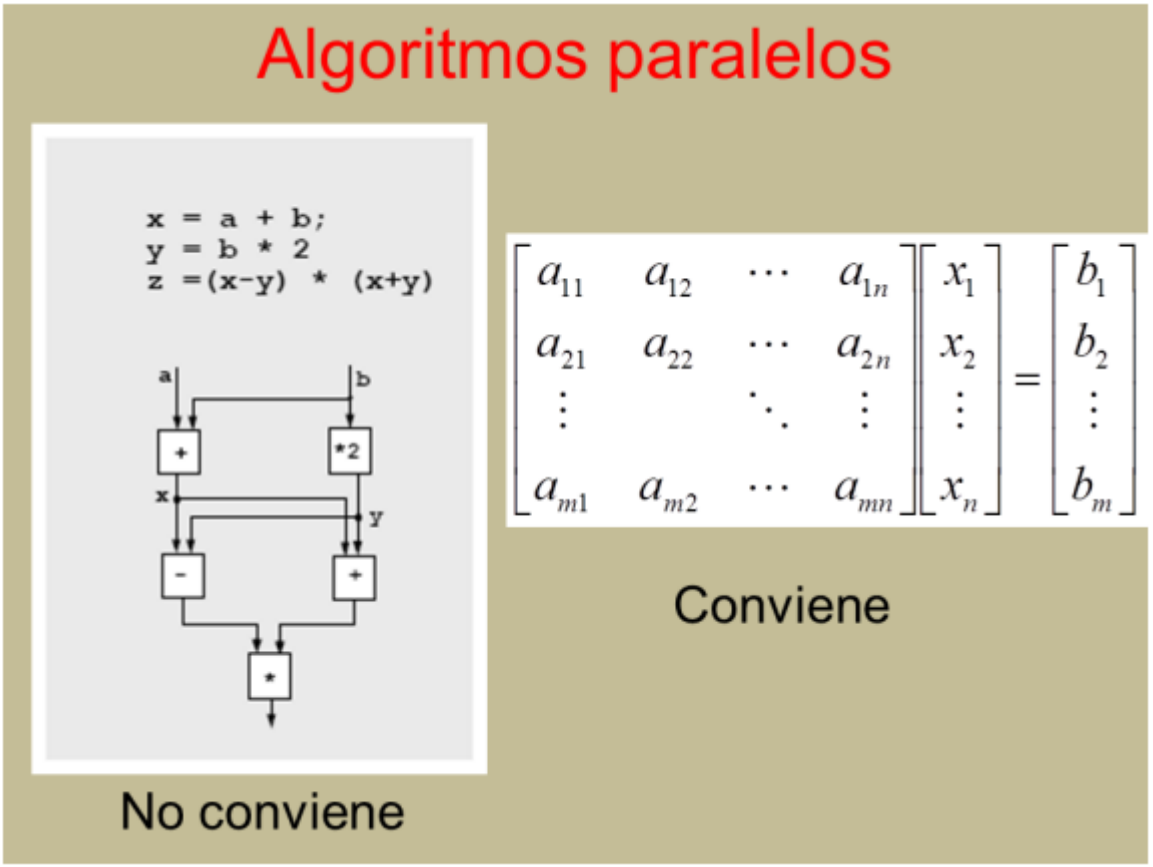
b. Objetivos Específicos

- Determinar el tiempo de ejecución de los algoritmos presentados.
- Seleccionar la mejor alternativa como algoritmo de búsqueda.

3. Definición de Algoritmos Paralelos

Normalmente en un algoritmo convencional un conjunto de instrucciones se ejecutan de manera secuencial en un único procesador una después de otra, pero en los algoritmos paralelos las instrucciones se separan en varias partes que son ejecutadas de forma simultánea en diferentes procesadores.

Debido a esto los algoritmos paralelos representan una enorme ventaja de rendimiento comparado con los algoritmos secuenciales. Por ejemplo, si un algoritmo que se conforma de 5 instrucciones que tardan 5 segundos en ejecutarse cada una, se ejecuta de forma secuencial el algoritmo tarda 25 segundos en ejecutarse, mientras en paralelo cada instrucción se ejecutará al mismo tiempo por lo que el tiempo sería de solo 5 segundos en total.

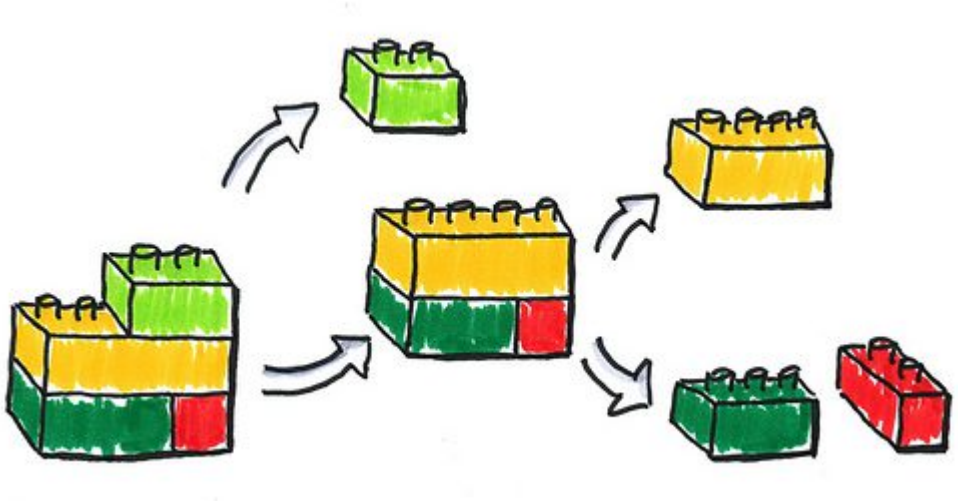


4. Etapas de los Algoritmos paralelos

A la hora de diseñar algoritmos paralelos se deben tener en cuenta una serie de aspectos independientemente de la estructura del procesador o la máquina, es necesario seguir una serie de pasos o rutinas para asegurarnos que nuestro algoritmo está correctamente estructurado, esas fases son:

a. Partición

En esta parte se inicia primero analizando el problema y tratando de descomponerlo en problemas más simples que pueda ser realizados de forma independiente por diferentes procesadores o en diferentes rutinas de ejecución, básicamente se trata de aplicar el concepto divide y vencerás en nuestro problema.

