

Esta interfaz dará a conocer lo que es método de ordenamiento Burbuja y la búsqueda Secuencial.

Creemos una clase de Ordenamiento burbuja donde vamos a comparar cada elemento del array utilizaremos un condicional de tipo for para comparar cada elemento del arreglo desde la posición cero hasta dicha cantidad va comparando cada intercambio de los elementos.

```
1 public class OrdenamientoBurbuja {
2     public static void ordenarBurbuja(int[] array, JTextArea intercambiosTextArea) {
3         int n = array.length;
4         boolean intercambio;
5
6         for (int i = 0; i < n - 1; i++) {
7             intercambio = false;
8             for (int j = 0; j < n - i - 1; j++) {
9                 if (array[j] > array[j + 1]) {
10                     // Intercambiar elementos
11                     int temp = array[j];
12                     array[j] = array[j + 1];
13                     array[j + 1] = temp;
14                     intercambio = true;
15
16                     // Mostrar intercambios en el JTextArea
17                     intercambiosTextArea.append(Arrays.toString(array) + "\n");
18                 }
19             }
20
21             // Si no hubo intercambios, el array ya está ordenado
22             if (!intercambio) {
23                 break;
24             }
25         }
26     }
27 }
```

```
public class OrdenamientoBurbuja {
    public static void ordenarBurbuja(int[] array, JTextArea
intercambiosTextArea) {
        int n = array.length;
        boolean intercambio;

        for (int i = 0; i < n - 1; i++) {
            intercambio = false;
            for (int j = 0; j < n - i - 1; j++) {
                if (array[j] > array[j + 1]) {
                    // Intercambiar elementos
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                    intercambio = true;

                    // Mostrar intercambios en el JTextArea
                    intercambiosTextArea.append(Arrays.toString(array) +
"\n");
                }
            }

            // Si no hubo intercambios, el array ya está ordenado
            if (!intercambio) {
                break;
            }
        }
    }
}
```

```

    }
  }
}

```

Este método toma el valor de un array y un valor buscado y devuelve la posición de valor buscado en el array 0 -1 no se encuentra

```

public class SecuencialSearch {
    public static int buscarSecuencial(int[] array, int valorBuscado) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == valorBuscado) {
                return i;
            }
        }
        return -1; // Valor no encontrado
    }
}

public class SecuencialSearch {
    // Este método toma un array y un valor a buscar y devuelve la
    // posición del valor buscado en el array o -1 si no se encuentra.
    public static int buscarSecuencial(int[] array, int valorBuscado) {
        // Se utiliza un bucle for para recorrer el array.
        for (int i = 0; i < array.length; i++) {
            // Se verifica si el elemento actual es igual al valor
            // buscado.
            if (array[i] == valorBuscado) {
                // Si se encuentra el valor, se devuelve la posición.
                return i;
            }
        }
        // Si el valor no se encuentra, se devuelve -1.
        return -1;
    }
}

```

Donde ingresaremos los datos y los valores de nuestras notas y nos ordenara comparando cada elemento y cambiara de posición hasta ordenar correctamente y el de la búsqueda secuencial busca el intermedio del arreglo si ya está ordenado y va comparando cada elemento hasta encontrar el valor correcto.

### Ordenamiento Quicksort

Este ordenamiento verifica los elementos de los array y verifica el índice del pivote y va comparando los elementos menores a la izquierda y los elemento mayores hacia la derecha.

```

public class OrdenamientoQuickSort {
    // Método principal quickSort para ordenar el array mediante el algoritmo
    // QuickSort
    public static void quickSort(int[] array, int inicio, int fin, JTextArea
    intercambiosTextArea) {
        // Verifica si hay más de un elemento en el array
    }
}

