Informe de Práctica en Teoría 01

Tema: Algoritmos de Ordenamiento y Búsqueda en java

Nota

Estudiantes	Escuela	Asignatura
Eduardo Portugal eportugalpor@unsa.edu.pe Hernan Choquehuanca hchoquehuancaz@unsa.edu.pe Jhonatan Mamani jmamanices@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de Programación 2 Semestre: II Código: 1701213

Practica	Tema	Duración
01	Algoritmos de Ordenamiento y Búsqueda en java	5 días

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 25 Setiembre 2023	Al 4 Octubre 2023

1. Tarea

- Elaborar un proyecto utilizando git. donde se elabore un sistema para ingresar datos de alumnos universitarios. (Clase Student)
- El sistema debe almacenar los estudiantes en un Array. (Considerar leer archivos CSV).
- Implemente el algoritmo de ordenamiento por Inserción(Iterativo-Cuadrático) para ordenar el arreglo de estudiantes por diferentes parámetros. Ejemplo: Por apellido, paterno.
- Descubra cuál es el tiempo que se demora en las ejecuciones.
- Explique cualquier otro algoritmo de ordenamiento de complejidad logarítmica. e implemente el ordenamiento utilizando los mismo parámetros anteriores.
- Grafique los resultados de las simulaciones realizadas considerando como unidad de medida los nanosegundos. Desde n=1 alumno hasta n=N alumnos.
- Luego, para el arreglo ordenado implemente el algoritmo de búsqueda binaria iterativo/recursivo y grafique los resultados de sus simulaciones.

2. Equipos, materiales y temas utilizados

- Sistema Operativo Windows 11 Home Single Language 22H2 64 bits.
- Sistema Operativo Ubuntu versión 22.04.3 LTS
- Visual Studio Code 1.82.2.0.
- JDK 17 Full 64-Bits 17.0.7.7.
- Git 2.41.0.2.
- Cuenta en GitHub con el correo institucional.
- Arreglos Estándar y de Objetos.
- JFreeChart 1.5.3.
- JCommon 1.0.22.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/hernanchoquehuanca/fp2-23b.git`
- URL para el práctica 01 en el Repositorio GitHub.
- `https://github.com/hernanchoquehuanca/fp2-23b/tree/main/fase01/prac01`

4. Trabajo de Práctica 01

- A continuación se muestra el desarrollo del trabajo grupal de la práctica 01:

4.1. Creación de la clase Student y sus atributos

- Se crea la clase Student, que contendrá atributos de distintos tipos de variables:
  - String: email, nombre, apellido materno, apellido paterno.
  - boolean: género, estado.
  - int: CUI, fecha de nacimiento.

Listing 1: Atributos de la clase "Student"

```
1 public class Student{
2     private String email, name, lastNameF, lastNameM;
3     private boolean gender, status;
4     private int cui, dateBirth;
```

- Se agregó el método constructor, este método asigna los distintos datos recibidos sobre el estudiante haciendo uso de los setters, que con según un arreglo llamado data considera para cada índice un atributo.

Listing 2: Método constructor de la clase "Student"

```
public Student(String[] data){
    setCUI(data[0]);
    setEmail(data[1]);
    setName(data[2]);
    setLastNameF(data[3]);
    setLastNameM(data[4]);
    setDateBirth(data[5]);
    setGender(data[6]);
    setStatus(data[7]);
}
```

- Se implementaron los setters respetivos a cada atributo de la clase, teniendo en consideración el tipo de variable recibida.
- En los atributos del tipo String, sea el CUI, fecha de nacimiento, genero y estado se consideró el tipo de variable que se debe asignar.
  - cui: se asignaba el substring que contiene a los últimos 8 caracteres.
  - dateBirth: el entero asignado al atributo se componía de una cadena al que se extraía los valores númericos considerando sus posiciones (año, mes, día).
  - gender: al recibir tanto 1 (masculino) o 0 (femenino) como valor del género, asignando asi un booleano al atributo donde true representa el género masculino y false al femenino.
  - status: de manera similar en el estado se consideraba el 1 como activo y el 0 como inactivo, de esta manera se asignaba tanto true como false respectivamente al atributo.

Listing 3: Setters de la clase "Student"

```
public void setCUI(String c){
    cui = Integer.parseInt(c.substring(c.length()-8));
}
public void setEmail(String e){
    email = e;
}
public void setName(String n){
    name = n;
}
public void setLastNameF(String nf){
    lastNameF = nf;
}
public void setLastNameM(String nm){
    lastNameM = nm;
}
public void setDateBirth(String db){
    String cadena = db.substring(0, 4) + db.substring(5, 7) + db.substring(8);
    int date = Integer.parseInt(cadena);
    dateBirth = date;
}
public void setGender(String gn){
    if (gn.equals("1")){
        gender = true;
    } else {
        gender = false;
    }
```

```
27      }
28      public void setStatus(String st){
29          if(st.equals("1")){
30              status = true;
31          } else {
32              status = false;
33          }
34      }
```

- Como parte de la clase, también se agregaron los getters correspondientes a cada atributo.

Listing 4: Getters de la clase "Student"

```
1       public int getCUI(){
2           return cui;
3       }
4       public String getEmail(){
5           return email;
6       }
7       public String getName(){
8           return name;
9       }
10      public String getLastNameF(){
11          return lastNameF;
12      }
13      public String getLastNameM(){
14          return lastNameM;
15      }
16      public int getDateBirth(){
17          return dateBirth;
18      }
19      public boolean getGender(){
20          return gender;
21      }
22      public boolean getStatus(){
23          return status;
24      }
```

4.2. Lectura de archivos csv utilizando Bufferedreader y Bufferedwriter

- Se crea un BufferedReader llamado Lines para leer el archivo especificado (StudentData).
- Utiliza un bucle while para leer cada línea del archivo usando readLine().
- Incrementa numLine por cada línea leída, lo que resulta en el conteo total de líneas en el archivo.

Listing 5: Clase "Student"

```
80      public String toString(){
81          String birth = (dateBirth/10000) + "-" + ((dateBirth/100)%100) + "-" +
                (dateBirth%100);
82          String gn = gender?"Masculino":"Femenino";
83          String st = status?"activo":"inhabilitado";
```

```
84          return cui + "\t" + email + "\t" + name + "\t" + lastNameF + "\t" + lastNameM + "\t"
                + birth + "\t" + gn + "\t" + st;
85      }
```

Listing 6: Clase "StudentRegistration"

```
118     public static void sortingAlgorithms()throws IOException{
119         int numLine = 0;
120         String file = "StudentData.csv";
121         String line;
122         BufferedReader lines = new BufferedReader(new FileReader(file));
123         while (lines.readLine() != null) {
124             numLine++;
125         }
126         timesSaved = new double[7][numIntervals];
127         listado = new Student[numLine];
128         BufferedReader data = new BufferedReader(new FileReader(file));
129         int j = 0;
130         while ((line = data.readLine()) != null){
131             String[] dataS = line.split(",");
132             listado[j] = new Student(dataS);
133             j++;
134         }
```

4.3. Interfaz 1 - Recepción de algoritmos de ordenamiento a utilizar

- La primera interfaz tiene como fin interactuar con el usuario de manera que se seleccione el dato a ordenar como también los algoritmos que desee utilizar para luego mostrar la grafica con su comparación.

  - Primeramente se asigna tanto el tamaño como nombre de ventana y configurarla para cuando se realice el cierre de ventana.

```
1           JFrame ventana = new JFrame("Seleccin");
2           ventana.setSize(600, 250);
3           ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

  - Se crean 3 JPanel, que sirven para la organización tanto de espacios de elementos y botones.