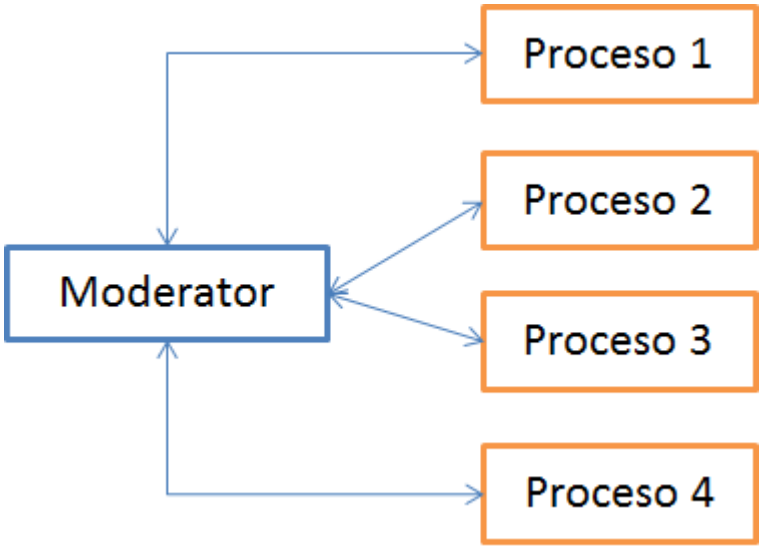


b. Comunicación

Después de dividir nuestros problemas en diferentes partes donde se tratará de dar solución a cada una en diferentes rutinas o procesos es necesario diseñar y mantener una forma de comunicación entre las diferentes tareas para que estas puedan trabajar en conjunto y coordinarse.

En esta fase es importante definir los canales de comunicación que se usarán y los datos que se transmiten entre procesos.

La comunicación es fundamental en los algoritmos paralelos a la hora de compartir recursos como la memoria, pues al escribir en un fichero o una variable un proceso debe avisar a los otros para que estos se coloquen en espera, también deben crearse los mecanismos para definir el orden de atención de cada proceso como si el primero en entrar a la pila de espera es el último o el primero en salir de ella.

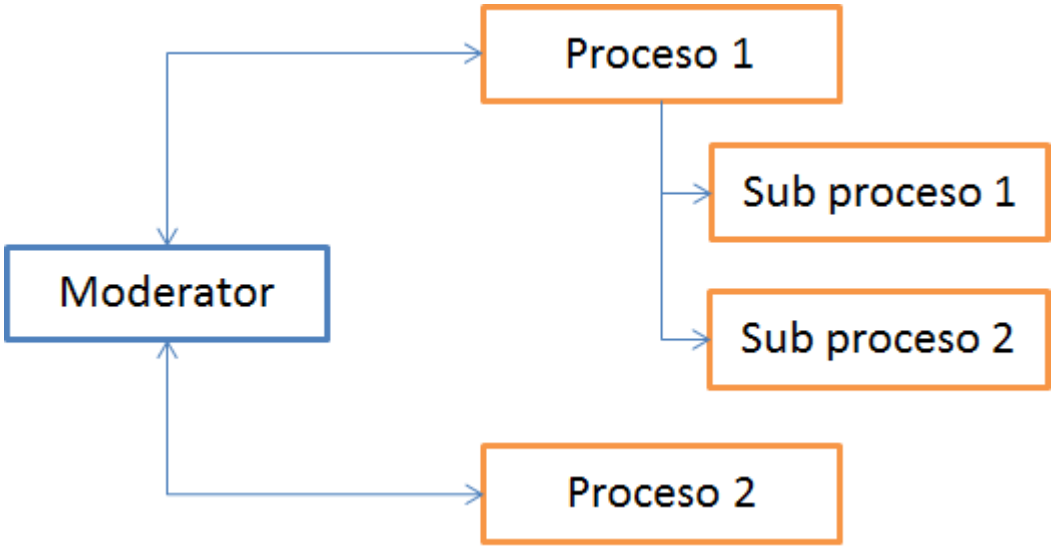


c. Agrupamiento

Después de realizar la descomposición del problema y establecer los mecanismos de comunicación entre las diferentes tareas que se ejecutarán es importante hacer un análisis tomando en cuenta las características de la máquina que ejecuta dicho algoritmo.

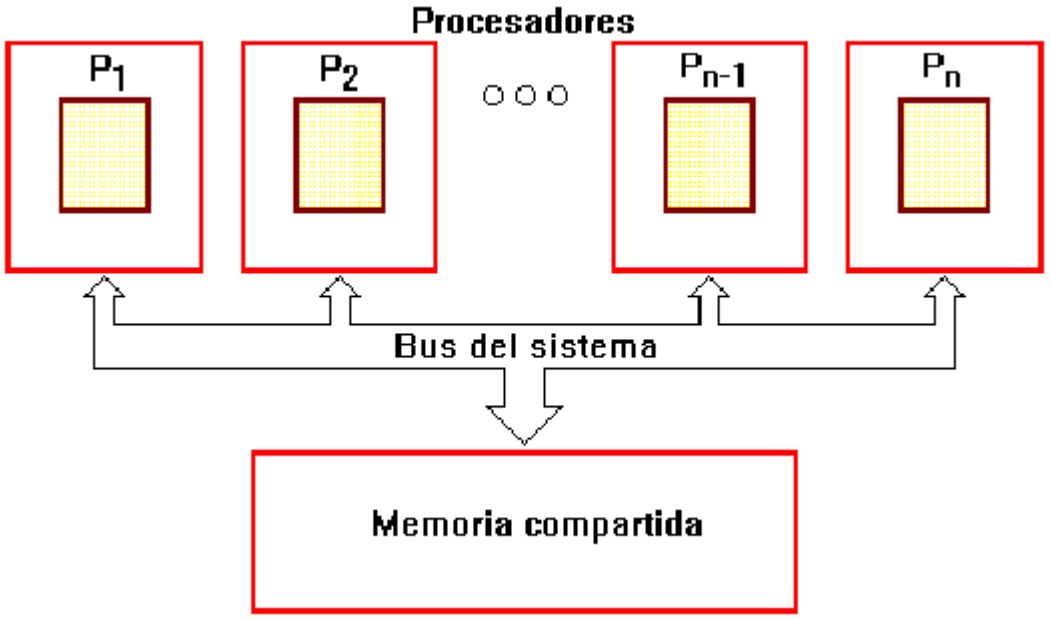
Es decir si el algoritmo está pensado para ser usado en dispositivos con baja potencia como smartphones se debería considerar reducir el número de tareas a ejecutarse en paralelo y optimizar el número de comunicación y datos consumidos por cada proceso en pro de optimizar el uso de la batería.

Por el contraria y se implementará en computadores potentes se podría considerar hacer el algoritmo más eficiente aumentando la redundancia de datos y el uso de la memoria para ahorrar ancho de banda al comunicar sólo los datos básicos necesarios.



d. Asignación

De esta etapa mayormente se encargan los sistemas operativos los cuales se encargan de repartir los procesos y tareas entre los diferentes procesadores disponibles balanceando la carga de trabajo entre ellos de forma que se puedan aprovechar la mayor cantidad de recursos posibles.



