Revista Nexos Científicos
Julio-Diciembre 2022 pp. 01-11
Volumen 1, Número 2
Fecha de recepción: agosto 2022

ISSN: 2773-7489
Correo: editor@istvidanueva.edu.ec
URL: http://nexoscientificos.vidanueva.edu.ec/index.php/ojs/index
Fecha de aceptación: octubre 2022

Métodos de ordenamiento usando hilos en el entorno de desarrollo NETBEANS

Marta Salazar [1]; Fabiola Bucheli [2]

[1] Universidad Tecnológica Israel, Quito-Ecuador, msalazar@uisrael.edu.ec
[2] Universidad Técnica Particular de Loja, Quito-Ecuador, fgbl_02@hotmail.com

Resumen: Los algoritmos de ordenamiento nos permiten, como su nombre lo dice, ordenar. En este caso, se emplearon los métodos de ordenamiento Bubble Sort, Quick Sort y Shell Sort usando hilos en un ambiente gráfico en Java con el fin de analizar la cantidad de comparaciones que suceden y el tiempo que demora cada algoritmo en ordenar un arreglo numérico generado de manera aleatoria. Se seleccionaron estos métodos por ser muy populares y presentar características muy distintivas a la hora de ordenar. El método burbuja es un simple algoritmo de ordenamiento que emplea simples iteraciones para llevar a cabo el intercambio de dos elementos adyacentes y así colocarlos en el lugar correcto. Por otra parte, el método rápido funciona con la técnica de algoritmo de división y ganancia en la que un elemento fundamental se convierte en el punto focal de división alrededor del arreglo dado. Por último, el método de ordenamiento Shell el cual consiste en dividir el arreglo en bloques de varios elementos para organizarlos después por medio del ordenamiento de inserción directa. En el presente trabajo se conocerá el proceso empleado para llevar a cabo la comparación entre estos tres métodos los cuales tienen que ser dominados por los estudiantes de la carrera a la hora de darle solución a muchos de los problemas que son necesarios resolver para que ciertas máquinas y programas que utilizan estos algoritmos empiecen a funcionar en el momento y en la forma que se desea.

Palabras clave: Bubble Sort, Quick Sort, Hilos, Java

Sort methods using threads in the netbeans development environment

Abstract: Sorting algorithms allow us, as the name says, to sort. In this case, the Bubble Sort, Quick Sort and Shell Sort sorting methods were used using threads in a graphical environment in Java in order to analyze the number of comparisons that occur and the time it takes for each algorithm to sort a generated numerical array. randomly. These methods were selected because they are very popular and have very distinctive characteristics when ordering. The bubble method is a simple sorting algorithm that uses simple iterations to swap two adjacent elements to place them in the correct place. On the other hand, the fast method works with the split-and-gain algorithm technique in which a fundamental element becomes the focal point of splitting around the given array. Finally, the Shell ordering method which consists of dividing the array into blocks of several elements to organize them later by means of direct insertion ordering. In the present work, the process used to carry out the comparison between these three methods will be known, which have to be mastered by the students of the career when it comes to solving many of the problems that are necessary to solve so that certain machines and programs that use these algorithms start working when and in the way you want.

Keywords: Bubble Sort, Quick Sort, Threads, Java.

1. INTRODUCCIÓN

Hoy en día la programación es un elemento muy importante dentro del área de la Computación para desarrollar desde funciones muy sencillas hasta el desarrollo de juegos. Ordenar un conjunto de elementos de una lista es una tarea que se presenta con frecuencia ya que la información al estar ordenada nos servirá para poder encontrarla y acceder de manera más eficiente, porque de lo contrario se tendría que hacer de manera secuencial. A menudo, un ser humano puede realizar esta tarea

1. msalazar@uisrael.edu.ec
2. fgbl_02@hotmail.com

Revista Nexos Científicos
Julio-Diciembre 2022 pp. 01-11
Volumen 1, Número 2
Fecha de recepción: agosto 2022
ISSN: 2773-7489
Correo: editor@istvidanueva.edu.ec
URL: http://nexoscientificos.vidanueva.edu.ec/index.php/ojs/index
Fecha de aceptación: octubre 2022

de forma intuitiva. Sin embargo, un programa de computadora debe seguir una secuencia de instrucciones exactas para lograrlo. Esta secuencia de instrucciones se llama algoritmo. Existen varios algoritmos de ordenamiento y difieren en cuanto a su eficiencia y rendimiento. Algunos algoritmos importantes y conocidos son el ordenamiento de burbuja, ordenamiento por selección, por inserción y el ordenamiento rápido.

El presente trabajo surge de la necesidad de estudiar los métodos de ordenamiento: Bubble Sort, Quick Sort y Shell Sort. El propósito principal es hacer uso de estos algoritmos implementando hilos en el lenguaje de programación Java e identificar la importancia del uso de estos como parte esencial para resolver problemas. Además, se busca desarrollar el pensamiento lógico y capacidad de análisis, cualidades que son indispensables a la hora de programar y tener el mayor dominio posible dentro del área de desarrollo.

La investigación busca proporcionar información que será de utilidad para toda la comunidad educativa para así mejorar el conocimiento sobre el problema planteado. Además, es conveniente para afianzar un mayor conocimiento sobre estos métodos de ordenamiento ya que son una herramienta fundamental para ordenar vectores o matrices.