```
38          JPanel panelPrincipal = new JPanel(new BorderLayout());
39          JPanel panelSuperior = new JPanel(new GridBagLayout());
40          JPanel panelBoton = new JPanel(new FlowLayout(FlowLayout.RIGHT));
```

  - Se configura las restricciones del diseño, para ordenarlos de manera que tenga una mejor apariencia.

```
42          GridBagConstraints gbc = new GridBagConstraints();
43          gbc.insets = new Insets(5, 5, 5, 5);
```

- Se crea las etiquetas y un JComboBox que contendrá los algoritmos de ordenamiento, de esta manera el usuario podrá marcar las que desee utilizar.

```
45      JLabel etiquetaDate = new JLabel("Seleccione una opcin:");
46      JComboBox<String> comboBoxDate = new JComboBox<>(date);
47      JLabel etiquetaAlgorithm = new JLabel("Seleccione algoritmos:");
```

- Se crean distintos checkBoxes utilizando el arreglo de algorithm e inicializandolos en false, ya que estos cambiarán si es que el usuario decide seleccionarlos.
- Posterior a ello utilizando ActionListener se obtienen las checkbox, que se actualizan en caso sea marcado o desmarcado (varie su valor booleano).

```
49      JCheckBox[] checkBoxes = new JCheckBox[algorithm.length];
50      for (int i = 0; i < algorithm.length; i++) {
51          checkBoxes[i] = new JCheckBox(algorithm[i]);
52          checkBoxes[i].setSelected(false);
53          checkBoxes[i].addActionListener(new ActionListener() {
54              public void actionPerformed(ActionEvent e) {
55                  JCheckBox checkBox = (JCheckBox) e.getSource();
56                  int index = Integer.parseInt(checkBox.getActionCommand());
57                  algorithms[index] = checkBox.isSelected();
58              }
59          });
60          checkBoxes[i].setActionCommand(Integer.toString(i));
61      }
```

- Se ubican los componentes que se encontrarán en el panerlSuperior.

```
63      gbc.gridx = 0;
64      gbc.gridy = 0;
65      gbc.anchor = GridBagConstraints.WEST;
66      panelSuperior.add(etiquetaDate, gbc);
67      gbc.gridx = 1;
68      gbc.gridy = 0;
69      panelSuperior.add(comboBoxDate, gbc);
70      gbc.gridx = 0;
71      gbc.gridy = 1;
72      panelSuperior.add(etiquetaAlgorithm, gbc);
```

- Creación de un panel que contendrá las JCheckBox y agregandolas al panelSuperior.

```
74      JPanel panelCheckBoxes = new JPanel(new GridLayout(0, 2));
75      for (int i = 0; i < algorithm.length; i++) {
76          panelCheckBoxes.add(checkBoxes[i]);
77      }
78      gbc.gridx = 1;
79      gbc.gridy = 1;
80      gbc.anchor = GridBagConstraints.EAST;
81      panelSuperior.add(panelCheckBoxes, gbc);
```

- Se hace un llamado al método sortingAlgorithms utilizando el ActionListener al interactuar con el botón creado.
- Además se toma en cuenta las excepciones que se produzcan durante la ejecución del código.

```
83      JButton boton1 = new JButton("Graficar");
84      boton1.setPreferredSize(new Dimension(80, 30));
85
86      boton1.addActionListener((ActionEvent x) -> {
87          //Llamando al metodo sortingAlgorithms para que ejecute los algoritmos de
                ordenamiento.
88          try {
89              orden = comboBoxDate.getSelectedIndex();
90              sortingAlgorithms();
91          } catch (IOException e) {
92              System.err.println("Se produjo un error de E/S: " + e.getMessage());
93          }
94
95      });
```

- Se declara e inicializa el boton2, que servirá para llamar al método con la segunda interfaz (Búsqueda).

```
97      JButton boton2 = new JButton("Buscar");
98      boton2.setPreferredSize(new Dimension(80, 30));
99
100     boton2.addActionListener((ActionEvent x) -> {
101         //Llama al metodo de la interfaz 2.
102         Interfaz2();
103         ventana.dispose();
104     });
```

- Finalmente se agregan los botones al panel que los contendrá.
- Se estructura la ventana principal colocando los paneles y sus ubicaciones.
- Se ubica y muestra la ventana de la interfaz.

```
106     panelBoton.add(boton1);
107     panelBoton.add(boton2);
108     panelPrincipal.add(panelSuperior, BorderLayout.CENTER);
109     panelPrincipal.add(panelBoton, BorderLayout.SOUTH);
110
111     ventana.add(panelPrincipal);
112     ventana.setLocationRelativeTo(null);
113     ventana.setVisible(true);
```

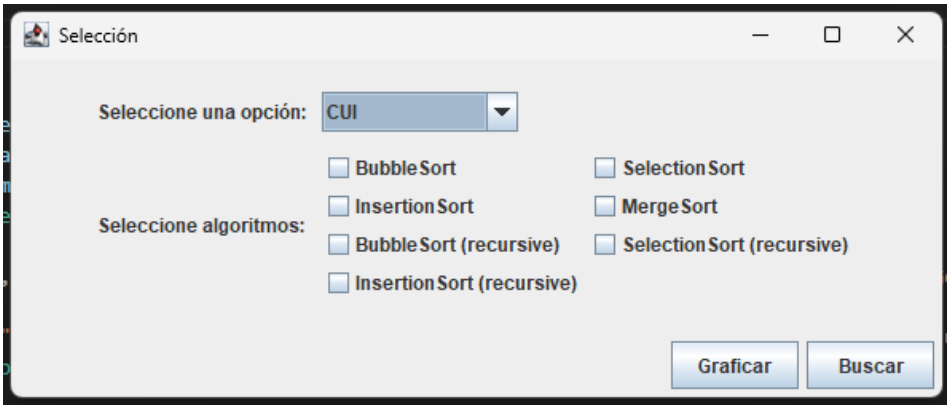

Figura 1: Interfaz 1

4.4. Interfaz 2 - Búsqueda Binaria de datos ordenados

- Esta segunda interfaz va dirigida a la interacción con el usario para realizar la búsqueda de un estudiante ingresando un dato específico.
- Dentro del método se utiliza el dato ingresaro, para luego según sea el tipo de dato recibido mostrarle los datos completos del estudiante en caso exista dentro del registro.
- Para realizar esta búsqueda se hace uso del algoritmo de ordenamiento MergeSort por su eficiencia, luego se invoca a los métodos tanto interativo como recursivo de la búsqueda binaria.
- Para este método se utilizó parte del código de la interfaz 1.
  - Primeramente se asigna tanto el tamaño como nombre de ventana y configurarla para cuando se realice el cierre de ventana.