5. Técnicas Algorítmicas Paralelas

Arboles balanceados: Es un árbol binario que siempre está balanceado o equilibrado para todos sus nodos, desde la altura de la rama izquierda a derecha, por lo tanto, el grado de complejidad de búsqueda de este tipo de árbol se mantiene igual.

Técnica Pointer Jumping: Es una técnica utilizada para el procesamiento de datos almacenados en forma de un conjunto de árboles específicos arraigados.

Divide y vencerás: Es una técnica que trata de dividir los problemas o el problema en subproblemas, el término divide hace referencia en la división del problema en subproblemas más pequeños mientras que vencer es resolver de forma recursiva cada sub-problema de manera independiente para luego combinar las soluciones que permiten resolver el problema original.

Particionamiento: Es la descomposición de los problemas en un gran número de tareas, están los funcionales y el dominio.

Técnica de Pipelining: Sirve para dividir un problema en varias tareas que son completadas una después de otra, lo que permite conocer la información necesaria para la solución del problema total.

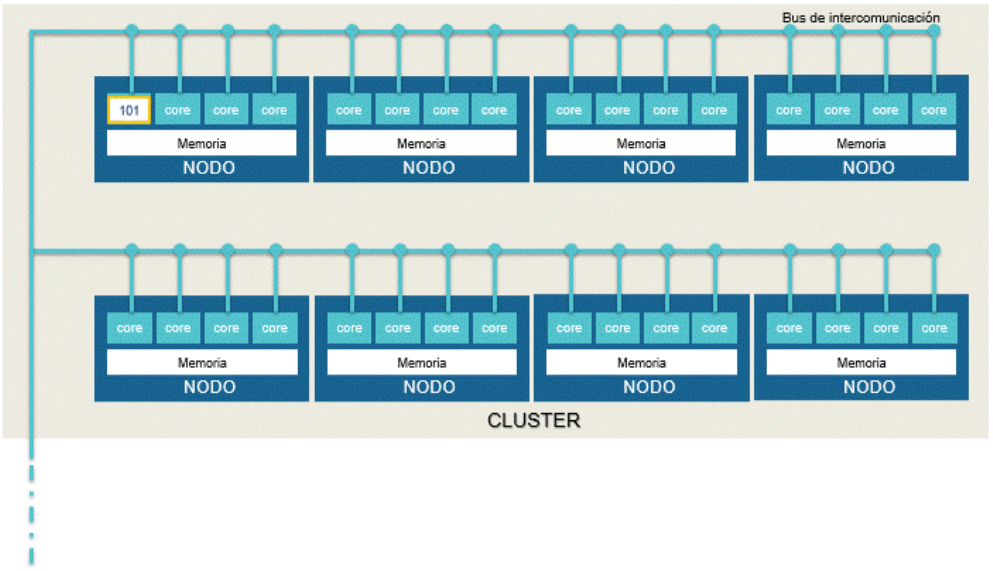
Aceleramiento en cascada: Consiste en dividir los procesos de desarrollos en pasos o fases sucesivas, las cuales sirven como hipótesis para continuar con la siguiente.

Balanceo de Cargas: Es la distribución de los recursos para aprovechar toda la capacidad del procesamiento disponible.

6. Modelos de Algoritmos Paralelos

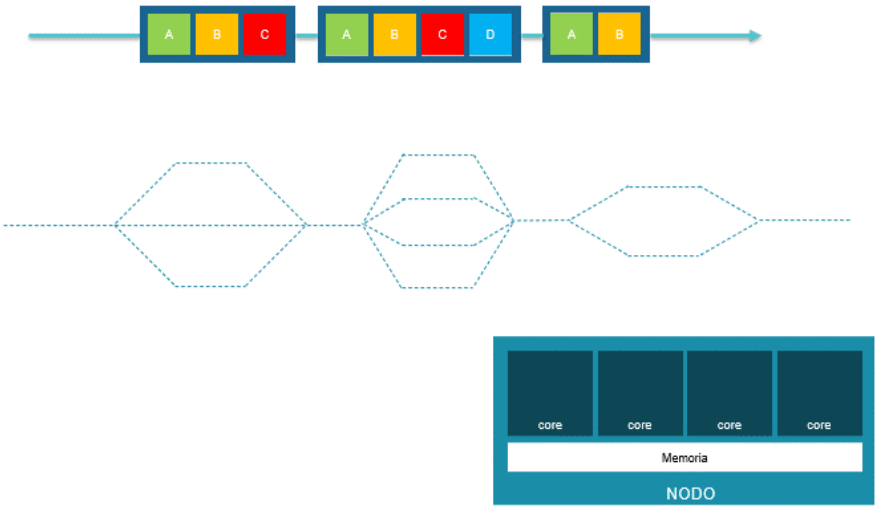
Modelo de intercambio de mensajes: Este es dividido en piezas para ser procesados dentro de un cluster de nodos conectados por un bus de intercomunicación. Este es utilizado para procesar o ejecutar el mismo proceso con la misma distribución de procesamiento por clúster.