2. METODOLOGÍA

El presente trabajo "MÉTODOS DE ORDENAMIENTO USANDO HILOS EN EL ENTORNO DE DESARROLLO NETBEANS" corresponde a un proyecto en el cual se realizó una indagación bibliográfica, utilizando los descriptores: escritura científica, mapas conceptuales, lectura crítica, a través de la exploración en internet con el buscador "Google académico" con los términos del tema a investigar. Por otro lado, la investigación a realizarse es de tipo explicativa, por cuanto determinará las relaciones entre las variables, estableciendo los factores que pueden explicar los fenómenos que se estudian.

2.1.Planificación

Para desarrollar el proyecto se empleó el entorno de desarrollo integrado libre hecho principalmente para el lenguaje de programación Java, NetBeans, ya que puede ser usado para desarrollar cualquier tipo de aplicación, es gratis, permite el uso de herramientas para crear interfases gráficas y por ser el entorno de desarrollo con el que se trabaja en clases. Se emplearon los métodos de ordenamiento Bubble Sort, Quick Sort y Shell Sort usando hilos para poder cuantificar el tiempo y el número de intercambios que realiza cada uno a la hora de ordenar un arreglo numérico y así poder mostrar de manera grafica el proceso de ordenamiento que ocurre detrás del funcionamiento de estos tres algoritmos.

2.2.Diseño

En el apartado gráfico se realizó una interfaz amigable para el usuario. Se creó un entorno visual fácil de usar empleando botones que representan la información y acciones que se encuentran en la interfaz y un área de texto en donde visualizar todo el proceso de ordenamiento. JFrame es la ventana principal de nuestra aplicación gráfica y en ella podemos ver los típicos controles de una ventana de Windows como el de cerrar, maximizar o minimizar.

Ilustración 1 Interfaz Gráfica
Fuente: Elaborado por el autor

2.3.Desarrollo

Para desarrollar la aplicación se creó una clase llamada OrdenamientoConHilos. Se utilizó la palabra clave extends para indicar que la clase que se está definiendo se derivaría mediante herencia de JFrame una clase utilizada en Swing (biblioteca gráfica) para generar ventanas sobre las cuales añadir distintos objetos con los que podrá interactuar o no el usuario. Además se utilizó

Revista Nexos Científicos
Julio-Diciembre 2022 pp. 01-11
Volumen 1, Número 2
Fecha de recepción: agosto 2022
ISSN: 2773-7489
Correo: editor@istvidanueva.edu.ec
URL: http://nexoscientificos.vidanueva.edu.ec/index.php/ojs/index
Fecha de aceptación: octubre 2022

implements para implementar la interfaz ActionListener que se encuentra en el paquete java.awt.event y así crear un programa dinámico y permitir la interacción del usuario.

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.Random;
```

```
public class OrdenamientoConHilos extends JFrame implements ActionListener {
```

El nivel de acceso utilizado para las variables empleadas en el proyecto fue private reduciéndose así el nivel de visibilidad. Se indicó de esta manera que las variables sólo son accesibles por los métodos de la misma clase en la que se declararon.

```
    private JLabel Titulo;
    private JTextArea Area;
    private JScrollPane Cuadro;
    private JButton Crear, Ordenar, Desordenar, QuickSort;
    private JButton BubbleSort, ShellSort, Limpiar, Salir;
    private JButton Tiempo;
    private boolean PEjecutandose = false;
    private int[] ArregloOrdenado = new int[0];
    private int[] ArregloDesordenado = new int[0];
    private int[] ArregloNum = new int[0];
    private int Tamaño, Milisegundos = 0;
    private int Moda;
    private double Media, Mediana;
```

Las variables Moda, Media y Mediana se utilizaron en una funcionalidad extra del programa encargada de obtener el valor que aparece con mayor frecuencia, el promedio del conjunto de valores del arreglo y el valor que ocupa el lugar central de todos los datos.

Haciendo uso del arreglo color y empleando un código de selección aleatoria dentro del constructor de la clase OrdenamientoConHilos el programa es capaz de cambiar de fondo con cada ejecución añadiéndose de esta manera un efecto visual más llamativo.

```
private Color colorBg;
    private Color[] color = {
        new Color(255, 0, 0),
        new Color(36, 0, 255),
        new Color(255, 0, 246),
        new Color(0, 246, 255),
        new Color(48, 255, 0),
        new Color(240, 225, 0),
        new Color(255, 174, 0),
        new Color(0, 186, 255)
};
```

```
Random rn = new Random();
colorBg = color[rn.nextInt(color.length)];
getContentPane().setBackground(colorBg);
```

Ilustración 2 Cambio de fondo
Fuente: Elaborado por el autor

En el constructor se creó un objeto con el nombre V1 perteneciente a la clase Variables con el fin de crear métodos que se encarguen del color, la fuente y el fondo de los botones, el área de texto y la etiqueta. Además, se asociaron acciones con addActionListener a cada botón.