```

```

        if (inicio < fin) {
            // Obtiene el índice del pivote después de realizar la partición
            int indicePivote = particion(array, inicio, fin,
intercambiosTextArea);
            // Llama recursivamente al quickSort para la sublista izquierda
del pivote
            quickSort(array, inicio, indicePivote - 1, intercambiosTextArea);
            // Llama recursivamente al quickSort para la sublista derecha del
pivote
            quickSort(array, indicePivote + 1, fin, intercambiosTextArea);
        }
    }
}

// Declaración de la clase OrdenamientoQuickSort
public class OrdenamientoQuickSort {
    // Método principal quickSort para ordenar el array mediante el algoritmo QuickSort
    public static void quickSort(int[] array, int inicio, int fin, JTextArea intercambiosTextArea) {
        // Verifica si hay más de un elemento en el array
        if (inicio < fin) {
            // Obtiene el índice del pivote después de realizar la partición
            int indicePivote = particion(array, inicio, fin, intercambiosTextArea);
            // Llama recursivamente al quickSort para la sublista izquierda del pivote
            quickSort(array, inicio, indicePivote - 1, intercambiosTextArea);
            // Llama recursivamente al quickSort para la sublista derecha del pivote
            quickSort(array, indicePivote + 1, fin, intercambiosTextArea);
        }
    }
}

```

Este método dará a conocer las posiciones de los array y devolverá el índice del pivote va inicializando desde el menor elemento de cada pivote

```

// Método para realizar la partición del array y devolver el índice del
pivote
public static int particion(int[] array, int inicio, int fin, JTextArea
intercambiosTextArea) {
    // Elige el último elemento como pivote
    int pivote = array[fin];
    // Inicializa el índice del elemento menor
    int i = inicio - 1;
    // Itera a través de la sublista
    for (int j = inicio; j < fin; j++) {
        // Compara cada elemento con el pivote
        if (array[j] < pivote) {
            // Incrementa el índice del elemento menor y realiza el
intercambio
            i++;
            intercambiar(array, i, j, intercambiosTextArea);
        }
    }
    // Coloca el pivote en su posición correcta después de la partición
    intercambiar(array, i + 1, fin, intercambiosTextArea);
    // Devuelve el índice del pivote
    return i + 1;
}

```

```
// Método para realizar la partición del array y devolver el índice del pivote
public static int particion(int[] array, int inicio, int fin, JTextArea intercambiosTextArea) {
    // Elige el último elemento como pivote
    int pivote = array[fin];
    // Inicializa el índice del elemento menor
    int i = inicio - 1;
    // Itera a través de la sublista
    for (int j = inicio; j < fin; j++) {
        // Compara cada elemento con el pivote
        if (array[j] < pivote) {
            // Incrementa el índice del elemento menor y realiza el intercambio
            i++;
            intercambiar(array, i, j, intercambiosTextArea);
        }
    }
    // Coloca el pivote en su posición correcta después de la partición
    intercambiar(array, i + 1, fin, intercambiosTextArea);
    // Devuelve el índice del pivote
    return i + 1;
}
```

Mostrara los intercambios de los elementos del array y mostrara los intercambios de la caja de texto donde aparecerá las posiciones de los elementos.

```
// Método para intercambiar dos elementos en el array y mostrar el intercambio en el JTextArea
public static void intercambiar(int[] array, int i, int j, JTextArea intercambiosTextArea) {
    // Realiza el intercambio de elementos en el array
    int temp = array[i];
    array[i] = array[j];
    array[j] = temp;

    // Agrega la impresión del intercambio paso a paso en el JTextArea
    String intercambio = "Intercambio: " + array[i] + " <-> " + array[j]
+ "\n";
    intercambiosTextArea.append(intercambio);
}

}
```

```
// Método para intercambiar dos elementos en el array y mostrar el intercambio en el JTextArea
public static void intercambiar(int[] array, int i, int j, JTextArea intercambiosTextArea) {
    // Realiza el intercambio de elementos en el array
    int temp = array[i];
    array[i] = array[j];
    array[j] = temp;

    // Agrega la impresión del intercambio paso a paso en el JTextArea
    String intercambio = "Intercambio: " + array[i] + " <-> " + array[j] + "\n";
    intercambiosTextArea.append(intercambio);
}
```