Modelo de intercambio de mensajes

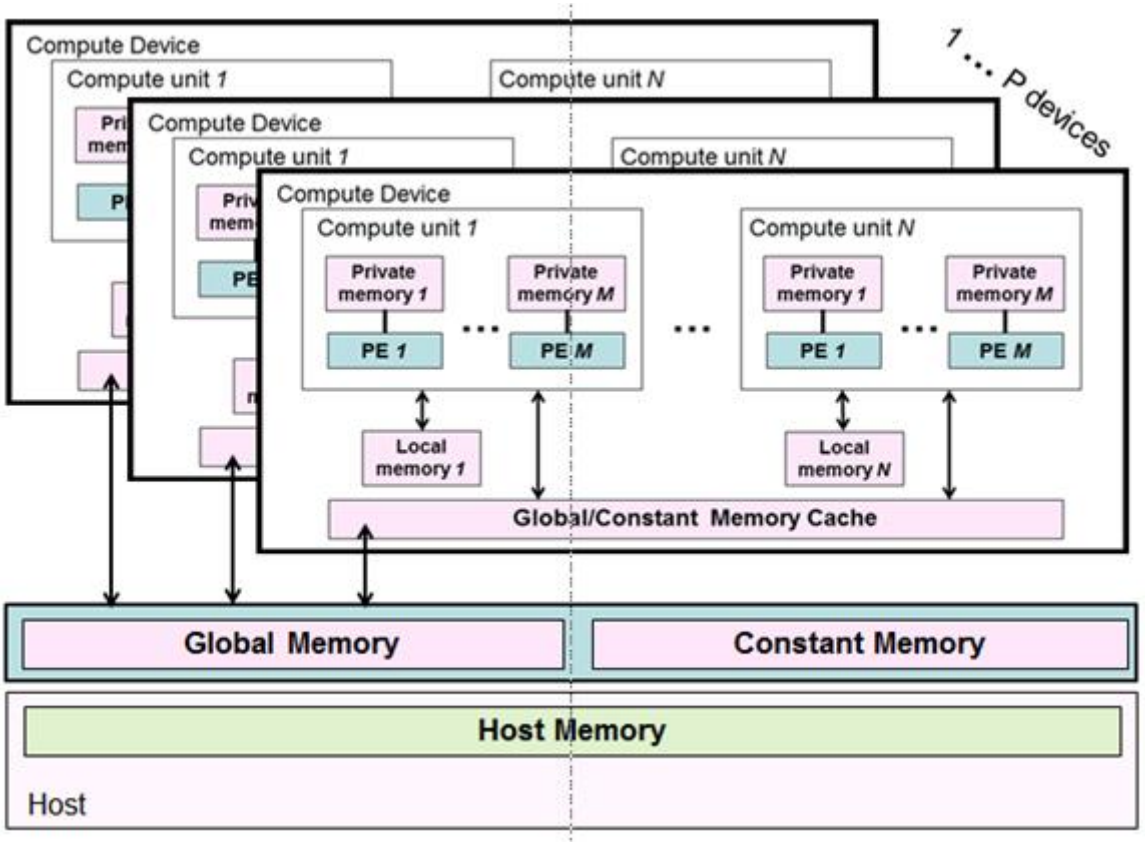


Modelo de programación por hilos: Es un modelo de memoria compartida su estructura base es un hilo con la unidad de ejecución que contiene instrucciones de un programa y la estructura de memoria necesaria para la ejecución independiente.

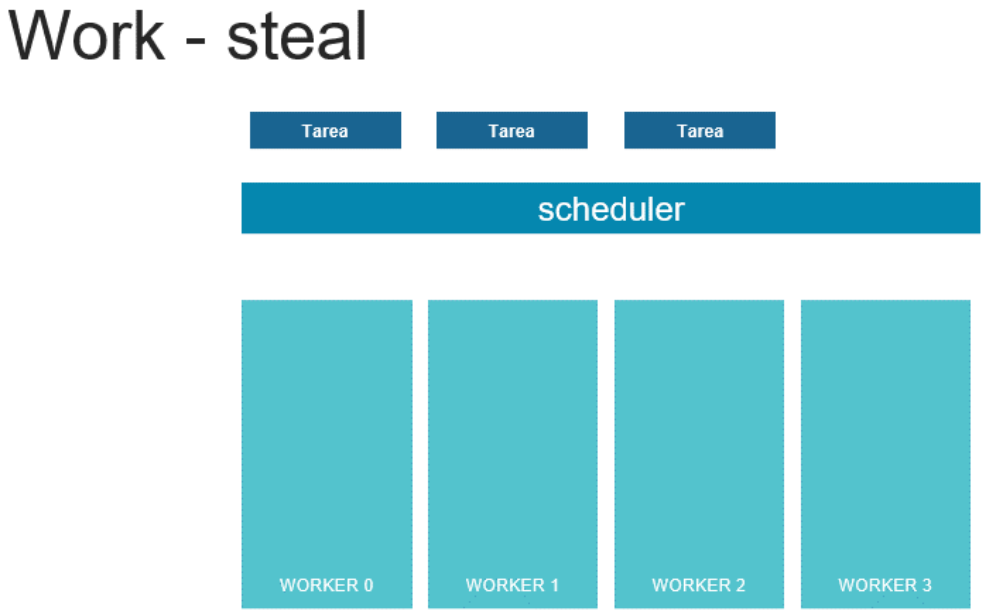
Modelo de programación por hilos



Modelos heterogéneos: Es este modelo se aprovecha al máximo los recursos de cálculos del sistema disponible mediante API y herramientas funcionales de la necesidad de especificación del hardware a utilizar o las limitantes del desempeño.



TBB (Threading Building Blocks): El modelo de TBB consiste en un organizador de las tareas que se encarga de distribuir la carga entre los distintos workers, cada worker o trabajador tiene una fila de tareas que debe ejecutar, sin embargo en caso de que algún núcleo termine antes, buscará a otro worker que tenga tareas pendientes por ejecutar y le robará la tarea más antigua para ayudarle con su carga y así hacer más eficiente el desempeño global del algoritmo.



7. Algoritmos de Búsquedas y Ordenamiento

a. Búsqueda Secuencial

PSeudocódigo

```
INICIO
    entero vec[5] ={95,25,30,15,1}
    bool encontro = falso
    entero I = 1, posicion

    LEER dato

        mientras (no (encontro) y i <= 10)
            SI (dato == vec[i]) ENTONCES
                encontro = verdadero
                posicion = i
            FIN SI
            i = i + 1
        FIN MIENTRAS

    SI (encontro) ENTONCES
        IMPRIMA “El dato se encuentra en la posición ”, posición
    FIN SI
FIN
```

Algoritmo en Java

```
public class BusquedaSecuencial {
    public static void main(String[] args) {
        int[] numeros = {12,45,67,27,89,84,65,21,44};
        int busqueda = 45;
        boolean existe= false;

        for(int b = 0; b < numeros.length; b++){

            if(numeros[b]==busqueda){
                System.out.println("El número si existe, en la posición "+(b+1));
                break;
            }

            if(b == numeros.length-1){
                existe = true;
            }
        }
        if(existe==true){

            System.out.println("El número no existe");
        }
    }
}
```

b. Búsqueda Binaria

Pseudocódigo

```
INICIO

       ENTERO numero, puntero=1, final=10, mitad
       ENTERO vec[10]= {8,11,22,35,50}
       LOGICO encontro =  “falso”
       IMPRIMIR “Favor ingrese un numero de busqueda: ”
       LEER numero

       MIENTRAS ( NO(encontro) Y puntero <= final )
              mitad = (puntero + final) / 2
              SI (numero == vec[mitad]) ENTONCES
                     encontro = “verdadero”
              SINO SI (numero < vec[mitad]) ENTONCES
                           final = mitad - 1
                     SINO
                           puntero = mitad + 1
                     FIN SI
              FIN SIN
       FIN MIENTRAS
       SI (encontro == “verdadero”) ENTONCES
              IMPRIMA “El dato se encuentra y esta en la posicion: ” , mitad
       SINO
              IMPRIMA “El dato no se encuentra”
       FIN SI
FIN
```