```
public OrdenamientoConHilos() {
        Random rn = new Random();
        colorBg = color[rn.nextInt(color.length)];
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        getContentPane().setBackground(colorBg);
        Color Negro = new Color(0, 0, 0);
        Color Blanco = new Color(255, 255, 255);
        Font Fuente = new Font("Arial", 0, 20);
        Font Fuente2 = new Font("Arial", 0, 40);
        setLayout(null);
        setTitle("Método de Ordenamiento");
        Variables V1 = new Variables();
        V1.Botones(Negro, Blanco, Fuente);
        V1.Etiquetas(Negro, Blanco, Fuente2);
        V1.Cuadro(Negro, Blanco, Fuente2);

        Crear.addActionListener(this);
        Desordenar.addActionListener(this);
        Ordenar.addActionListener(this);
        QuickSort.addActionListener(this);
        BubbleSort.addActionListener(this);
        ShellSort.addActionListener(this);
        Limpiar.addActionListener(this);
        Salir.addActionListener(this);
        Tiempo.addActionListener(this);
}
```

Revista Nexos Científicos
Julio-Diciembre 2022 pp. 01-11
Volumen 1, Número 2
Fecha de recepción: agosto 2022

ISSN: 2773-7489
Correo: editor@istvidanueva.edu.ec
URL: http://nexoscientificos.vidanueva.edu.ec/index.php/ojs/index
Fecha de aceptación: octubre 2022

La clase Variables es la encargada de la creación de los botones, etiquetas y el cuadro donde se muestra todo el proceso de ordenamiento del proyecto. La clase contiene métodos que reciben el color, la fuente y fondo de cada uno de los elementos que conforman la interfaz del programa.

```
private class Variables {...96 lines }
```

Método Botones:

```
public void Botones(Color Negro, Color Blanco, Font Fuente) {
        Fuente = new Font("Arial Black", 0, 15);
        Crear = new JButton("Crear");
        Crear.setBounds(10, 80, 142, 30);
        Crear.setBackground(Blanco);
        Crear.setForeground(Negro);
        Crear.setFont(Fuente);
        add(Crear);
```

Método Etiquetas:

```
public void Etiquetas(Color Negro, Color Blanco, Font Fuente2)
{
        Titulo = new JLabel("Método de Ordenamiento");
        Titulo.setBounds(170, 0, 550, 80);
        Titulo.setBackground(Blanco);
        Titulo.setForeground(Negro);
        Titulo.setFont(Fuente2);
        add(Titulo);
}
```

Método Cuadro:

```
public void Cuadro(Color Negro, Color Fondo, Font Fuente) {
        Fondo = new Color(0, 70, 255);
        Fuente = new Font("Arial Black", 0, 15);
        Area = new JTextArea();
        Cuadro = new JScrollPane(Area);
        Cuadro.setBounds(180, 80, 430, 430);
        Area.setLineWrap(true);
        Area.setWrapStyleWord(true);
        Area.setBackground(Fondo);
        Area.setForeground(Negro);
        Area.setFont(Fuente);
        add(Cuadro);
}
```

La interfaz ActionListener, solo tiene un método actionPerformed(ActionEvent) al que se llama cuando el usuario acaba de realizar una acción. En este caso este evento se invoca automáticamente cada vez que se hace clic en alguno de los botones. Dentro colocamos el objeto O1 de la clase Ordenamiento, los hilos 1, 2 y 3 y el código de ejecución de cada uno de los botones del programa

```
public void actionPerformed(ActionEvent A) {
      Ordenamiento O1 = new Ordenamiento();
      Hilo1 Hilo1 = new Hilo1();
      Hilo2 Hilo2 = new Hilo2();
      Hilo3 Hilo3 = new Hilo3();
      if (A.getSource() == Crear) {
         if (PEjecutandose == false) {
            MostrarDesordenado = "";
            //Tamaño = (int)(Math.random()*7+3);
            Tamaño = 10;
            ArregloOrdenado = new int[Tamaño];
            ArregloDesordenado = new int[Tamaño];
            O1.CrearArreglo(0);
            O1.MostrarDesordenado(0);
            Area.append("Tamaño: " + Tamaño +
"\nArreglo: " + MostrarDesordenado + "\n");
         } else {
            JOptionPane.showMessageDialog(null,
"ERROR!!! El hilo aún se ejecuta");
         }
      }
      if (A.getSource() == Desordenar) {
         if (PEjecutandose == false) {
            Area.append("\nArreglo: " +
MostrarDesordenado + "\n");
         } else {
            JOptionPane.showMessageDialog(null,
"ERROR!!! El hilo aún se ejecuta");
         }
      }
      if (A.getSource() == Ordenar) {
         if (PEjecutandose == false) {
            int B = Tamaño;
            MostrarOrdenado = "";
            O1.Bubble();
            O1.MostrarOrdenado(0);
            Area.append("\nArreglo ordenado: " +
MostrarOrdenado + "\n");
            O1.Estadisitica();
            Area.append("\nMedia: " + Media + "\n"
                  + "Moda: " + Moda + "\n"
                  + "Mediana: " + Mediana + "\n");
         } else {
            JOptionPane.showMessageDialog(null,
"ERROR!!! El hilo aún se ejecuta");
         }
      }
      if (A.getSource() == QuickSort) {
         if (PEjecutandose == false) {
            Hilo2.start();
         } else {
            JOptionPane.showMessageDialog(null,
"ERROR!!! El hilo aún se ejecuta");
         }
      }
      if (A.getSource() == BubbleSort) {
         if (PEjecutandose == false) {
            Hilo1.start();
         } else {
```

Revista Nexos Científicos
Julio-Diciembre 2022 pp. 01-11
Volumen 1, Número 2
Fecha de recepción: agosto 2022

ISSN: 2773-7489
Correo: editor@istvidanueva.edu.ec
URL: http://nexoscientificos.vidanueva.edu.ec/index.php/ojs/index
Fecha de aceptación: octubre 2022