```
544         JFrame ventana = new JFrame("Interfaz 2");
545         ventana.setSize(400, 200);
546         ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

  - Se crean 3 JPanel, que sirven para la organización tanto de espacios de elementos y botones.

```
548         JPanel panelPrincipal = new JPanel(new BorderLayout());
549         JPanel panelSuperior = new JPanel(new GridBagLayout());
550         JPanel panelBoton = new JPanel(new FlowLayout(FlowLayout.RIGHT));
```

  - Se configura las restricciones del diseño, para ordenarlos de manera que tenga una mejor apariencia.

```
552         GridBagConstraints gbc = new GridBagConstraints();
553         gbc.insets = new Insets(5, 5, 5, 5);
```

  - Creación de las etiquetas, comboBox para seleccionar el tipo de dato, y el campo para ingresar el texto del dato a buscar.

```
555         JLabel etiquetaDate = new JLabel("Seleccione una opcin:");
556         JComboBox<String> comboBoxDate = new JComboBox<>(date);
557 
558         JLabel etiquetaDato = new JLabel("Dato a buscar:");
559         JTextField campoDato = new JTextField(20);
```

- Creación del botón para la búsqueda.
- Configuración del ActionListener para el botón creado.
  - Primero se realiza el ordenamiento de los datos en el arreglo principal, según el atributo especificado, y se hace un llamado al método mergeSort
  - Se realiza un llamado, pero ahora a los algoritmos de búsqueda binaria iterativa y recursiva.
  - Mientras estas búsquedas son ejecutadas, se considera el tiempo y con los distintos tamaños de datos ya fijados, esto en ambos tipos búsqueda.
  - Finalmente muestra los resultados de la búsqueda si es que el estudiante con los datos recibidos existe, todo esto mediante una ventana emergente que se muestra luego de llamar al método crearVentana.
  - Por último muestra la gráfica de los tiempos tomados a través del método graphic.

```
561         JButton boton = new JButton("Buscar");
562         boton.setPreferredSize(new Dimension(80, 30));
563 
564         boton.addActionListener((ActionEvent x) -> {
565             //Se realiza el ordenamiento de los datos en el arreglo principal, segun el
                    atributo especificado.
566             int opcionSeleccionadaDate = comboBoxDate.getSelectedIndex();
567             datoAbuscar = campoDato.getText();
568             orden = opcionSeleccionadaDate;
569             mergeSort(listado, 0, listado.length-1);
570 
571             //Llama a los algoritmos de busqueda para encontrar el registro especificado.
572             int site = -1;
573             timeDataAB = new double[2][numIntervals];
574             int rangeIntervals = listado.length/numIntervals;
575             for (int i = 0; i < (numIntervals); i++){
576                 Student[] arr = new Student[rangeIntervals * (i + 1)];
577                 System.arraycopy(listado, 0, arr, 0, rangeIntervals * (i + 1));
578                 initialTime = System.nanoTime();
579                 iterativeBinarySearch(arr, datoAbuscar);
580                 currentTime = System.nanoTime();
581                 timesSearchBinary = currentTime - initialTime;
582                 timeDataAB[0][i] = timesSearchBinary;
583 
584                 Student[] arr1 = new Student[rangeIntervals * (i + 1)];
585                 System.arraycopy(listado, 0, arr1, 0, rangeIntervals * (i + 1));
586                 initialTime = System.nanoTime();
587                 site = binarySearchRecursive(arr1, 0, arr1.length-1, datoAbuscar);
588                 currentTime = System.nanoTime();
589                 timesSearchBinaryRc = currentTime - initialTime;
590                 timeDataAB[1][i] = timesSearchBinaryRc;
591             }
592 
```

```
593             String result;
594             if (site != -1){
595                 result = "Estudiante encontrado:<br>"+listado[site];
596             } else {
597                 result = "No existe el estudiante";
598             }
599 
600             SwingUtilities.invokeLater(() -> {
601                 crearVentana(result);
602             });
603             //Llama al metodo para graficar los tiempos de los algoritmos de busqueda.
604             graphic(2);
```

- Se configura la ubicación y ubicación de los componentes del diseño, todo esto dentro del panelSuperior

```
608         gbc.gridx = 0;
609         gbc.gridy = 0;
610         gbc.anchor = GridBagConstraints.WEST;
611         panelSuperior.add(etiquetaDate, gbc);
612         gbc.gridx = 1;
613         gbc.gridy = 0;
614         panelSuperior.add(comboBoxDate, gbc);
615         gbc.gridx = 0;
616         gbc.gridy = 1;
617         panelSuperior.add(etiquetaDato, gbc);
618         gbc.gridx = 1;
619         gbc.gridy = 1;
620         gbc.anchor = GridBagConstraints.EAST;
```

- Finalmente se agregan los botones y otros componentes al panel que los contendrá.
- Se estructura la ventana principal colocando los paneles y sus ubicaciones.
- Se ubica y muestra la ventana de la interfaz.

```
622         panelSuperior.add(campoDato, gbc);
623         panelBoton.add(boton);
624         panelPrincipal.add(panelSuperior, BorderLayout.CENTER);
625         panelPrincipal.add(panelBoton, BorderLayout.SOUTH);
626 
627         ventana.add(panelPrincipal);
628         ventana.setLocationRelativeTo(null);
629         ventana.setVisible(true);
```

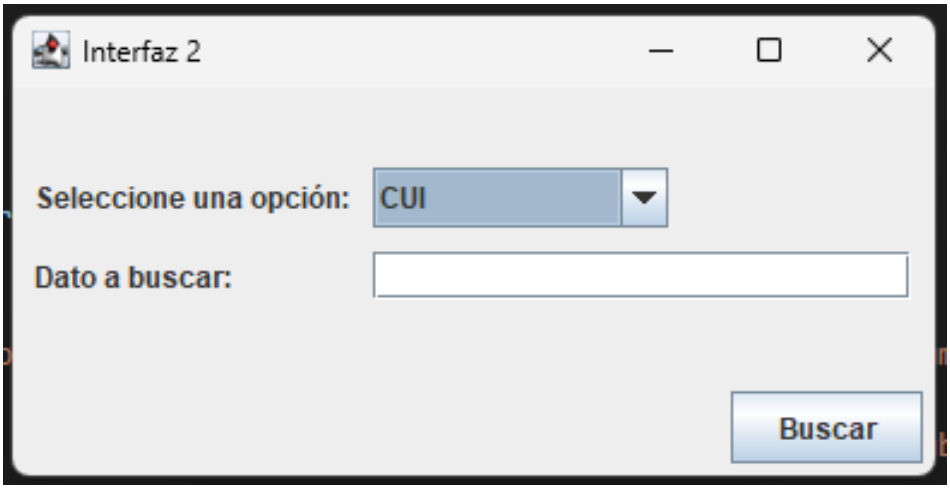

Figura 2: Interfaz 2

- Como fue mencionado antes, se utilizó un método llamado , el cual será explicado:
  - Se crea la ventana y sus características.
  - Se agrega un JLabel para mostrar el texto y se agrega a la ventana.
  - Por último es mostrado, actualizando el texto y usando formato en HTML.

```
633     public static void crearVentana(String result) {
634         JFrame ventana = new JFrame("Resultados de busqueda");
635         ventana.setSize(400, 150);
636         ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
637 
638         JLabel labelTexto = new JLabel("Texto a mostrar");
639 
640         ventana.add(labelTexto);
641         ventana.setVisible(true);
642 
643         SwingUtilities.invokeLater(new Runnable() {
644             public void run() {
645                 labelTexto.setText("<html>Tiempo de busqueda binaria<br>Busqueda iterativa:
                        "+timesSearchBinary
646                                     +"   Busqueda recursiva:
                                        "+timesSearchBinaryRc+"<br><br>Resultado:<br>"+result);
647             }
648         });
649     }
```

Figura 3: Ventana con los datos encontrados

4.5. Creación de los algoritmos de ordenamiento

- Para iniciar con la creación de los algoritmos se tuvo que definir qué datos se iban a comparar (CUI, email, nombres, etc.), para esto, en la selección de algoritmos se define a la variable 'orden' de la siguiente forma:

Listing 7: Obtención del valor de orden

```
406         boton1.addActionListener((ActionEvent x) -> {
407             //Llamando al metodo sortingAlgorithms para que ejecute los algoritmos de
                    ordenamiento.
408             try {
409                 orden = comboBoxDate.getSelectedIndex();
410                 sortingAlgorithms();
411             } catch (IOException e) {
412                 System.err.println("Se produjo un error de E/S: " + e.getMessage());
413             }
414 
415         });
```

- El evento (ActionEvent) se desencadena cuando ocurre la acción de hacer click en el botón.
- El TRY CATCH de la línea 408, se ejecutará cuando ocurra esto.
- Se obtiene el índice del elemento seleccionado en un ComboBox llamado comboBoxDate y se asigna a la variable 'orden'.
- Con este dato guardado, se procede a llamar a la ejecución de los algoritmos de ordenamiento.
- Finalmente, se capturan posibles excepciones de tipo IOException que puedan ocurrir al llamar al método sortingAlgorithms() y en caso de existir, se imprime un mensaje de error.