Creamos una clase de Búsqueda Binaria donde vamos a buscar el índice izquierdo y derecho busca lo que es la búsqueda binaria si es menor o igual al índice derecho y sino es menor damos a conocer la posición medio es igual al valor buscado.

```
// Declaración de la clase BusquedaBinaria
public class BusquedaBinaria {
    // Método estático para realizar la búsqueda binaria en un array
    public static int buscar(int[] array, int valorBuscado) {
        // Inicialización de los índices izquierda y derecha para la búsqueda binaria
    }
```

```

    int izquierda = 0;
    int derecha = array.length - 1;
    // Bucle mientras el índice izquierdo sea menor o igual al índice
derecho
    while (izquierda <= derecha) {
        // Cálculo del índice medio para la búsqueda binaria
        int medio = izquierda + (derecha - izquierda) / 2;
        // Verificación si el valor en la posición media es igual al valor
buscado
        if (array[medio] == valorBuscado) {
            // Si es igual, se retorna el índice medio como resultado de
la búsqueda
            return medio;
        }
        // Si el valor en la posición media es menor que el valor buscado
        if (array[medio] < valorBuscado) {
            // Se actualiza el índice izquierdo para buscar en la mitad
derecha del array
            izquierda = medio + 1;
        } else {
            // Si el valor en la posición media es mayor que el valor
buscado
            // Se actualiza el índice derecho para buscar en la mitad
izquierda del array
            derecha = medio - 1;
        }
    }
    // Si no se encuentra el valor buscado, se retorna -1
    return -1;
}
}

```

Tendremos nuestro menú Principal donde vamos a tener dos botones para elegir cualquiera de los dos ordenamientos.

```

public class MenuPrincipal extends JFrame {

    public MenuPrincipal() {
        setTitle("Menu Principal de Estudiante");
        setSize(300, 150);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        // Crear panel con fondo amarillo
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        panel.setBackground(Color.YELLOW);

        // Crear label
        JLabel label = new JLabel("Menu de Registro de Estudiante");
        label.setFont(new Font("Arial", Font.BOLD, 14));
        label.setForeground(Color.BLUE);

        // Crear botones
        JButton btnBurbuja = new JButton("Burbuja");
        JButton btnQuickSort = new JButton("QuickSort"); // Agregado el botón
QuickSort
    }
}

```

```

// Agregar ActionListener a los botones
btnBurbuja.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // Lógica para el método de burbuja
        new InterfazOrbuja().setVisible(true);
    }
});

btnQuickSort.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // Abrir la interfaz de QuickSort al hacer clic en el botón
        new InterfazQuicksor().setVisible(true);
    }
});

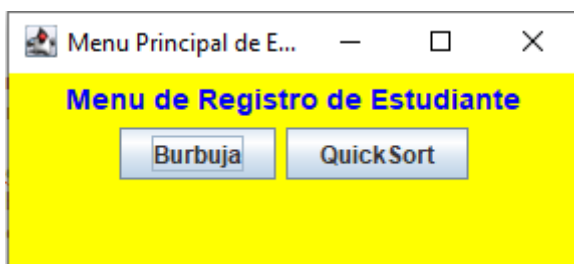
// Agregar label y botones al panel
panel.add(label);
panel.add(btnBurbuja);
panel.add(btnQuickSort); // Agregado el botón QuickSort

// Agregar panel al JFrame
add(panel);

// Hacer visible la interfaz
setVisible(true);
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            new MenuPrincipal();
        }
    });
}
}