```
            JOptionPane.showMessageDialog(null,
"ERROR!!! El hilo aún se ejecuta");
         }
      }
      if (A.getSource() == ShellSort) {
         if (PEjecutandose == false) {
            Hilo3.start();
         } else {
            JOptionPane.showMessageDialog(null,
"ERROR!!! El hilo aún se ejecuta");
         }
      }
      if (A.getSource() == Limpiar) {
         if (PEjecutandose == false) {
            Area.setText(null);
         } else {
            JOptionPane.showMessageDialog(null,
"ERROR!!! El hilo aún se ejecuta");
         }
      }
      if (A.getSource() == Tiempo) {
         if (PEjecutandose == false) {
            try {
               Milisegundos =
Integer.parseInt(JOptionPane.showInputDialog("Tiempo
de hilo actual: " + Milisegundos + "ml\n"
                     + "500 = 1/2 segundo\n"
                     + "1000 = 1 segundo\n"
                     + "10,000 = 10 segundos\n"
                     + "Introdusca un nuevo valor"));
               JOptionPane.showMessageDialog(null, "El
tiempo del hilo cambio a: " + Milisegundos + "ml");
            } catch (Exception e) {
               JOptionPane.showMessageDialog(null,
"ERROR!!!");
            }
         } else {
            JOptionPane.showMessageDialog(null,
"ERROR!!! El hilo aún se ejecuta");
         }
      }
      if (A.getSource() == Salir) {
         System.exit(0);
      }
   }
```

La clase Ordenamiento contendrá 5 métodos (Estadisitica, CrearArreglo, MostrarDesordenado, MostrarOrdenado y Bubble).

```
private class Ordenamiento {

    public void Estadisitica() {...56 lines }

    public void CrearArreglo(int i) {...8 lines }

    public void MostrarDesordenado(int i) {...6 lines }

    public void MostrarOrdenado(int i) {...6 lines }

    public void Bubble() {...12 lines }
}
```

El método Estadística se encargará de las medidas de tendencia central (media, moda, medina).

En la estadística, la moda es el valor que aparece con mayor frecuencia en un conjunto de datos. En el proyecto se empleó el siguiente código para determinar la moda de un arreglo numérico y así obtener aquel valor que presenta una mayor frecuencia absoluta dentro del arreglo numérico.

```
int numDistintos[] = new int[ArregloOrdenado.length];
         for (int i = 0; i < numDistintos.length; i++) {
            numDistintos[i] = -1;
         }
         int posND = 0;
         int posR = 0;
         while (posR < numDistintos.length) {
            int nr = ArregloOrdenado[posR];
            int numExt = 0;
            for (int i = 0; i < numDistintos.length; i++) {
               if (nr == numDistintos[i]) {
                  numExt++;
               }
            }
            if (numExt == 0) {
               numDistintos[posND] = nr;
               posND++;
            }
            posR++;
         }
         int contNum[] = new int[posND];
         for (int i = 0; i < posND; i++) {
            int ND = numDistintos[i];
            for (int j = 0; j < ArregloOrdenado.length; j++) {
               if (ND == ArregloOrdenado[j]) {
                  contNum[i]++;
               }
            }
         }
         int max = Integer.MIN_VALUE;
         int numEncontrado = -1;
         for (int i = 0; i < contNum.length; i++) {
            if (max < contNum[i]) {
               numEncontrado = numDistintos[i];
               max = contNum[i];
            }
```

Revista Nexos Científicos
Julio-Diciembre 2022 pp. 01-11
Volumen 1, Número 2
Fecha de recepción: agosto 2022

ISSN: 2773-7489
Correo: editor@istvidanueva.edu.ec
URL: http://nexoscientificos.vidanueva.edu.ec/index.php/ojs/index
Fecha de aceptación: octubre 2022

```
        }
        Moda = numEncontrado;
```

La media, también conocida como promedio, es el valor que se obtiene al dividir la suma de un conglomerado de números entre la cantidad de ellos.

```
for (int i = 0; i < ArregloOrdenado.length; i++) {
            Media += ArregloOrdenado[i];
        }
        Media = Media / ArregloOrdenado.length;
```

La mediana es el valor más importante cuando los datos tienen varios valores que se producen con frecuencia, y varios valores comparativamente altos. En el proyecto se empleó el siguiente código para determinar la mediana y así poder conocer el 50% de los datos que están por debajo de la Me, y el 50% por encima.

```
if (ArregloOrdenado.length == 1) {
            Mediana = ArregloOrdenado[0];
        } else {
            if (ArregloOrdenado.length % 2 == 0) {
                Mediana = (ArregloOrdenado[(ArregloOrdenado.length / 2)
- 1] + ArregloOrdenado[(ArregloOrdenado.length / 2)]) / 2;
            } else {
                Mediana = ArregloOrdenado[ArregloOrdenado.length / 2];
            }
```

El método CrearArreglo se ejecutará cuando se haga click en el botón Crear. Se generarán 10 números de manera aleatoria desde el 1 al 100 y se les asignarán a los arreglos: ArregloOrdenado y ArregloDesordenado los números aleatorios.