4.5.1. Método comparison() para el ordenamiento

Listing 8: Creación del método para comparar en el ordenamiento

```
406     public static int comparison(Student st1, Student st2, int orden){
407         int result = 0;
408         switch (orden){
409             case 0:
```

```
410             if(st1.getCUI() > st2.getCUI()) result = 1;
411             break;
412         case 1:
413             result = st1.getEmail().compareTo(st2.getEmail());
414             break;
415         case 2:
416             result = st1.getName().compareTo(st2.getName());
417             break;
418         case 3:
419             result = st1.getLastNameF().compareTo(st2.getLastNameF());
420             break;
421         case 4:
422             result = st1.getLastNameM().compareTo(st2.getLastNameM());
423             break;
424         case 5:
425             if(st1.getDateBirth() > st2.getDateBirth()) result = 1;
426             break;
427         case 6:
428             result = (st1.getGender()?"a":"z").compareTo(st2.getGender()?"a":"z");
429             break;
430         case 7:
431             result = (st1.getStatus()?"a":"z").compareTo(st2.getStatus()?"a":"z");
                    //cambiar
432             break;
433     }
434     return result;
435 }
```

- Este método fue creado para comparar dos objetos de la clase Student basándose en el criterio especificado previamente por el parámetro orden (tipo de dato a comparar).
- Como se ve en los casos, se comparan los valores de cada uno de los tipos de datos de los estudiantes.
- Se retorna el resultado de la comparación (result), que puede tener un valor menor, mayor o igual a 0. Menor a 0, si el primer objeto resulta de menor valor que el segundo, mayor a 0 en caso contrario y 0 si ambos tienen el mismo valor.

4.5.2. Algoritmo Bubble Sort

Listing 9: Creación del algoritmo Bubble Sort iterativo

```
222 public static void bubbleSortIterative(Student[] arr){
223     for (int i = 1; i < arr.length; i++){
224         for (int j = 0; j < arr.length - 1; j++){
225             if (comparison(arr[j], arr[j + 1], orden) > 0){
226                 Student temp;
227                 temp = arr[j];
228                 arr[j] = arr[j + 1];
229                 arr[j + 1] = temp;
230             }
231         }
232     }
233 }
```

- Se toma como único argumento el arreglo de objetos Student (arr) que se va a ordenar.
- Con el bucle FOR exterior de la línea 223, se itera sobre el arreglo desde el segundo elemento (i = 1) hasta el último elemento (i = arr.length - 1).
- El bucle FOR interior de la línea 224, itera sobre el arreglo desde el primer elemento hasta el penúltimo elemento. Será el encargado de recorrer los elementos para intercambiar si se cumple la condición de IF.
- La condicional IF en la línea 225, compara dos elementos adyacentes en el arreglo utilizando la función comparison(), el de la posición actual del bucle interior y el que le sigue, de ser mayor el primero, deberán intercambiarse.

4.5.3. Algoritmo Bubble Sort recursivo

Listing 10: Creación del algoritmo Bubble Sort recursivo

```
236 public static void recursiveBubbleSort(Student arr[], int n){
237     if (n == 1)
238         return;
239 
240     int count = 0;
241 
242     for (int i = 0; i < n - 1; i++) {
243         if (comparison(arr[i], arr[i + 1], orden) > 0) {
244             Student temp = arr[i];
245             arr[i] = arr[i + 1];
246             arr[i + 1] = temp;
247             count = count + 1;
248         }
249     }
250     if (count == 0)
251         return;
252     recursiveBubbleSort(arr, n - 1);
253 }
```

- En este método recursivo se recibe el arreglo arr y la longitud del mismo.
- El primer caso condicional del IF en la líne 237 presenta el caso base de la recursión. Ya que, cuando n es igual a 1, la recursión se detiene y el arreglo se considera ordenado.
- Se inicializa un contador 'count' para mantener un seguimiento de los intercambios realizados en cada iteración.
- El bucle FOR de la línea 242 compara cada elemento con su siguiente en el arreglo y realiza un intercambio como en el caso del iterativo utilizando el método comparison() en el IF de la línea 243.
- En la línea 250, el IF indica que si no se realizó ningún intercambio en la iteración actual, el arreglo ya está ordenado y se cierra el método recursivo.
- Si existe intercambios, se llama recursivamente al mismo método con n reducido en 1 para los elementos restantes.

4.5.4. Algoritmo Selection Sort

Listing 11: Creación del algoritmo Selection Sort iterativo

```
256 public static void iterativeSelectionSort(Student[] arr){
257     int n = arr.length;
258     for (int i = 0; i < n-1; i++){
259         int min_idx = i;
260         for (int j = i + 1; j < n; j++)
261             if (comparison(arr[min_idx], arr[j], orden) > 0)
262                 min_idx = j;
263         Student temp = arr[min_idx];
264         arr[min_idx] = arr[i];
265         arr[i] = temp;
266     }
267 }
```

- Se recibe únicamente el arreglo de objetos Student (arr) que se va a ordenar.
- En la primera linea dentro del método, se obtiene la longitud del arreglo arr y se almacena en la variable n.
- En primer lugar, se inicializa como el índice del elemento más pequeño al valor de i.
- Este elemento se evalúa con el bucle FOR en la línea 256 y el IF integrado para buscar en la parte del arreglo que aún no se ha ordenado, el elemento con el menor valor.
- Tras haber encontrado el elemento más pequeño en el índice "min-idx", se intercambia con el primer elemento aún no ordenado del arreglo que va reduciendose hasta haber comparado todo.

4.5.5. Algoritmo Selection Sort recursivo

- Se crea un método de apoyo llamado "minIndex":

Listing 12: Método de apoyo minIndex

```
282 public static int minIndex(Student arr[], int i, int j) {
283     if (i == j)
284         return i;
285     int k = minIndex(arr, i + 1, j);
286 
287     if (comparison(arr[i], arr[k], orden) < 0 || comparison(arr[i], arr[k], orden) != 1) {
288         return i;
289     } else {
290         return k;
291     }
292 }
```

- Este método ayudará a encontrar el índice del elemento mínimo en un rango dado (i a j) del arreglo arr ingresado de forma recursiva.
- La condición IF de la línea 283, devuelve el índice i si es igual al valor del índice j.