```



Damos a conoce el la interfaz

```
import javax.swing.*;
```

```
import java.awt.*;
```

```
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;

import java.util.Vector;

import javax.swing.table.DefaultTableModel;

public class InterfazOrbuja extends JFrame {

    private JTextField nombreTextField;

    private JTextArea inputTextArea;

    private JTextArea intercambiosTextArea;

    private JTextArea outputTextArea;

    private JTable dataTable;

    private DefaultTableModel tableModel;

    private JTextField buscarTextField;

    private JTextArea resultadoBusquedaTextArea;

    public InterfazOrbuja() {

        setTitle("Principal");

        setSize(900, 600);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Cierre al hacer clic en la "X"

        setLocationRelativeTo(null); // Centrar la ventana en la pantalla

        // Configuración del color de fondo en color celeste

        getContentPane().setBackground(Color.CYAN);

        // Creación y configuración de componentes gráficos

        JPanel panel = new JPanel(null);

        // Configuración del color de fondo en color celeste

        panel.setBackground(Color.CYAN);

        JLabel titleLabel = new JLabel("Registro de Estudiante"); // Etiqueta para el título

        titleLabel.setBounds(400,10, 400, 30);

        titleLabel.setFont(new Font("Arial", Font.BOLD, 16)); // Estilo del título

        panel.add(titleLabel);
```

```
JLabel nombreLabel = new JLabel("Nombre:");  
nombreLabel.setBounds(10, 40, 100, 30);  
panel.add(nombreLabel);
```

```
nombreTextField = new JTextField();  
nombreTextField.setBounds(120, 40, 150, 30);  
panel.add(nombreTextField);
```

```
JLabel infoAria = new JLabel("Ingrese Notas");  
infoAria.setBounds(10, 80, 300, 30);  
panel.add(infoAria);
```

```
inputTextArea = new JTextArea();  
JScrollPane inputScrollPane = new JScrollPane(inputTextArea);  
inputScrollPane.setBounds(10, 110, 350, 70);  
panel.add(inputScrollPane);
```

```
JButton sortButton = new JButton("Ordenar");  
sortButton.setBounds(10, 200, 100, 30);  
sortButton.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        if (validarEntradaNumerica(inputTextArea.getText())) {  
            ordenarArray();  
        } else {  
            mostrarMensaje("Ingrese solo números en el área de texto.");  
        }  
    }  
});  
panel.add(sortButton);  
JLabel infoOrden = new JLabel("Notas en Cambio de Ordenados:");
```



```
infoOrden.setBounds(10, 250, 300, 30);  
panel.add(infoOrden);
```

```
intercambiosTextArea = new JTextArea();  
intercambiosTextArea.setEditable(false);  
JScrollPane intercambiosScrollPane = new JScrollPane(intercambiosTextArea);  
intercambiosScrollPane.setBounds(10, 280, 350, 100);  
panel.add(intercambiosScrollPane);
```

```
JLabel infoArreglo = new JLabel("Notas Ordenadas Realizados:");  
infoArreglo.setBounds(10, 370, 300, 30);  
panel.add(infoArreglo);
```

```
outputTextArea = new JTextArea();  
outputTextArea.setEditable(false);  
JScrollPane outputScrollPane = new JScrollPane(outputTextArea);  
outputScrollPane.setBounds(10, 400, 350, 50);  
panel.add(outputScrollPane);
```

```
JButton guardarButton = new JButton("Guardar");  
guardarButton.setBounds(130, 200, 100, 30);  
guardarButton.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        guardarDatos();  
    }  
});  
panel.add(guardarButton);
```

```
JButton eliminarButton = new JButton("Eliminar");  
eliminarButton.setBounds(250, 200, 100, 30);
```

```
eliminarButton.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        eliminarDatos();  
    }  
});  
panel.add(eliminarButton);
```

```
JLabel buscarLabel = new JLabel("Buscar:");  
buscarLabel.setBounds(10, 480, 50, 30);  
panel.add(buscarLabel);
```

```
buscarTextField = new JTextField();  
buscarTextField.setBounds(70, 480, 100, 30);  
panel.add(buscarTextField);
```

```
JButton buscarButton = new JButton("Buscar");  
buscarButton.setBounds(180, 480, 100, 30);  
buscarButton.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        buscarSecuencial();  
    }  
});  
panel.add(buscarButton);
```

```
resultadoBusquedaTextArea = new JTextArea();  
resultadoBusquedaTextArea.setEditable(false);  
JScrollPane resultadoBusquedaScrollPane = new JScrollPane(resultadoBusquedaTextArea);  
resultadoBusquedaScrollPane.setBounds(290, 480, 150, 30);  
panel.add(resultadoBusquedaScrollPane);
```

```

tableModel = new DefaultTableModel();
tableModel.addColumn("Nombre");
tableModel.addColumn("Información del Arreglo");
tableModel.addColumn("Numero buscado");
dataTable = new JTable(tableModel);
JScrollPane tableScrollPane = new JScrollPane(dataTable);
tableScrollPane.setBounds(400, 50, 430, 100);
panel.add(tableScrollPane);

setLayout(new BorderLayout());
add(panel, BorderLayout.CENTER);
}

private boolean validarEntradaNumerica(String entrada) {
    // Verificar si todos los caracteres son números o espacios
    return entrada.matches("^\\d+(\\s\\d+)*$");
}

// Método para mostrar un mensaje
private void mostrarMensaje(String mensaje) {
    JOptionPane.showMessageDialog(this, mensaje, "Advertencia",
JOptionPane.WARNING_MESSAGE);
}

private void guardarDatos() {
    // Obtener los datos de los campos
    String nombre = nombreTextField.getText();
    String arreglo = outputTextArea.getText();

    // Realizar la búsqueda
    buscarSecuencial();

    // Obtener el valor buscado