Algoritmo en Java

```
import java.util.Arrays;

public class BusquedaBinaria {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int[] numeros = {12,45,67,27,89,84,65,21,44};
        int busqueda = 67;
        System.out.println("Numero encontrado: "+binaria(numeros, busqueda));
    }

    public static boolean binaria(int[] numeros, int busqueda) {

        int size = numeros.length;  // Número de elementos
        int middle = size/2;        // Mitad del array

        if (numeros[middle] == busqueda)
          return true;
        else if (size == 1)
          return false;
        else if (numeros[middle] > busqueda)
          return binaria(Arrays.copyOfRange(numeros,0,middle),busqueda);
        else
          return binaria(Arrays.copyOfRange(numeros,middle+1,size),busqueda);
     }

}
```

c. Algoritmo de Ordenamiento de la Burbuja

PSeudocódigo

```
Proceso Burbuja
       Dimension lista[5];
       lista[1]=5;
       lista[2]=2;
       lista[3]=4;
       lista[4]=1;
       lista[5]=3;
       //Longitud de la lista
       lon=5;
       l=lon;

       mostrarLista(lista,lon);
    Hacer
```

```
            n=0;
            //Recorrer la lista
            Para i=2 Hasta l Con Paso 1 Hacer
                  //Verificar si los dos valores estan ordenados
                  Si lista[i-1]>lista[i]
                        //Ordenar si es necesario
                        temp=lista[i-1];
                        lista[i-1]=lista[i];
                        lista[i]=temp;
                        n=i;
                        mostrarLista(lista,lon);
                  FinSi

            FinPara
            l=n;
      Hasta Que n=0;
FinProceso

//Función para mostrar estado de la lista
SubProceso mostrarLista(lista,lon)
      Para i=1 Hasta lon Con Paso 1 Hacer
            Escribir Sin Saltar lista[i] " ";
      FinPara
      Escribir "";
FinSubProceso
```

Algoritmo en Java

```
public class Burbuja {
    public static void main(String[] args) {
        int[] numeros = {12,45,67,27,89,84,65,21,44};
        System.out.println("Antes del método de la burbuja: " +
Arrays.toString(numeros));
        burbujas(numeros);
        System.out.println("Después del método de la burbuja: " +
Arrays.toString(numeros));
    }

    private static void burbujas(int[] numeros) {
        for (int x = 0; x < numeros.length; x++) {
            // Aquí "y" se detiene antes de llegar
            // a length - 1 porque dentro del for, accedemos
            // al siguiente elemento con el índice actual + 1
            for (int y = 0; y < numeros.length - 1; y++) {
                int elementoActual = numeros[y],
                        elementoSiguiente = numeros[y + 1];
                if (elementoActual > elementoSiguiente) {
                    // Intercambiar
                    numeros[y] = elementoSiguiente;
                    numeros[y + 1] = elementoActual;
                }
            }
        }
    }

}
```

d. Quick Sort

Pseudocódigo

```
INICIO
Llenar(A)
Algoritmo quicksort(A,inf,sup)
i<-inf
j<-sup
x<-A[(inf+sup)div 2]
mientras i=<j hacer
  mientras A[i]< x hacer
     i<-i+1
  fin_mientras
  mientras A[j]>x hacer
     j<- j-1
  fin_mientras
  si i=<j entonces
    tam<-A[i]
    A[i]<-A[j]
    A[j]<-tam
    i=i+1
    j=j-1
  fin_si
fin_mientras
si inf<j
   llamar_a quicksort(A,inf,j)
fin_si
si i<sup
   llamar_a quicksort(A,i,sup)
fin_si
FIN
```

Algoritmo

```
public class AlgoritmoQuickSort {

    static void intercambio(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    static int dividir(int[] arr, int low, int high) {

        // pivote
        int pivot = arr[high];
        int i = (low - 1);

        for (int j = low; j <= high - 1; j++) {
            if (arr[j] < pivot) {
                // Incrementar índice del elemento menor.
                i++;
                intercambio(arr, i, j);
            }
        }
        intercambio(arr, i + 1, high);
        return (i + 1);
    }

    static void quickSort(int[] arr, int bajo, int alto) {
        if (bajo < alto) {
            int pi = dividir(arr, bajo, alto);
            quickSort(arr, bajo, pi - 1);
            quickSort(arr, pi + 1, alto);
        }
    }

    static void printArray(int[] arr, int size) {
        for (int i = 0; i < size; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int[] arr = {10, 7, 8, 9, 1, 5};
        int n = arr.length;

        quickSort(arr, 0, n - 1);
        System.out.println("Sorted array: ");
        printArray(arr, n);
    }
}
```

e. Método de Inserción.

Pseudocódigo

```
Proceso OrdenamientoInsercion
    N<-6 //Dimension del arreglo (Longitud)
    Dimension A(N) //Definir el arreglo
    A(1)<-5 //Ingresar a memoria el arreglo
    A(2)<-2
    A(3)<-4
    A(4)<-6
    A(5)<-1
    A(6)<-3
    mostrarArreglo(A,N) //Imprimir
    ordenarPorInsercion(A,N) //Ordenar
    mostrarArreglo(A,N) //Imprimir
FinProceso
```

```
Subproceso ordenarPorInsercion(A,N)
    Para j <- 2 hasta N
        clave <- A(j)
        i <- j-1
        Mientras i>0 y A(i)>clave Hacer
            A(i+1) <- A(i)
            i <- i-1
        FinMientras
        A(i+1) <- clave
    FinPara
FinSubProceso
```

```
Subproceso mostrarArreglo(A,N)
    Para i <- 1 hasta N
        Escribir Sin Saltar A(i)," "
    FinPara
    Escribir "";
FinSubProceso
```

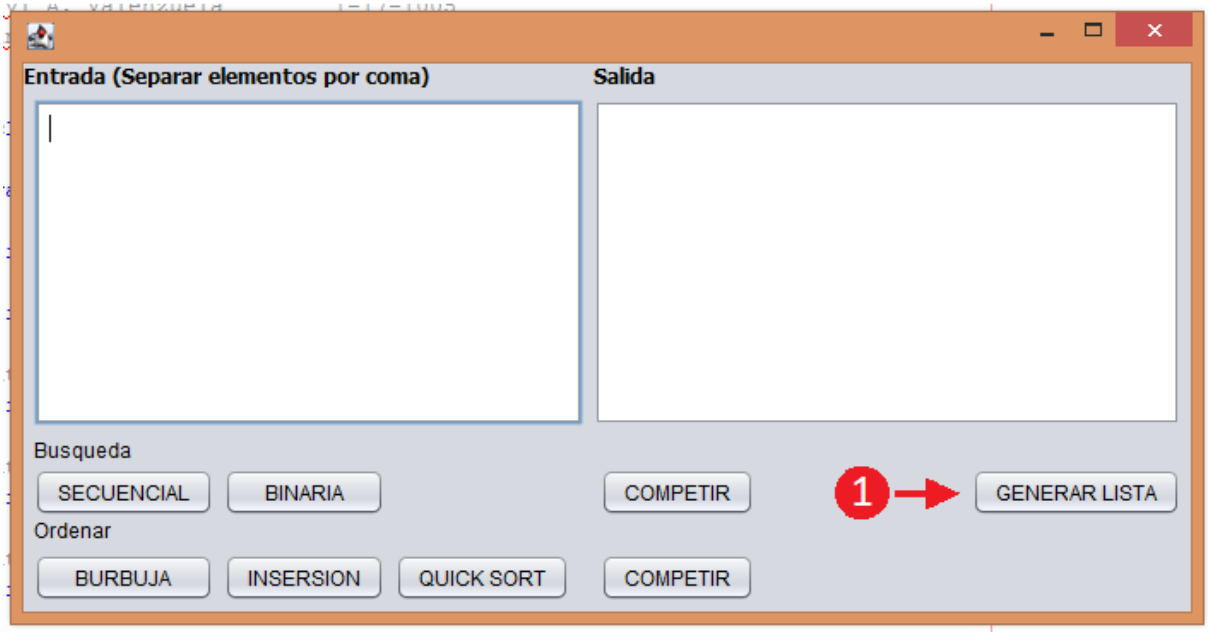
Algoritmo

```
public int[] ordenarInsercion(int[] array){
        int aux;
        for (int i = 1; i < array.length; i++) {
            aux = array[i];
            for (int j = i-1; j >=0 && array[j]>aux; j--) {
                array[j+1]=array[j];
                array[j]=aux;
            }
        }
        return array;
   }
```