```
public void CrearArreglo(int i) {
        int Numero = (int) (Math.random() * 100 + 1);
        if (i < Tamaño) {
            ArregloDesordenado[i] = Numero;
            ArregloOrdenado[i] = Numero;
            CrearArreglo(i + 1);
        }
}
```

El método MostrarDesordenado se ejecutará cuando se haga click en el botón Desordenado y podremos ver el arreglo numérico en el mismo orden que se creó.

```
public void MostrarDesordenado(int i) {
        if (i < ArregloOrdenado.length) {
            MostrarDesordenado += "[" + ArregloDesordenado[i]
+ "]";
            MostrarDesordenado(i + 1);
        }
    }
```

Los métodos Bubble y MostrarOrdenado van de la mano ya que serán los encargados de ordenar el arreglo y mostrarlo. Se utilizará el método burbuja para realizar esta acción y el botón encargado de ejecutar estos métodos será Ordenar.

```
public void Bubble() {
        int aux;
        for (int i = 0; i < ArregloOrdenado.length - 1; i++) {
            for (int j = 0; j < ArregloOrdenado.length - 1; j++) {
                if (ArregloOrdenado[j] > ArregloOrdenado[j + 1]) {
                    aux = ArregloOrdenado[j];
                    ArregloOrdenado[j] = ArregloOrdenado[j + 1];
                    ArregloOrdenado[j + 1] = aux;
                }
            }
        }
}
public void MostrarOrdenado(int i) {
        if (i < ArregloOrdenado.length) {
            MostrarOrdenado += "[" + ArregloOrdenado[i] + "]";
            MostrarOrdenado(i + 1);
        }
}
```

El objetivo del proyecto es calcular el tiempo que tardan los métodos Quick Sort, Bubble Sort y Shell Sort en ordenan los arreglos. Para ello se utilizan tres hilos. La clase Hilo1 emplea el método Bubble Sort, la clase Hilo 2 el Quick Sort y por último la clase Hilo 3 se encarga del método Shell Sort.

Hilo1:

```
private class Hilo1 extends Thread {
    JScrollBar Desplazamiento =
Cuadro.getVerticalScrollBar();
    private int[] ArregloBubbleSort = new int[0];
    long Tinicio, Tfinal, Tiempo;
    private int contador;
```

```
@Override
public void run() {...28 lines }

public void Bubblesort() throws InterruptedException {...29 lines }
```

Hilo2:

Revista Nexos Científicos
Julio-Diciembre 2022 pp. 01-11
Volumen 1, Número 2
Fecha de recepción: agosto 2022

ISSN: 2773-7489
Correo: editor@istvidanueva.edu.ec
URL: http://nexoscientificos.vidanueva.edu.ec/index.php/ojs/index
Fecha de aceptación: octubre 2022

```
private class Hilo2 extends Thread {
      JScrollBar Desplazamiento =
Cuadro.getVerticalScrollBar();
      private int A = 0, B = Tamaño - 1;
      private int[] ArregloQuickSort = new int[0];
      long Tinicio, Tfinal, Tiempo;
      private int contador;
```

```
@Override
public void run() {...29 lines }

public void Quicksort(int A, int B) throws InterruptedException {...43 lines }
```

Hilo3:

```
private class Hilo3 extends Thread {
      JScrollBar Desplazamiento =
Cuadro.getVerticalScrollBar();
      private int[] ArregloShellSort = new int[0];
      long Tinicio, Tfinal, Tiempo;
      private int contador;
```

```
@Override
public void run() {...28 lines }

public void Shellsort() throws InterruptedException {...35 lines }
```

El método run () constituye el cuerpo de un hilo en ejecución. Este es el único método del interfaz Runnable y es llamado por el método start () después de que el hilo apropiado del sistema se haya inicializado.

Los métodos run de la clase Hilo 1, Hilo 2 e Hilo 3 comparten un código similar ya que ambos contabilizan el tiempo de ejecución de cada uno de los hilos y ejecutan los métodos de ordenamiento previamente mencionados.

```
@Override
      public void run() {
         PEjecutandose = true;
         Tinicio = System.currentTimeMillis();
         ArregloBubbleSort = new int[ArregloDesordenado.length];
         for (int i = 0; i < ArregloDesordenado.length; i++) {
            ArregloBubbleSort[i] = ArregloDesordenado[i];
         }
         try {
            Area.append("Arreglo Desordenado:\n"+ MostrarDesordenado
+"\n");
            Hilo1.sleep(Milisegundos);
            Bubblesort();
            Area.append("Arreglo Ordenado:\n");
            for (int i = 0; i < ArregloBubbleSort.length; i++) {
               Area.append("[" + ArregloBubbleSort[i] + "]");
               Desplazamiento.setValue(Desplazamiento.getMaximum());
            }
            Tfinal = System.currentTimeMillis();
            Area.append("\n\n");
            Hilo1.sleep(500);
            Desplazamiento.setValue(Desplazamiento.getMaximum());
            PEjecutandose = false;
         } catch (InterruptedException e) {
            System.out.println("ERROR!!! " + e);
         }
         Tiempo = (Tinicio - Tfinal) * -1;
         Area.append("tiempo de ejecucion: " + (Tiempo) + "ml\n\n");
}
```

Bubble Sort:

El ordenamiento de la burbuja o método de la burbuja es un algoritmo que consiste en comparar si el primer elemento del vector es mayor que el segundo, si esta condición se cumple entonces una variable temporal toma el valor del primer elemento del vector y el primer elemento del vector toma la posición del segundo elemento por último el segundo elemento toma el valor de la variable temporal, logrando con este método pasar el valor de la primera posición a la segunda posición. (Castañon, 2015)