```

```
String valorBuscado = resultadoBusquedaTextArea.getText().replaceAll("\\D", ""); //
Obtener solo los dígitos
```

```
// Agregar datos a la tabla

Vector<String> rowData = new Vector<>();

rowData.add(nombre);

rowData.add(arreglo);

rowData.add(valorBuscado);

tableModel.addRow(rowData);

// Limpiar los campos después de guardar

nombreTextField.setText("");

inputTextArea.setText("");

outputTextArea.setText("");

buscarTextField.setText("");

resultadoBusquedaTextArea.setText("");

intercambiosTextArea.setText("");

}
```

```
private void eliminarDatos() {

    int selectedRow = dataTable.getSelectedRow();

    if (selectedRow != -1) {

        tableModel.removeRow(selectedRow);

    }

}
```

// Reemplaza el método ordenarArray con el siguiente código

```
private void ordenarArray() {

    String inputText = inputTextArea.getText();

    String[] elementos = inputText.split(" ");

    int[] array = new int[elementos.length];

    for (int i = 0; i < elementos.length; i++) {

        array[i] = Integer.parseInt(elementos[i]);

    }

}
```

```

    }

    // Llama al método de ordenamiento por burbuja con el JTextArea para mostrar
    intercambios

    OrdenamientoBurbuja.ordenarBurbuja(array, intercambiosTextArea);

    // Mostrar el array ordenado en el área de texto de salida

    StringBuilder outputText = new StringBuilder();

    for (int num : array) {

        outputText.append(num).append(" ");

    }

    outputTextArea.setText(outputText.toString());

}

```

```

private void buscarSecuencial() {

    String valorBuscar = buscarTextField.getText();

    if (!valorBuscar.isEmpty()) {

        int valorBuscado = Integer.parseInt(valorBuscar);

        int[] array = obtenerArrayOrdenado(); // Obtener el array ordenado

        // Utilizar la clase SecuencialSearch para realizar la búsqueda secuencial

        int indice = SecuencialSearch.buscarSecuencial(array, valorBuscado);

        // Mostrar el resultado de la búsqueda

        if (indice != -1) {

            resultadoBusquedaTextArea.setText("Encontrado en índice: " + indice);

        } else {

            resultadoBusquedaTextArea.setText("No encontrado");

        }

    }

}