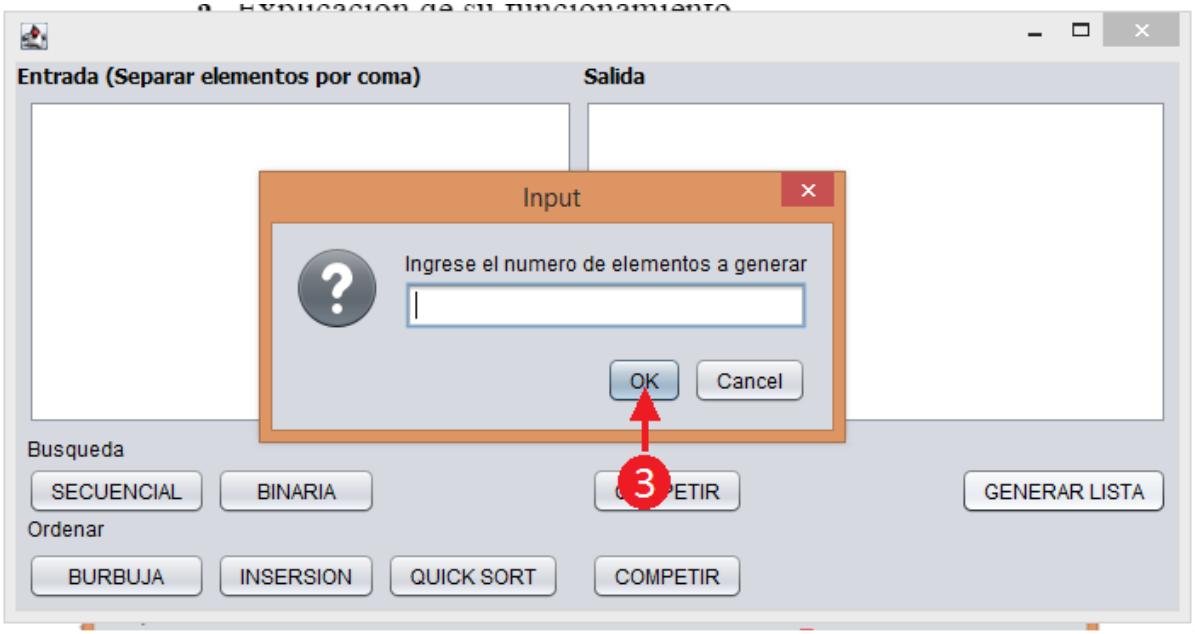
8. Programa desarrollado

a. Explicación de su funcionamiento

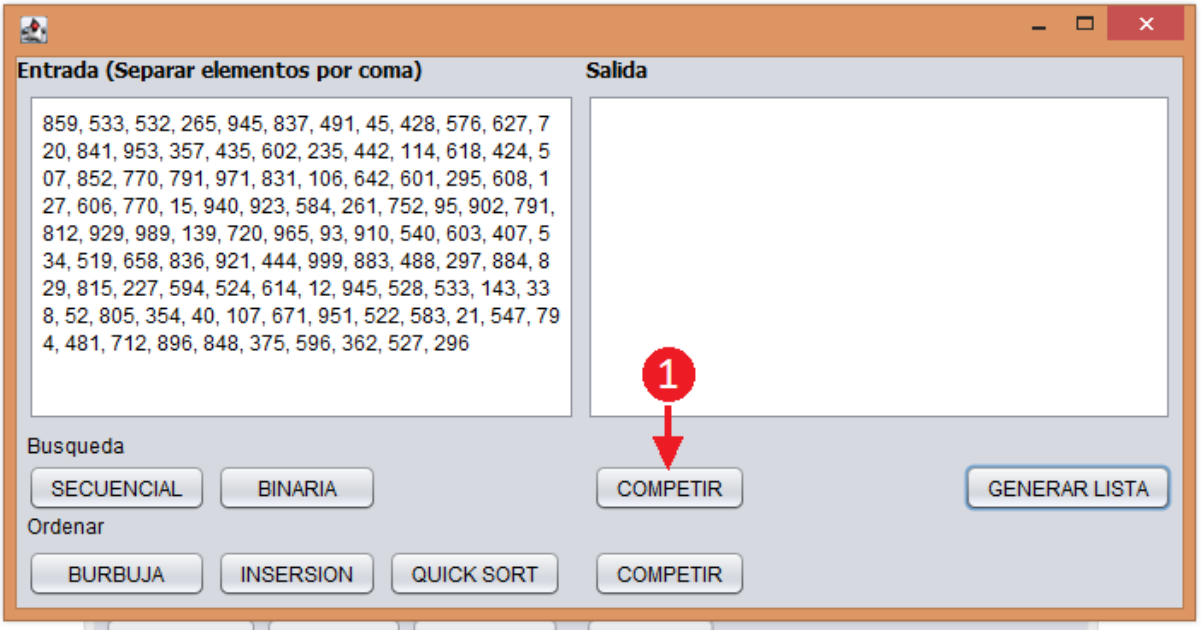
Primero genere un listado de elementos para probar (1)