```
public void Bubblesort() throws InterruptedException {
         contador = 1;
         int aux;
         for (int i = 0; i < ArregloBubbleSort.length - 1; i++) {
            for (int j = 0; j < ArregloBubbleSort.length - 1; j++) {
               if (ArregloBubbleSort[j] > ArregloBubbleSort[j +
1]) {
                  Area.append("PASADA: " + contador++ + " [" +
ArregloBubbleSort[j] + "] SI HAY INTERCAMBIO\n");
                  aux = ArregloBubbleSort[j];
                  ArregloBubbleSort[j] = ArregloBubbleSort[j + 1];
                  ArregloBubbleSort[j + 1] = aux;
                  for (int k = 0; k < ArregloBubbleSort.length; k++)
{
                     Area.append("[" + ArregloBubbleSort[k] + "]");

Desplazamiento.setValue(Desplazamiento.getMaximum());
                  }
                  Area.append("\n");
                  Hilo1.sleep(Milisegundos);
               } else {
                  Area.append("PASADA: " + contador++ + " [" +
ArregloBubbleSort[j] + "] NO HAY INTERCAMBIO\n");
                  for (int k = 0; k < ArregloBubbleSort.length; k++)
{
                     Area.append("[" + ArregloBubbleSort[k] + "]");

Desplazamiento.setValue(Desplazamiento.getMaximum());
                  }
                  Area.append("\n");
                  Hilo1.sleep(Milisegundos);
               }
            }
         }
      }
```

ISSN: 2773-7489
Correo: editor@istvidanueva.edu.ec
URL: http://nexoscientificos.vidanueva.edu.ec/index.php/ojs/index
Fecha de aceptación: octubre 2022

Ventajas

1. Este método es fácil de comprender, programar y es el más extendido.
2. Es bastante sencillo
3. En un código reducido se realiza el ordenamiento
4. Eficaz
5. Trabaja in situ.
6. Emplea operacionales en promedio para ordenar elementos.
7. Su bucle interno es extremadamente corto.

Desventajas

1. Es uno de los menos eficientes y por ello, normalmente, se aprende su técnica, pero no se utiliza.
2. Consume bastante tiempo de computadora
3. Requiere muchas lecturas/escrituras en memoria
4. Es recursivo y su implementación no recursiva es complicada.
5. Mal desempeño. (Blanco, 2007)

A pesar de que el ordenamiento de burbuja es uno de los algoritmos más sencillos de implementar, su orden O(n2) lo hace muy ineficiente para usar en listas que tengan más que un número reducido de elementos. Incluso entre los algoritmos de ordenamiento de orden O(n2), otros procedimientos como el Ordenamiento por inserción son considerados más eficientes. Dada su simplicidad, el ordenamiento de burbuja es utilizado para introducir el concepto de algoritmo, o de algoritmo de ordenamiento para estudiantes de ciencias de la computación. El ordenamiento de burbuja es asintóticamente equivalente, en tiempos de ejecución con el Ordenamiento por inserción en el peor de los casos, pero ambos algoritmos difieren principalmente en la cantidad de intercambios que son necesarios. (Abelardo, 2021)

Quick Sort:

El método Quick Sort es actualmente el más eficiente y veloz de los métodos de ordenación interna. Es también conocido con el nombre del método rápido y de ordenamiento por partición. Quicksort es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar n elementos en un tiempo proporcional a n log n.

Ventajas

1. Requiere de pocos recursos en comparación a otros métodos de ordenamiento.
2. En la mayoría de los casos, se requiere aproximadamente N log N operaciones.
3. Su ciclo interno es extremadamente corto.
4. No se requiere de espacio adicional durante ejecución (in-place processing).

Desventajas

1. Se complica la implementación si la recursión no es posible.
2. Un simple error en la implementación puede pasar sin detección, lo que provocaría un rendimiento pésimo.
3. No es útil para aplicaciones de entrada dinámica, donde se requiere reordenar una lista de elementos con nuevos valores.
4. Se pierde el orden relativo de elementos idénticos. (Alvarez, 2017)

El algoritmo trabaja de la siguiente forma:

1. Elegir un elemento de la lista de elementos a ordenar, al que llamaremos pivote.
2. Resituar 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.
3. La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
4. Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados. (Abelardo, 2021)

Revista Nexos Científicos
Julio-Diciembre 2022 pp. 01-11
Volumen 1, Número 2
Fecha de recepción: agosto 2022
ISSN: 2773-7489
Correo: editor@istvidanueva.edu.ec
URL: http://nexoscientificos.vidanueva.edu.ec/index.php/ojs/index
Fecha de aceptación: octubre 2022