```

```

    }
} else {
    resultadoBusquedaTextArea.setText("Ingrese un valor para buscar");
}
}

//metodo
private int[] obtenerArrayOrdenado() {
    String inputText = inputTextArea.getText();
    String[] elementos = inputText.split(" ");
    int[] array = new int[elementos.length];
    for (int i = 0; i < elementos.length; i++) {
        array[i] = Integer.parseInt(elementos[i]);
    }

    // Llama al método de ordenamiento por burbuja con el JTextArea para mostrar
    intercambios

    OrdenamientoBurbuja.ordenarBurbuja(array, intercambiosTextArea);

    return array;
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            new InterfazOrbujita().setVisible(true);
        }
    });
}
}

```

Principal

**Registro de Estudiante**

Nombre:

Ingrese Notas

Ordenar Guardar Eliminar

Notas en Cambio de Ordenados:

[8, 55, 77, 1]  
 [8, 55, 1, 77]  
 [8, 1, 55, 77]  
 [1, 8, 55, 77]  
 [8, 55, 77, 1]  
 [8, 55, 1, 77]

Notas Ordenadas Realizados:

Buscar:  Buscar Encontrado en índice: 2

Nombre	Información del Arreglo	Numero buscado
Kelvin	1 8 55 77	2

Aquí tenemos el ordenamiento de Quicksort y la búsqueda Binaria

```
import javax.swing.*.*;
import java.awt.*.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Vector;
import javax.swing.table.DefaultTableModel;

public class InterfazQuicksort extends JFrame {
    //creamos los JTextField, JTable, DefaultTableModel
    private JTextField nombreTextField;
    private JTextArea intercambiosTextArea;
    private JTextArea inputTextArea;
    private JTextArea outputTextArea;
    private JTable dataTable;
    private DefaultTableModel tableModel;

    private JTextField buscarTextField;
```

```
private JTextArea resultadoBusquedaTextArea;
```

```
//creamos el constructor
```

```
public InterfazQuicksor() {
```

```
    // Configuramos el JFrame
```

```
        setTitle("Registro Estudiante");
```

```
        setSize(900, 600);
```

```
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        setLocationRelativeTo(null);
```

```
    // Creamos y configuramos componentes gráficos
```

```
    JPanel panel = new JPanel(null);
```

```
    panel.setBackground(new Color(255, 182, 193)); // Color rosado
```

```
    JLabel tituloLabel = new JLabel("Registro Estudiante");
```

```
    tituloLabel.setBounds(10, 10, 200, 30);
```

```
    tituloLabel.setFont(new Font("Arial", Font.BOLD, 16));
```

```
    panel.add(tituloLabel);
```

```
    JLabel nombreLabel = new JLabel("Nombre:");
```

```
    nombreLabel.setBounds(10, 50, 100, 30);
```

```
    panel.add(nombreLabel);
```

```
    nombreTextField = new JTextField();
```

```
    nombreTextField.setBounds(120, 50, 240, 30);
```

```
    panel.add(nombreTextField);
```

```
    JLabel infoAria = new JLabel("Ingrese el dinero que ingresó toda la semana");
```

```
    infoAria.setBounds(10, 90, 300, 30);
```

```
    panel.add(infoAria);