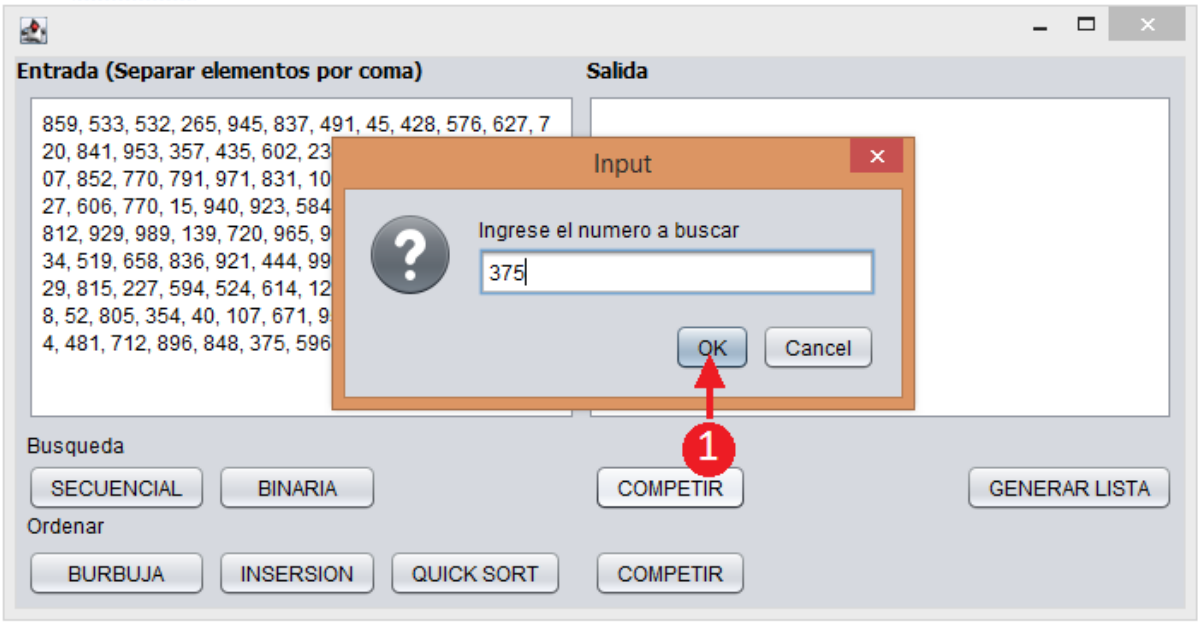
Indico el tamaño de la lista:

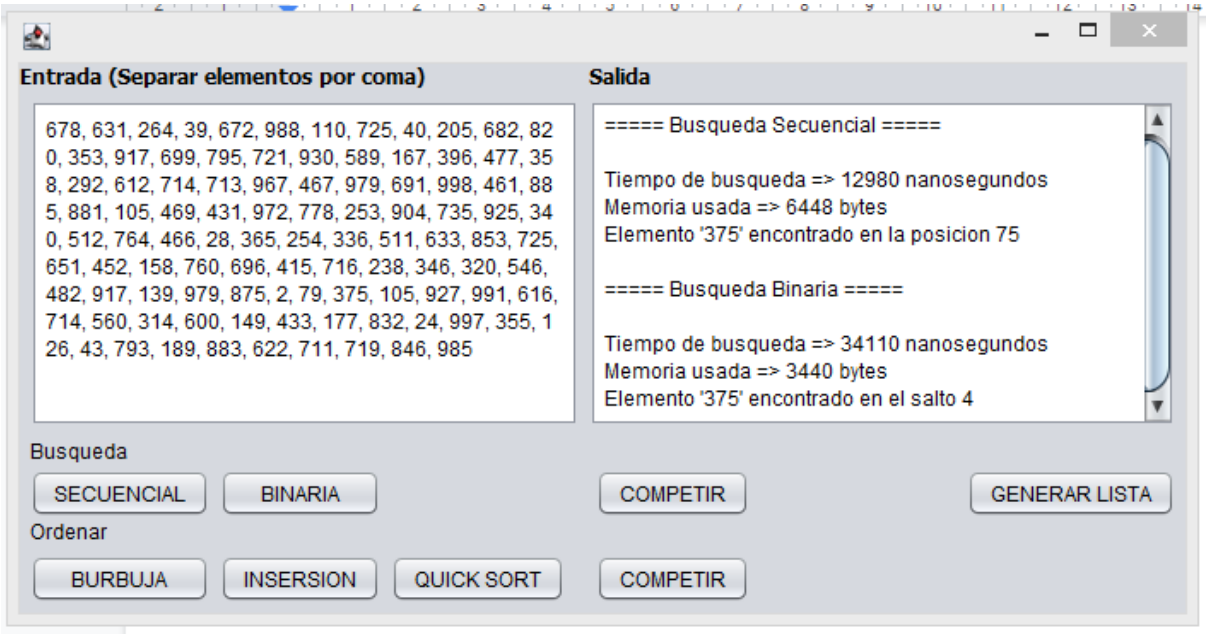


Indicamos un tamaño de 100 y procedemos a probar con la búsqueda presionando en botón competir de la sección búsqueda.