Listing 13: Creación del algoritmo Selection Sort recursivo

```
270 public static void recursiveSelectionSort(Student[] arr, int n, int idx) {
271     if (idx == n)
272       return;
273 
274     int k = minIndex(arr, idx, n - 1);
275     if (k != idx){
276       Student temp = arr[k];
277       arr[k] = arr[idx];
278       arr[idx] = temp;
279     }
280     recursiveSelectionSort(arr, n, idx + 1);
281 }
```

- En el método de ordenación principal, se toma como argumentos el arreglo arr, su longitud, y el índice idx para controlar la posición actual durante el proceso de selección.
- La primera condición en el IF de la línea 271, evalúa si el index del elemento es igual al tamaño del arreglo para finalizar su ejecución.
- En la línea 274, el valor de k, llama a la función minIndex() para encontrar el índice del elemento mínimo desde la posición idx hasta el final del arreglo (n - 1).Al finalizar esa recursión, k es el índice del elemento mínimo encontrado.
- Al igual que el iterativo, compara los elementos del mínimo y la posición inicial de la parte desordenada para intercambiarlos si no son iguales haciendo la comparación con el método comparison().
- Finalmente, el proceso recursivo continúa, mientras se excluye el elemento más pequeño que ya está en su posición correcta modificando el index como: idx + 1.

4.5.6. Algoritmo Insertion Sort

Listing 14: Creación del algoritmo Insertion Sort iterativo

```
295 public static void iterativeInsertionSort(Student[] arr){
296     int j;
297     Student key;
298     for (int i = 1; i < arr.length; i++) {
299         key = arr[i];
300         j = i - 1;
301         while (j >= 0 && comparison(arr[j], key, orden) > 0) {
302             arr[j + 1] = arr[j];
303             j--;
304         }
305         arr[j + 1] = key;
306     }
307 }
```

- El único argumento recibido es el arreglo de los objetos que se va a ordenar.
- En las líneas siguientes, se declara una variable j para representar una posición en el arreglo y una variable key para almacenar temporalmente un elemento durante el proceso de ordenamiento.

- Se declara el elemento en la posición i del arreglo como la clave para comparar
- El bucle WHILE de la línea 301, compara el elemento clave con los elementos anteriores a él (posición j) para encontrar su posición adecuada. Mientras arr[j] es mayor que el elemento la clave, se desplaza y se reduce el valor de j para continuar la comparación hacia atrás.
- Finalmente, la clave se inserta en la posición correcta (j + 1) y se continúa con el ciclo.

4.5.7. Algoritmo Insertion Sort recursivo

Listing 15: Creación del algoritmo Insertion Sort recursivo

```
310 private static void recursiveInsertionSort(Student[] arr, int n) {
311     if (n <= 1)
312         return;
313
314     recursiveInsertionSort(arr, n - 1);
315
316     Student key = arr[n - 1];
317     int j = n - 2;
318     while (j >= 0 && comparison(arr[j], key, orden) > 0) {
319         arr[j + 1] = arr[j];
320         j--;
321     }
322     arr[j + 1] = key;
323 }
```

- En este método se toma como argumentos el arreglo de objetos Student arr y su longitud n.
- Se crea una condición con un IF en la línea 311 para acabar el método si queda un elemento o si no hay ninguno, ya que se consideraría ya ordenado.
- El método recursivo ordena los primeros n-1 elementos.
- Se toma como clave (key) al último elemento del arreglo, posteriormente, inicializamos j con n - 2 para asegurar que apunte al último elemento de la parte ordenada.
- El bucle WHILE de la línea 318, compara el elemento clave con los elementos anteriores a él (en posición j) para encontrar su posición adecuada.
- Finalmente, una vez encuentra su ubicación, se inserta en la posición correcta de su orden.

4.5.8. Algoritmo Merge Sort

- Se crea un primer método "merge"que recibe el arreglo de objetos Student (arr), los índices de los extremos izquierdo (l) y derecho (r) de un subarreglo y un índice intermedio (m).

Listing 16: Creación del método Merge

```
335 public static void merge(Student[] arr, int l, int m, int r) {
336     int n1 = m - l + 1;
337     int n2 = r - m;
338     Student L[] = new Student[n1];
339     Student R[] = new Student[n2];
340
341     for (int i = 0; i < n1; ++i)
```

```
342             L[i] = arr[l + i];
343         for (int j = 0; j < n2; ++j)
344             R[j] = arr[m + 1 + j];
345 
346         int i = 0, j = 0;
347         int k = l;
348 
349         while (i < n1 && j < n2) {
350             if (comparison(L[i], R[j], orden) <= 0) {
351                 arr[k] = L[i];
352                 i++;
353             }
354             else {
355                 arr[k] = R[j];
356                 j++;
357             }
358             k++;
359         }
360         while (i < n1) {
361             arr[k] = L[i];
362             i++;
363             k++;
364         }
365         while (j < n2) {
366             arr[k] = R[j];
367             j++;
368             k++;
369         }
370     }
```

- Se inicializa n1 y n2. n1 representará la cantidad de elementos en el primer subarreglo (L) y n2 representará la cantidad de elementos en el segundo subarreglo (R).
- Posteriormente, divide el arreglo en dos subarreglos ordenados (L y R) en base a la posición intermedia m. Combina los subarreglos ordenados L y R en el arreglo principal arr.

Listing 17: Creación del método Merge Sort

```
326     public static void mergeSort(Student arr[], int l, int r) {
327         if (l < r){
328             int m = l + (r - l) / 2;
329             mergeSort(arr, l, m);
330             mergeSort(arr, m + 1, r);
331             merge(arr, l, m, r);
332         }
333     }
```

- Toma tres parámetros: el arreglo de objetos Student (arr), y los índices de los extremos izquierdo (l) y derecho (r) del subarreglo que debe ordenarse.
- Divide recursivamente el arreglo en subarreglos más pequeños hasta que los subarreglos contengan un solo elemento. Entre las líneas 329 y 331 llama a la función merge para combinar y ordenar los subarreglos.

4.6. Creación de los algoritmos de búsqueda

4.6.1. Método comparisonSearch para la búsqueda binaria

Listing 18: Creación de método para comparar en la búsqueda

```
438 public static int comparisonSearch(Student st, String data){
439     int result = 0;
440     switch (orden){
441         case 0:
442             if(st.getCUI() > Integer.parseInt(data)){
443                 result = 1;
444             } else if (st.getCUI() < Integer.parseInt(data)){
445                 result = -1;
446             }
447             break;
448         case 1:
449             result = st.getEmail().compareTo(data);
450             break;
451         case 2:
452             result = st.getName().compareTo(data);
453             break;
454         case 3:
455             result = st.getLastNameF().compareTo(data);
456             break;
457         case 4:
458             result = st.getLastNameM().compareTo(data);
459             break;
460         case 5:
461             String cadena = data.substring(0, 4) + data.substring(5, 7) +
                     data.substring(8);
462             if(st.getDateBirth() > Integer.parseInt(cadena)){
463                 result = 1;
464             } else if (st.getDateBirth() < Integer.parseInt(cadena)){
465                 result = -1;
466             }
467             break;
468         case 6:
469             result = (st.getGender()?"a":"z").compareTo((data.equals("1"))?"a":"z");
470             break;
471         case 7:
472             result = (st.getStatus()?"a":"z").compareTo((data.equals("1"))?"a":"z");
473             break;
474     }
475     return result;
476 }
```

- Este método realiza comparaciones entre un objeto Student y un dato de tipo String que se modificará según los criterios específicos (CUI, email, nombre, apellidos, fecha de nacimiento, género y estado) establecidos en un SWITCH CASE que se basa en el orden definido.
- El criterio básico, es la devolución de un valor positivo cuando el primer valor es mayor, uno negativo si es menor, y devuelve 0 si los dos son iguales.