```



```
inputTextArea = new JTextArea();

JScrollPane inputScrollPane = new JScrollPane(inputTextArea);
inputScrollPane.setBounds(10, 130, 350, 50);
panel.add(inputScrollPane);

JButton sortButton = new JButton("Ordenar");
sortButton.setBounds(10, 210, 100, 30);
sortButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (validarEntradaNumerica(inputTextArea.getText())) {
            ordenarArray();
        } else {
            mostrarMensaje("Ingrese solo números en el área de texto.");
        }
    }
});
panel.add(sortButton);
```

```
JLabel infoOrden = new JLabel("Información de Pasos Ordenados:");
infoOrden.setBounds(10, 240, 300, 30);
panel.add(infoOrden);
```

```
intercambiosTextArea = new JTextArea();
intercambiosTextArea.setEditable(false);
JScrollPane intercambiosScrollPane = new JScrollPane(intercambiosTextArea);
intercambiosScrollPane.setBounds(10, 270, 350, 100);
panel.add(intercambiosScrollPane);
```

```
JLabel infoArreglo = new JLabel("Ingresos Realizados:");
```

```
infoArreglo.setBounds(10, 410, 300, 30);
```

```
panel.add(infoArreglo);
```

```
outputTextArea = new JTextArea();
```

```
outputTextArea.setEditable(false);
```

```
JScrollPane outputScrollPane = new JScrollPane(outputTextArea);
```

```
outputScrollPane.setBounds(10, 440, 350, 50);
```

```
panel.add(outputScrollPane);
```

```
// JButton para guardar
```

```
JButton guardarButton = new JButton("Guardar");
```

```
guardarButton.setBounds(130, 210, 100, 30);
```

```
guardarButton.addActionListener(new ActionListener() {
```

```
    @Override
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        guardarDatos();
```

```
    }
```

```
});
```

```
panel.add(guardarButton);
```

```
// JButton para eliminar
```

```
JButton eliminarButton = new JButton("Eliminar");
```

```
eliminarButton.setBounds(250, 210, 100, 30);
```

```
eliminarButton.addActionListener(new ActionListener() {
```

```
    @Override
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        eliminarDatos();
```

```
    }
```

```

});

panel.add(eliminarButton);


// JButton para buscar con método secuencial
JLabel buscarLabel = new JLabel("Buscar:");
buscarLabel.setBounds(10, 500, 50, 30);
panel.add(buscarLabel);

buscarTextField = new JTextField();
buscarTextField.setBounds(70, 500, 100, 30);
panel.add(buscarTextField);

JButton buscarButton = new JButton("Buscar");
buscarButton.setBounds(180, 500, 100, 30);
buscarButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        busquedaBinariaResultado(null);
    }
});

panel.add(buscarButton);

resultadoBusquedaTextArea = new JTextArea();
resultadoBusquedaTextArea.setEditable(false);
JScrollPane resultadoBusquedaScrollPane = new JScrollPane(resultadoBusquedaTextArea);
resultadoBusquedaScrollPane.setBounds(290, 500, 150, 30);
panel.add(resultadoBusquedaScrollPane);

// Configuramos la JTable
tableModel = new DefaultTableModel();