```
    public void Quicksort(int A, int B) throws InterruptedException {
        contador++;
        int pivote = ArregloQuickSort[A];
        int i = A, j = B, auxiliar;
        while (i < j) {
            while (ArregloQuickSort[i] <= pivote && i < j) {
                Area.append("PASADA: " + contador + " [" +
ArregloQuickSort[A] + "] [" + pivote + " SI HAY INTERCAMBIO\n");
                for (int k = 0; k < ArregloQuickSort.length; k++) {
                    Area.append("[" + ArregloQuickSort[k] + "]");
Desplazamiento.setValue(Desplazamiento.getMaximum());
                }
                Area.append(" " + pivote);
                Area.append("\n");
                Hilo2.sleep(Milisegundos);
                i++;
            }
            while (ArregloQuickSort[j] > pivote) {
                Area.append("PASADA: " + contador + " [" +
ArregloQuickSort[B] + "] [" + pivote + " NO HAY INTERCAMBIO\n");
                for (int k = 0; k < ArregloQuickSort.length; k++) {
                    Area.append("[" + ArregloQuickSort[k] + "]");
Desplazamiento.setValue(Desplazamiento.getMaximum());
                }
                Area.append(" " + pivote);
                Area.append("\n");
                Hilo2.sleep(Milisegundos);
                j--;
            }
            if (i < j) {
                auxiliar = ArregloQuickSort[i];
                ArregloQuickSort[i] = ArregloQuickSort[j];
                ArregloQuickSort[j] = auxiliar;
            }
        }
        ArregloQuickSort[A] = ArregloQuickSort[j];
        ArregloQuickSort[j] = pivote;
        if (A < j - 1) {
            Quicksort(A, j - 1);
        }
        if (j + 1 < B) {
            Quicksort(j + 1, B);
        }
    }
```

Este tipo de ordenamiento es considerado como el mejor algoritmo de ordenamiento debido a su importante ventaja en términos de eficiencia es por ello que produce el método más efectivo y mayormente usado de ordenamiento para listas de cualquier tamaño.

Shell Sort:

El ordenamiento Shell es un algoritmo altamente eficiente basado en la comparación. En este algoritmo se comienzan a ordenar los elementos más lejanos y gradualmente se va reduciendo la distancia basándose en una secuencia para ordenar todo el array. Cualquier algoritmo de ordenación que intercambia elementos adyacentes (como los algoritmos burbuja, selección o inserción) tiene un tiempo promedio de ejecución de orden cuadrático (n^2). El método Shell mejora este tiempo comparando cada elemento con el que está a un cierto número de posiciones llamado salto, en lugar de compararlo con el que está justo a su lado. Este salto es constante, y su valor inicial es N/2 (siendo N el número de elementos, y siendo división entera). Se van dando pasadas con el mismo salto hasta que en una pasada no se intercambie ningún elemento de sitio. Entonces el salto se reduce a la mitad, y se vuelven a dar pasadas hasta que no se intercambie ningún elemento, y así sucesivamente hasta que el salto vale 1. (Hernández, 2015)

Ventajas

1. Es un algoritmo muy simple teniendo un tiempo de ejecución aceptable.
2. Es uno de los algoritmos más rápidos.
3. No requiere memoria adicional.
4. Fácil implementación.

Desventajas

1. Su complejidad es difícil de calcular y depende mucho de la secuencia de incrementos que utilice.
2. Shell Sort es un algoritmo no estable porque se puede perder el orden relativo inicial con facilidad.
3. Es menos eficiente que los métodos Merge, Heap y Quick Sort.
4. Realiza numerosas comparaciones e intercambios.

```
public void Shellsort() throws InterruptedException {
        int salto, aux, i;
        boolean cambios;
        for (salto = ArregloShellSort.length / 2; salto != 0; salto
/= 2) {
            cambios = true;
            while (cambios) {
                cambios = false;
                for (i = salto; i < ArregloShellSort.length; i++) {
                    if (ArregloShellSort[i - salto] >
ArregloShellSort[i]) {
                        Area.append("PASADA: " + contador++ + " ["
+ ArregloShellSort[i] + "] SI HAY INTERCAMBIO\n");
                        aux = ArregloShellSort[i];              // se
reordenan
                        ArregloShellSort[i] = ArregloShellSort[i -
salto];
                        ArregloShellSort[i - salto] = aux;
                        cambios = true;
                        for (int k = 0; k < ArregloShellSort.length; k++)
{
```

Revista Nexos Científicos
Julio-Diciembre 2022 pp. 01-11
Volumen 1, Número 2
Fecha de recepción: agosto 2022
ISSN: 2773-7489
Correo: editor@istvidanueva.edu.ec
URL: http://nexoscientificos.vidanueva.edu.ec/index.php/ojs/index
Fecha de aceptación: octubre 2022

```
                        Area.append("[" + ArregloShellSort[k] +
"]");
Desplazamiento.setValue(Desplazamiento.getMaximum());
                    }
                    Area.append("\n");
                    Hilo3.sleep(Milisegundos);
                } else {
                    Area.append("PASADA: " + contador++ + " ["
+ ArregloShellSort[i] + "] NO HAY INTERCAMBIO\n");
                    for (int k = 0; k < ArregloShellSort.length; k++)
{
                        Area.append("[" + ArregloShellSort[k] +
"]");
Desplazamiento.setValue(Desplazamiento.getMaximum());
                    }
                    Area.append("\n");
                    Hilo3.sleep(Milisegundos);
                }