4.6.2. Algoritmo de Búsqueda Binaria Iterativa

Listing 19: Creación de algoritmo Búsqueda Binaria Iterativa

```
375 public static int iterativeBinarySearch(Student arr[], String x) {
376     int l = 0, r = arr.length - 1;
377     while (l <= r) {
378         int m = l + (r - l) / 2;
379         if (comparisonSearch(arr[m], x) == 0)
380             return m;
381         if (comparisonSearch(arr[m], x) < 0)
382             l = m + 1;
383         else
384             r = m - 1;
385     }
386     return -1;
387 }
```

- El método recibe el arreglo ordenado de objetos y el String x que se buscará.
- El entero 'l', recibe el índice izquierdo inicial del arreglo (0). Y el entero 'r' toma el valor del índice derecho inicial del arreglo (tamaño del arreglo menos 1).
- El bucle while de la línea 377 se ejecuta mientras el índice izquierdo (l) sea menor o igual al índice derecho (r), lo que permite saber si aún hay elementos en el rango de búsqueda.
- Dentro del ciclo WHILE, el valor de 'm' se calcula como el índice medio del rango actual (entre l y r). Esto nos asegura que se compara el objeto medio.
- Con el IF de la línea 379, se compara el objeto en la posición m del arreglo con el valor buscado x utilizando el método comparisonSearch(). Si la comparación resulta en 0, se ha encontrado el valor y se devuelve el índice m del objeto hallado.
- En cambio, en el IF de la línea 381, cuando la comparación es menor que 0, se infiere que el valor buscado está en la mitad derecha del rango actual, se actualiza el valor de $l = m + 1$.
- En el ELSE en la línea 383, queda la comparación mayor a 0, lo cual, indica que el valor buscado está en la mitad izquierda del rango actual, y se actualiza r según corresponda
- Por útlimo, si no se encuentra el valor después de completar el bucle, se devuelve -1 para indicar que el valor no se ha encontrado en el arreglo.

4.6.3. Algoritmo de Búsqueda Binaria Recursiva

Listing 20: Creación de algoritmo Búsqueda Binaria Recursiva

```
390 public static int binarySearchRecursive(Student arr[], int l, int r, String x){
391     if (r >= l) {
392         int mid = l + (r - l) / 2;
393         if (comparisonSearch(arr[mid], x) == 0)
394             return mid;
395         if (comparisonSearch(arr[mid], x) > 0)
396             return binarySearchRecursive(arr, l, mid - 1, x);
397 
398         return binarySearchRecursive(arr, mid + 1, r, x);
399     }
400     return -1;
401 }
```

- Se toma como argumentos el arreglo ordenado arr, los índices l y r que representan los límites izquierdo y derecho de la búsqueda, y un String x que es el elemento que se busca.
- La condional IF de la línea 391 verifica si el límite derecho r es mayor o igual que el límite izquierdo l. Si esto es verdadero, se continúa con la búsqueda, en caso contrario, ya se ha recorrido el arreglo y no se encontró x (devuelve -1).
- Se calcula el índice medio del rango actual de búsqueda.
- En el primer IF de la línea 393 se realiza una comparación con el método comparisonSearch() entre el objeto de la posición media y el valor que se busca, al igual que en el iterativo, si son iguales se devuelve inmediatamente el índice m.
- En el siguiente IF, en la línea 395, si la comparación resulta mayor a 0, se sigue la búsqueda recursiva en la mitad izquierda.
- Por último, si no se cumple ninguno (comparación menor a 0), en la línea 398, la búsqueda recursiva se continúa en la mitad derecha del arreglo.

4.7. Gráfica lineal de los algoritmos de ordenamiento/búsqueda

4.7.1. Varibles para el registro de tiempos

Listing 21: Variables Globales para el tiempo

```
20  static long initialTime, currentTime;
21  static double[][] timesSaved, timeDataAB;
22  static double timesSearchBinary, timesSearchBinaryRc;
```

- Para el registro de los tiempos de ejecución de los datos, en la línea 20 se declararon 2 variable globales de tipos long para registrar el tiempo en nanosegundos, antes y después de ejecutar los algoritmos de ordenamiento y búsqueda.
- De igual forma en la línea 21 se declararon dos arreglos bidimensionales (timesSaved) (timeDataAB) de doubles donde se registrarán todos los registros de tiempo de los 7 algoritmos de ordenamiento y de los 2 algoritmos de búsqueda respectivamente.
- Finalmente, en la línea 22 se declararon dos variables de tipo double donde se registrará el último tiempo registrado de los algoritmos de búsqueda, que luego se mostrarán en pantalla.

4.7.2. Registro de tiempos de ejecución en los algoritmos de ordenamiento

Listing 22: Registro de los tiempos de ejecución - algoritmos de ordenamiento

```
136     int rangeIntervals = listado.length/numIntervals;
137     for (int i = 0; i < (numIntervals); i++){
138         Student[] arr = new Student[rangeIntervals * (i + 1)];
139         System.arraycopy(listado, 0, arr, 0, rangeIntervals * (i + 1));
140
141         //iterativeBubbleSort
142         if (algorithms[0]){
143             Student[] arr0 = new Student[arr.length];
144             System.arraycopy(arr, 0, arr0, 0, arr.length);
145             //Se toman los tiempos antes y despues de la ejecuacion del algoritmo para
                    medir sus tiempos,
```

```
146             //Estos son guardados en un arreglo bidimensional, esto se realiza en cada
                    algoritmo.
147             initialTime = System.nanoTime();
148             bubbleSortIterative(arr0);
149             currentTime = System.nanoTime();
150             timesSaved[0][i] = currentTime - initialTime;
151         }
152
153         //iterativeSelectionSort
154         if (algorithms[1]){
155             Student[] arr1 = new Student[arr.length];
156             System.arraycopy(arr, 0, arr1, 0, arr.length);
157             initialTime = System.nanoTime();
158             iterativeSelectionSort(arr1);
159             currentTime = System.nanoTime();
160             timesSaved[1][i] = currentTime - initialTime;
161         }
162
163         //iterativeInsertionSort
164         if (algorithms[2]){
165             Student[] arr2 = new Student[arr.length];
166             System.arraycopy(arr, 0, arr2, 0, arr.length);
167             initialTime = System.nanoTime();
168             iterativeInsertionSort(arr2);
169             currentTime = System.nanoTime();
170             timesSaved[2][i] = currentTime - initialTime;
171         }
172
173         //mergeSort
174         if (algorithms[3]){
175             Student[] arr3 = new Student[arr.length];
176             System.arraycopy(arr, 0, arr3, 0, arr.length);
177             initialTime = System.nanoTime();
178             mergeSort(arr3, 0, arr3.length - 1);
179             currentTime = System.nanoTime();
180             timesSaved[3][i] = currentTime - initialTime;
181         }
182
183         //recursiveBubbleSort
184         if (algorithms[4]){
185             Student[] arr4 = new Student[arr.length];
186             System.arraycopy(arr, 0, arr4, 0, arr.length);
187             initialTime = System.nanoTime();
188             recursiveBubbleSort(arr4, arr4.length);
189             currentTime = System.nanoTime();
190             timesSaved[4][i] = currentTime - initialTime;
191         }
192
193         //recursiveSelectionSort
194         if (algorithms[5]){
195             Student[] arr5 = new Student[arr.length];
196             System.arraycopy(arr, 0, arr5, 0, arr.length);
197             initialTime = System.nanoTime();
198             recursiveSelectionSort(arr5, arr5.length, 0);
199             currentTime = System.nanoTime();
200             timesSaved[5][i] = currentTime - initialTime;
```

```
201             }
202
203             //recursiveInsertionSort
204             if (algorithms[6]){
205                 Student[] arr6 = new Student[arr.length];
206                 System.arraycopy(arr, 0, arr6, 0, arr.length);
207                 initialTime = System.nanoTime();
208                 recursiveInsertionSort(arr6, arr6.length);
209                 currentTime = System.nanoTime();
210                 timesSaved[6][i] = currentTime - initialTime;
211             }
212         }
213
214         graphic(1);
```