```

```

tableModel.addColumn("Nombre");
tableModel.addColumn("Información del Arreglo");
tableModel.addColumn("Rango Buscado");
dataTable = new JTable(tableModel);
JScrollPane tableScrollPane = new JScrollPane(dataTable);
tableScrollPane.setBounds(400, 50, 430, 100);
panel.add(tableScrollPane);
setLayout(new BorderLayout());
add(panel, BorderLayout.CENTER);
}

```

```

private void guardarDatos() {
    // Obtener los datos de los campos
    String nombre = nombreTextField.getText();
    String arreglo = outputTextArea.getText();
    // Realizar la búsqueda
    busquedaBinariaResultado(null);
    // Obtener el valor buscado
    String valorBuscado = resultadoBusquedaTextArea.getText().replaceAll("\\D", ""); //
    Obtener solo los dígitos

    // Agregar datos a la tabla
    Vector<String> rowData = new Vector<>();
    rowData.add(nombre);

    rowData.add(arreglo);
    rowData.add(valorBuscado);
    tableModel.addRow(rowData);

    // Limpiar los campos después de guardar
    nombreTextField.setText("");
    inputTextArea.setText("");
}

```

```
outputTextArea.setText("");  
  
buscarTextField.setText("");  
  
resultadoBusquedaTextArea.setText("");  
  
intercambiosTextArea.setText("");  
  
}
```

```
private void eliminarDatos() {  
  
    int selectedRow = dataTable.getSelectedRow();  
  
    if (selectedRow != -1) {  
  
        tableModel.removeRow(selectedRow);  
  
    }  
}
```

```
// Método para validar si la entrada es numérica  
  
private boolean validarEntradaNumerica(String entrada) {  
  
    // Verificar si todos los caracteres son números o espacios  
  
    return entrada.matches("^\\d+(\\s\\d+)*$");  
  
}
```

```
// Método para mostrar un mensaje  
  
private void mostrarMensaje(String mensaje) {  
  
    JOptionPane.showMessageDialog(this, mensaje, "Advertencia",  
JOptionPane.WARNING_MESSAGE);  
  
}
```

```
// Método para ordenar el array utilizando el algoritmo QuickSort  
  
private void ordenarArray() {  
  
    // Obtener el texto de entrada del área de texto  
  
    String inputText = inputTextArea.getText();  
  
    // Dividir el texto en elementos separados por espacio
```

```

String[] elementos = inputText.split(" ");

// Crear un array de enteros y convertir cada elemento del texto a entero
int[] array = new int[elementos.length];
for (int i = 0; i < elementos.length; i++) {
    array[i] = Integer.parseInt(elementos[i]);
}

// Llamar al método QuickSort para ordenar el array
OrdenamientoQuickSort.quickSort(array, 0, array.length - 1, intercambiosTextArea);

// Construir una cadena con los elementos ordenados y mostrarla en el área de texto de
salida
StringBuilder outputText = new StringBuilder();
for (int num : array) {
    outputText.append(num).append(" ");
}
outputTextArea.setText(outputText.toString());
}

// Método para obtener el array ordenado
private int[] obtenerArrayOrdenado() {
    // Obtener el texto de entrada del área de texto
    String inputText = inputTextArea.getText();

    // Dividir el texto en elementos separados por espacio
    String[] elementos = inputText.split(" ");

    // Crear un array de enteros y convertir cada elemento del texto a entero
    int[] array = new int[elementos.length];
    for (int i = 0; i < elementos.length; i++) {
        array[i] = Integer.parseInt(elementos[i]);
    }

    // Llamar al método QuickSort para ordenar el array
    OrdenamientoQuickSort.quickSort(array, 0, array.length - 1, intercambiosTextArea);

    return array;
}

```

```
}
```

```
private void busquedaBinariaResultado(ActionEvent e) {  
    // Comienza el manejo del evento del botón de búsqueda  
    // Verifica si la entrada en el campo de búsqueda es numérica  
    if (validarEntradaNumerica(buscarTextField.getText())) {  
        // Convierte la entrada a un entero  
        int valorBuscado = Integer.parseInt(buscarTextField.getText());  
        // Obtiene un array ordenado para realizar la búsqueda binaria  
        int[] arrayOrdenado = obtenerArrayOrdenado();  
        // Utiliza la clase BusquedaBinaria para realizar la búsqueda binaria  
        int resultadoBusqueda = BusquedaBinaria.buscar(arrayOrdenado, valorBuscado);  
        // Verifica el resultado de la búsqueda  
        if (resultadoBusqueda != -1) {  
            // Muestra la posición si el valor es encontrado  
            resultadoBusquedaTextArea.setText("Encontrado en la posición: " +  
resultadoBusqueda);  
        } else {  
            // Muestra un mensaje si el valor no es encontrado en el array ordenado  
            resultadoBusquedaTextArea.setText("No encontrado en el array ordenado.");  
        }  
    } else {  
        // Muestra un mensaje de error si la entrada no es numérica  
        mostrarMensaje("Ingrese solo números en el campo de búsqueda.");  
    }  
}
```

```
public static void main(String[] args) {  
    SwingUtilities.invokeLater(new Runnable() {  
        @Override
```

```

public void run() {

    new InterfazQuicksor().setVisible(true);


}

});

}

}

```

 Registro Estudiante

### Registro Estudiante

Nombre:

Ingrese el dinero que ingresó toda la semana

Información de Pasos Ordenados:

Intercambio: 11 <-> 11  
 Intercambio: 1 <-> 55  
 Intercambio: 22 <-> 55  
 Intercambio: 1 <-> 11

Ingresos Realizados:

Buscar:

Nombre	Información del Arreglo	Rango Buscado
Peter	1 11 22 55	2