            }
        }
    }
}
```

3. RESULTADOS Y DISCUSIÓN

La aplicación Métodos de Ordenamiento se utiliza para comparar los algoritmos Quick Sort, Bubble Sort y Shell Sort y de esta manera observar de una forma grafica el procedimiento de ordenamiento que cada uno de estos algoritmos lleva a cabo para cumplir su función. Tras el análisis del código y ejecución del programa es evidente como existen muchas diferencias entre los tres métodos analizados ya que pudieron observarse ventajas de unos sobre otros en la eficiencia en tiempo de ejecución.

3.1.Resultados.

Sobre la conceptualización de los métodos de ordenamiento se desarrollaron y construyeron ideas que permitieron generar un desarrollo lógico sobre estos algoritmos logrando con esto elaborar con mayor facilidad la comparación que posteriormente sería planteada.

Se realizó un programa en el entorno de desarrollo integrado libre NetBeans en donde gracias al uso de hilos se logró comparar el tiempo de ejecución de los métodos Bubble Sort, Quick Sort y Shell Sort. Con esto se comprobó lo importante que es seleccionar el método de ordenamiento adecuado ya que es de vital importancia que el algoritmo provea una solución eficiente y genere el menor de los costos de procesamiento posible.

Fueron explicadas las distintas aplicaciones que se le da a cada uno de los algoritmos y se determinó la importancia que desempeñan en el desarrollo de la Programación ya que ordenar un conjunto de elementos de una lista es una tarea que se presenta con frecuencia.

4. CONCLUSIONES

- Tener un mayor conocimiento de los métodos de ordenamiento es fundamental para resolver problemas en menos tiempo, mejorar el razonamiento y análisis lógico y ser más creativos. Cuanto más dominemos estos algoritmos más sabios seremos en el mundo de la programación ya que entenderemos lo que realmente hay detrás de muchos programas informáticos.
- El ordenamiento de burbuja tiene una complejidad $O(n^2)$. En este algoritmo el número de repeticiones depende de n (términos del vector) y no del orden de los términos, esto significa que, si pasamos al algoritmo una lista ya ordenada, realizará todas las comparaciones exactamente igual que para una lista no ordenada. Cuando una lista ya está ordenada, a diferencia de otros métodos que pasan por la lista una vez y encontraran que no hay necesidad de intercambiar las posiciones de los elementos, el método de ordenación por burbuja está forzado a pasar por dichas comparaciones, lo que hace que su complejidad sea cuadrática en el mejor de los casos. Esto lo cataloga como el algoritmo más ineficiente que existe, aunque para muchos programadores sea el más sencillo de implementar.
- Luego de analizarse el algoritmo Quick Sort se pudo comprobar que es uno de los mejores métodos de ordenación ya que a pesar de no ser tan sencillo su código tampoco fue tan complicado, resultando ser un algoritmo con una estructura elegante y con buena eficiencia. Con este método queda claro que en muchas ocasiones es mejor dividir para un óptimo desarrollo.

ISSN: 2773-7489
Correo: editor@istvidanueva.edu.ec
URL: http://nexoscientificos.vidanueva.edu.ec/index.php/ojs/index
Fecha de aceptación: octubre 2022

- El algoritmo Shell Sort es un algoritmo de ordenación interna muy sencillo pero muy ingenioso, basado en comparaciones e intercambios, y con unos resultados radicalmente mejores que los que se pueden obtener con el método de la burbuja.

REFERENCIAS

Abelardo. (7 de 1 de 2021). *blogspot*. Obtenido de blogspot: https://abelardo301504.blogspot.com/2021/01/metodos-de-ordenamiento.html

Alvarez, A. A. (7 de 10 de 2017). Recuperado el 10 de 1 de 2022, de https://quicksortweb.wordpress.com/2017/10/07/ventajas-desventajas-y-aplicaciones/#:~:text=Ventajas%2C%20Desventajas%20y%20Aplicaciones%20%E2%80%93%20Quicksort%20Ventajas%3A%20Requiere,log%20N%20operaciones.%20Ciclo%20interno%20es%20extremadamente%20corto

Blanco, O. (11 de 2007). Recuperado el 10 de 1 de 2022, de https://uneginginf05.es.tl/M-e2-todo-de-Ordenamiento-BURBUJA.htm

Castañon, M. (15 de 10 de 2015). *pseudocodigoejemplos*. Recuperado el 10 de 1 de 2022, de pseudocodigoejemplos: https://pseudocodigoejemplos.com/ordenamiento-burbuja-pseint/#:~:text=El%20ordenamiento%20de%20la%20burbuja%20o%20m%C3%A9todo%20de,y%20el%20primer%20elemento%20del%20vector%20toma%20la

Hernández, G. E. (15 de 8 de 2015). *blogspot*. Obtenido de blogspot: https://puntocomnoesunlenguaje.blogspot.com/2014/09/metodo-shell-de-ordenacion.html#:~:text=El%20m%C3%A9todo%20Shell%20de%20ordenaci%C3%B3n%20en%20Java%20para,%3B%20for%20%28i%20%3D%20salto%3B%20i%20%3C%20A.length%3B