- En un comienzo se colocaron las variables de tiempo dentro de cada algoritmo lo cual registraba el tiempo que se demoraba el algoritmo en ordenar cada elemento, sin embargo esto no permitía obtener los gráficos que necesitábamos.
- Luego se optó por colocar las variables antes y después de llamar al método del algoritmo de ordenamiento, para poder registrar el tiempo que demora en el ordenamiento de todos los elementos.
- El buble FOR de la línea 137 permite realizar la obtención de los datos de tiempo, teniendo en cuenta una variación en la cantidad de elementos a ordenar, que va aumentando sucesivamente.
- En la línea 143 se puede apreciar que se declara e inicializa un arreglo al cual se le copian los elementos originales del arreglo principal, y esto se realiza con todos los métodos, logrando que cada uno de los algoritmos de ordenamiento reciba un arreglo que pueda ordenar.
- En la línea 214 se hace un llamado al método graphic(1) con el parámetro 1 para que realice la gráfica lineal de los tiempos de los algoritmos de ordenamiento.

4.7.3. Registro de tiempos de ejecución en los algoritmos de búsqueda

Listing 23: Registro de los tiempos de ejecución - algoritmos de ordenamiento

```
565             //Se realiza el ordenamiento de los datos en el arreglo principal, segun el
                    atributo especificado.
566             int opcionSeleccionadaDate = comboBoxDate.getSelectedIndex();
567             datoAbuscar = campoDato.getText();
568             orden = opcionSeleccionadaDate;
569             mergeSort(listado, 0, listado.length-1);
570
571             //Llama a los algoritmos de busqueda para encontrar el registro especificado.
572             int site = -1;
573             timeDataAB = new double[2][numIntervals];
574             int rangeIntervals = listado.length/numIntervals;
575             for (int i = 0; i < (numIntervals); i++){
576                 Student[] arr = new Student[rangeIntervals * (i + 1)];
577                 System.arraycopy(listado, 0, arr, 0, rangeIntervals * (i + 1));
578                 initialTime = System.nanoTime();
579                 iterativeBinarySearch(arr, datoAbuscar);
580                 currentTime = System.nanoTime();
581                 timesSearchBinary = currentTime - initialTime;
```

```
582             timeDataAB[0][i] = timesSearchBinary;
583
584             Student[] arr1 = new Student[rangeIntervals * (i + 1)];
585             System.arraycopy(listado, 0, arr1, 0, rangeIntervals * (i + 1));
586             initialTime = System.nanoTime();
587             site = binarySearchRecursive(arr1, 0, arr1.length-1, datoAbuscar);
588             currentTime = System.nanoTime();
589             timesSearchBinaryRc = currentTime - initialTime;
590             timeDataAB[1][i] = timesSearchBinaryRc;
591         }
592
593         String result;
594         if (site != -1){
595             result = "Estudiante encontrado:<br>"+listado[site];
596         } else {
597             result = "No existe el estudiante";
598         }
599
600         SwingUtilities.invokeLater(() -> {
601             crearVentana(result);
602         });
603         //Llama al metodo para graficar los tiempos de los algoritmos de busqueda.
604         graphic(2);
```

- Para ejecutar los algoritmos de búsqueda primero se debe ordenar el arreglo, por lo que se llama al método mergeSort para ordenar los elementos dependiendo de lo que el usuario quiera buscar.
- Si el usuario quiere buscar un nombre, entonces el ordenamiento será según los nombres.
- En la línea 576 se puede apreciar que se declara e inicializa un arreglo al cual se le copian los elementos originales del arreglo principal, y esto también se realiza antes del otro método, logrando que cada uno de los algoritmos de búsqueda reciba un arreglo en donde buscar.
- En la línea 578 y 580 se registran los tiempos antes y después de realizada la búsqueda binaria iterativa, lo mismo sucede con la recursiva, y luego la resta de ambos valores se guarda en sus respectivas variables.
- El bucle FOR de la línea 575 permite realizar la obtención de los datos de tiempo, teniendo en cuenta una variación en la cantidad de elementos donde se va a realizar la búsqueda, que va aumentando sucesivamente.
- En la línea 594 existe una condicional que determina que si con el dato especificado se logró encontrar al estudiante o no. Si el valor a evaluar es -1 entonces no se encontró al estudiante, si cualquier otro número, este representara el índice y por lo tanto la ubicación del elemento dentro del arreglo.
- En la línea 604 se hace un llamado al método graphic(2) con el parámetro 2 para que realice la gráfica lineal de los tiempos de los algoritmos de ordenamiento.

4.7.4. Gráficas lineales

Listing 24: Método para crear la gráfica lineal

```
481     public static void graphic(int num) {
482         String title;
```

```
483         XYSeries[] series;
484         int seriesT;
485         if (num == 1){
486             title = "ordenamiento";
487             seriesT = 7;
488             series = new XYSeries[seriesT];
489             for (int i = 0; i < seriesT; i++){
490                 if (algorithms[i]){
491                     series[i] = new XYSeries(algorithm[i]);
492                     for (int j = 0; j < timesSaved[i].length; j++){
493                         series[i].add(j, timesSaved[i][j]);
494                     }
495                }
496             }
497         } else {
498             title = "busqueda";
499             seriesT = 2;
500             series = new XYSeries[seriesT];
501             series[0] = new XYSeries("Busqueda iterativa");
502             series[1] = new XYSeries("Busqueda recursiva");
503             for (int i = 0; i < seriesT; i++)
504                 for (int j = 0; j < timeDataAB[i].length; j++)
505                     series[i].add(j, timeDataAB[i][j]);
506         }
507 
508         XYSeriesCollection dataset = new XYSeriesCollection();
509         for (int i = 0; i < seriesT; i++){
510             if (series[i] != null){
511                 dataset.addSeries(series[i]);
512             }
513         }
514         JFreeChart chart = ChartFactory.createXYLineChart(
515             "Tiempo de ejecucion de los algoritmos de "+ title +" con respecto al tamao de
                 datos",
516             " Tamao  de datos",
517             "Tiempo",
518             dataset,
519             PlotOrientation.VERTICAL,
520             true,
521             true,
522             false
523         );
524 
525         XYPlot plot = chart.getXYPlot();
526         NumberAxis xAxis = (NumberAxis) plot.getDomainAxis();
527 
528         xAxis.setTickUnit(new NumberTickUnit(20.0));
529 
530         ChartPanel chartPanel = new ChartPanel(chart);
531         chartPanel.setPreferredSize(new java.awt.Dimension(800, 600));
532 
533         JFrame frame = new JFrame("Grfico Lineal");
534         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
535         frame.getContentPane().add(chartPanel);
536         frame.pack();
537         frame.setVisible(true);
```

- El método recibe un entero que permitirá diferenciar si la gráfica es para los algoritmos de ordenamiento o para los de búsqueda.
- En la condicional de la línea 485 en caso de que el número sea 1, se hará referencia a la gráfica de los algoritmos de ordenamiento, se crearan las series de aquellos algoritmos que hayan sido seleccionados en la primera interfaz (línea 491) y también se añadirán sus registros de tiempos a las series (línea 493).
- En caso de que el número sea diferente de 1, se hará referencia a la gráfica de los algoritmos de búsqueda, se crearan las series de los dos algoritmos (línea 501 y 502) y también se añadirán sus registros de tiempos a las series (línea 505)
- En la línea 508 se crea el objeto llamado "dataset"del tipo XYSeriesCollecion, luego, en las siguientes líneas se agregan todas las series al conjunto de datos "dataset".
- Se crea la variable chart que será el grafico que representara los datos, seguidamente se especifican el título de la gráfica, los títulos del ejeX y del ejeY, el conjunto de datos, la orientación del gráfico, y otros parámetros booleanos (mostrar leyendas, mostrar etiqueta de herramienta emergente, generar una URL para el gráfico).
- Luego de las líneas 520 a la 532, se establecen otros parámetros y características de la gráfica, para que, finalmente la sentencia de la línea 532 permita la visualización de la gráfica.