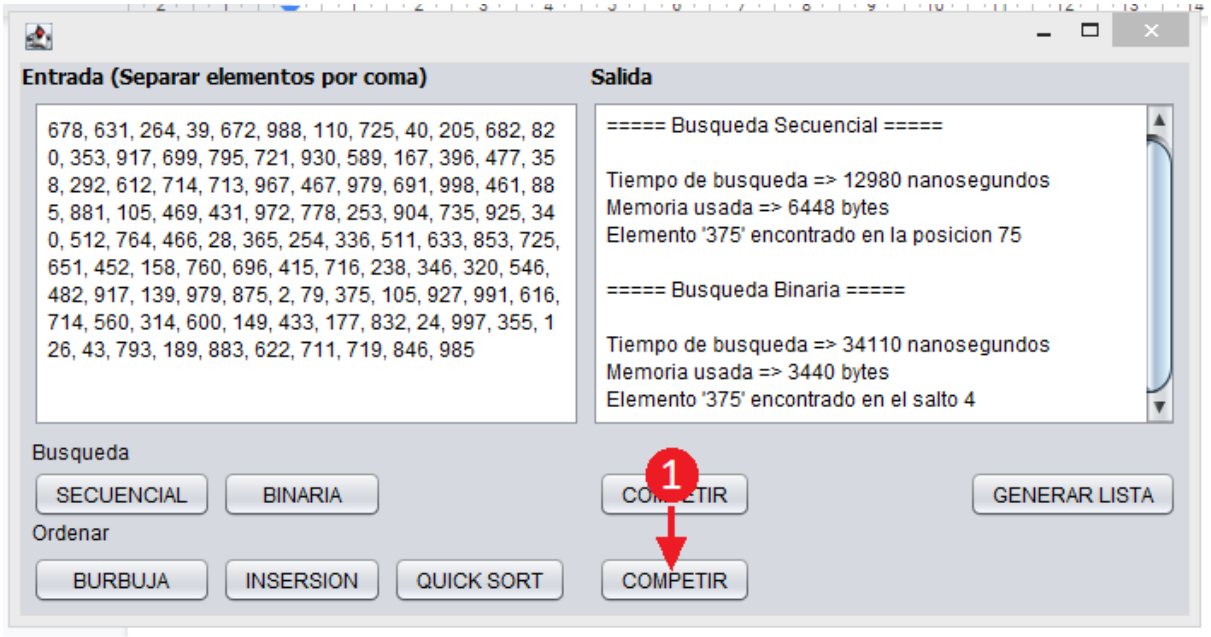


Indicamos el numero a buscar y presionamos el botón OK

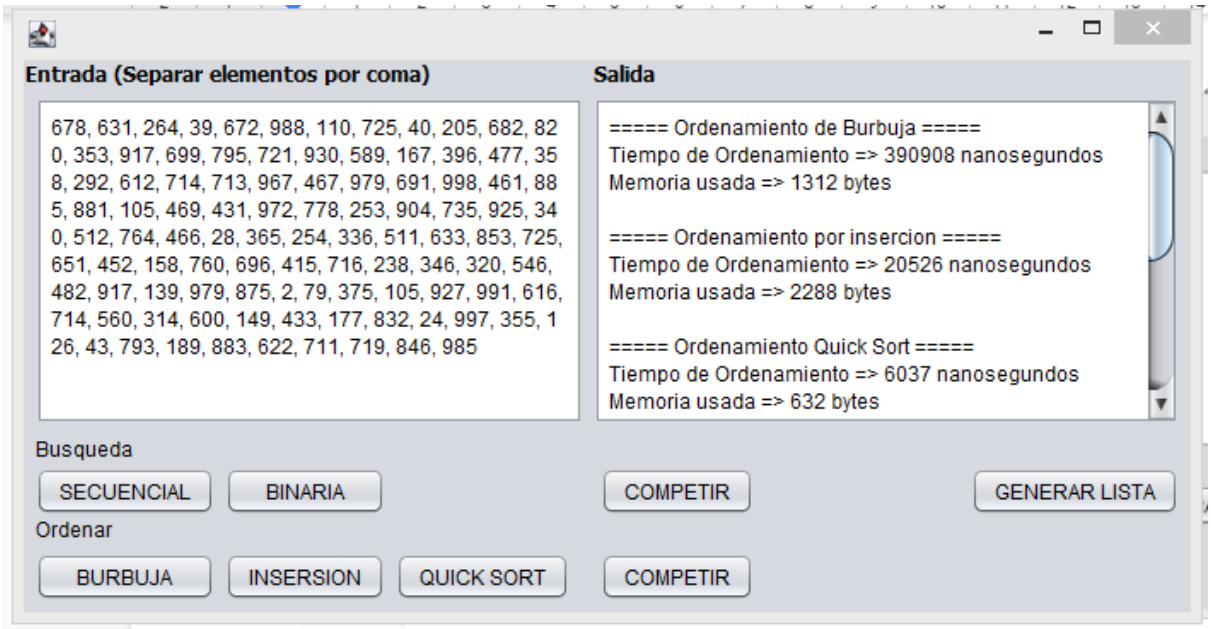




Ahora probamos con los métodos de ordenamiento



Resultados:



b. Link de Github y Ejecutable de la aplicación

Link: https://github.com/brayan-isaac/ProyectoFinalParalelos.git

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

Algoritmo	Tamaño de la lista	Tiempo
Secuencial	100 elementos	12,980 ns
Binaria	100 elementos	34,110 ns
Burbuja	100 elementos	390,908 ns
Inserción	100 elementos	20,526 ns
Quick Sort	100 elementos	6,037 ns
Secuencial	1000 elementos	37430 ns
Binaria	1000 elementos	9357 ns
Burbuja	1000 elementos	5689449 ns
Inserción	1000 elementos	2986901 ns
Quick Sort	1000 elementos	505917 ns

d. ¿Qué tanta memoria se consumió este proceso?

Algoritmo	Tamaño de la lista	Memoria
Secuencial	100 elementos	6,448 bytes
Binaria	100 elementos	3,440 bytes
Burbuja	100 elementos	1,312 bytes
Inserción	100 elementos	2,228 bytes

Quick Sort	100 elementos	632 bytes
Secuencial	1000 elementos	6144 bytes
Binaria	1000 elementos	3304 bytes
Burbuja	1000 elementos	2384 bytes
Inserción	1000 elementos	3344 bytes
Quick Sort	1000 elementos	2400 bytes

9. ¿Cuál fue el algoritmo que realizó la búsqueda y el ordenamiento más rápido? Explique.

En los algoritmos de **búsqueda** el que hace la búsqueda más rápida en la mayoría de las veces es **Binario** aunque existen casos donde el secuencial encuentra más rápido los elementos si se encuentran más cerca del inicio de la lista mientras el binario comienza a desplazarse desde el medio y debe realizar varios saltos para encontrarlo.

Mientras más largas son las listas mayor será el margen de velocidad del algoritmo **binario** con respecto al **secuencial** debido a que este puede ir reduciendo el número de comparaciones restantes a la mitad después de cada salto hasta encontrar el elemento.

En los algoritmos de ordenamiento el más rápido es el **quick Sort**, seguido por el método de **inserción** y al final el método de la **burbuja**.

El **método Quick** sort procede a descomponer la lista en varias partes y ordenarlas de forma atómica, luego las lista ordenadas comienzan a juntarse unas con otras usando una técnica externa semejante a la inserción hasta terminar con la lista completamente ordenada.

Conclusión

Después de realizar los ejercicios y diferentes técnicas de algoritmos para la búsqueda y ordenamiento de datos, podemos concluir que todos tienen un mismo objetivo relativamente pero lo que lo diferencian son sus procesos para realizar dicho objetivo, en este trabajo pudimos notar que el algoritmo que mejor rendimiento posee para la búsqueda de datos fue el Binario y para el ordenamiento Quicksort, esta conclusión permite tomar la decisión de cuál utilizar en un caso real, donde se utilizan grandes cantidades de datos y el rendimiento permite agilidad para obtener información a mejor tiempo.

También agregar que aprendimos técnicas para medir la eficiencia de nuestros algoritmos lo cual resulta útil para la optimización de código y la creación de programas más eficientes.

Bibliografía

http://unestudiantedeinformatica.blogspot.com/2014/07/medir-el-tiempo-de-ejecucion-en-java.html#:~:text=En%20java%20contamos%20con%20el,dar%C3%A1%20el%20tiempo%20de%20ejecuci%C3%B3n.

https://professor-falken.com/programacion/java/como-conocer-la-cantidad-de-memoria-total-usada-y-libre-en-java/

https://www.jerolba.com/midiendo-el-uso-real-de-memoria-jmnemohistosyne/

https://codigo--java.blogspot.com/2013/05/java-basico-021-ordenamiento-por.html

https://www.geeksforgeeks.org/quick-sort/