4.8. Commits

Listing 25: Comit n°1

```
$ git add .
$ git commit -m "Creamos la clase Student.java con los atributos necesarios, adems del
    archivo StudentRegistration.java. Por el momento lee los archivos .csv y los
    imprime (utilizando arreglos estndar)"
$ git push -u origin main
```

- En este commit de registro la creación de la clase Student.java con todos sus atributos, métodos mutadores, métodos accesores y el método toString para que se impriman los objetos con todos sus datos.

Listing 26: Comit n°2

```
$ git add .
$ git commit -m "Creando arreglos de datos del tiempo, crendo un metodo grafic que
    permite realizar una grafica lineal de los datos"
$ git push -u origin main
```

- Commit realizado para añadir los arreglos creados que se encargarán de registrar los tiempos de ejecución de los algoritmos utilizados, asimismo se implementó el método graphic() que crea el gráfico lineal de los tiempos de ejecución.

Listing 27: Comit n°3

```
$ git add .
$ git commit -m "Agregando la busqueda binaria recursiva, ademas del metodo
    comparisionSearch que sera utilizado en la busqueda"
$ git push -u origin main
```

- Este commit fue realizado para añadir el método que ejecuta la búsqueda binaria recursiva, para su correcto funcionamiento, se creó en conjunto a un método llamado comparisionSearch, que permitió comparar los datos según el parámetro requerido con un SwitchCase implementado.

Listing 28: Comit n°4

```
$ git add .
$ git commit -m "Implementando los metodos de la interfaz restantes, aunque aun
    seguiran en revision por posibles errores u omisiones"
$ git push -u origin main
```

- En el commit se agregó los métodos en la interfaces para interactuar con el usuario, tanto para el ordenamiento, como para la búsqueda.
- Pero están en revisión debido a posibles errores en casos específicos.

Listing 29: Comit n°5

```
$ git add .
$ git commit -m "Arreglando tlimos errores de la Bsqueda, versin final"
$ git push -u origin main
```

- Commit realizado para añadir los últimos cambios en el código para corregir un error dentro de la búsqueda de datos que solo permitía buscar una única vez, con esta corrección ahora se pueden realizar varias búsquedas una después de otra.

4.9. Compilación y ejecución del programa

Listing 30: Compilacion y ejecución

```
$ javac -cp jfreechart-1.0.19.jar:jcommon-1.0.23.jar StudentRegistration.java
    Student.java
  $ java -cp .:jfreechart-1.0.19.jar:jcommon-1.0.23.jar StudentRegistration
```

4.9.1. Primera interfaz

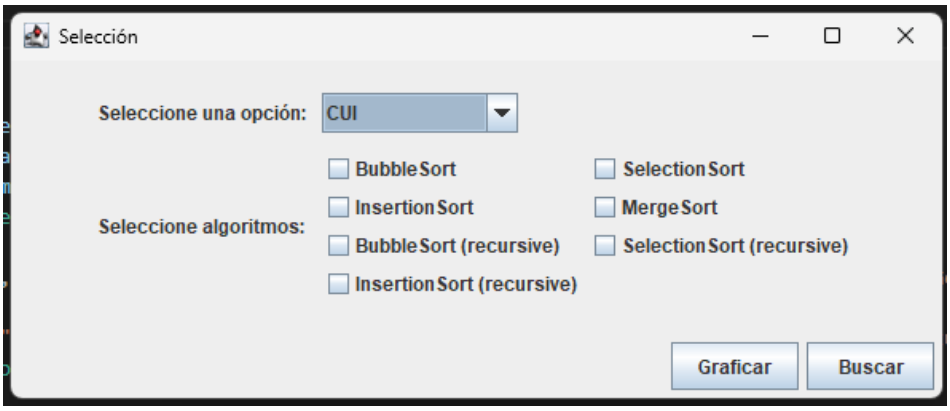

Figura 4: Interfaz 1

- El programa le permite al usuario poder seleccionar el atributo con el cual realizará el ordenamiento, además de especificar los algoritmos que desea utilizar, los cuales se mostrarán en la gráfica lineal.
- Al hacer click en el botón "Graficar"se mostrará en pantalla el gráfico lineal correspondiente.
- Al hacer click en el botón "Buscar"se cerrará automáticamente esta interfaz y se mostrará la segunda interfaz de búsqueda.

4.9.2. Gráfico lineal de algoritmos de ordenamiento

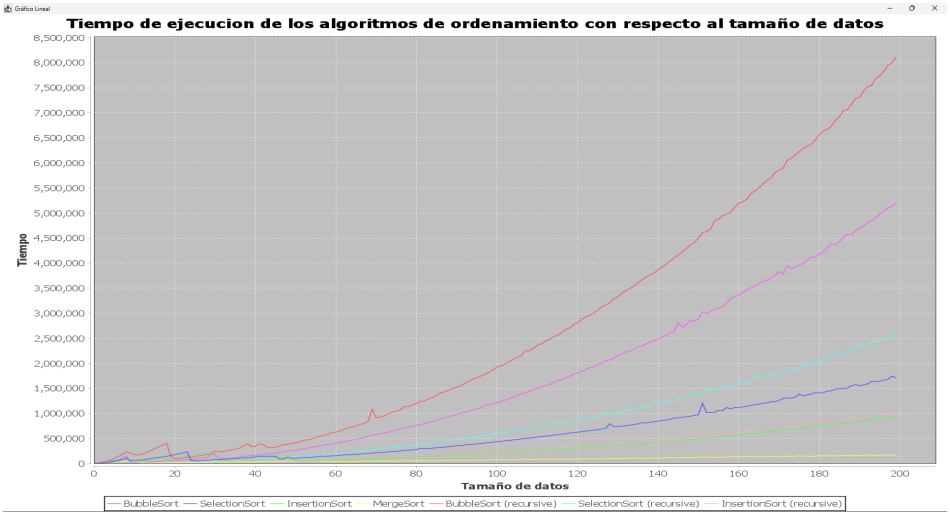

Figura 5: Gráfico lineal de algoritmos de ordenamiento

- Comparando todos lo gráficos lineales de los tiempos que les toma a los algoritmos ordenar la diferente cantidad de elementos, podemos notar que el algoritmo MergeSort es el más veloz y eficiente para realizar dicho ordenamiento.
- Es por esto que, para antes de la ejecución de los algoritmos de búsqueda se optó por usar el algortimo MergeSort por su eficacia.

4.9.3. Segunda interfaz

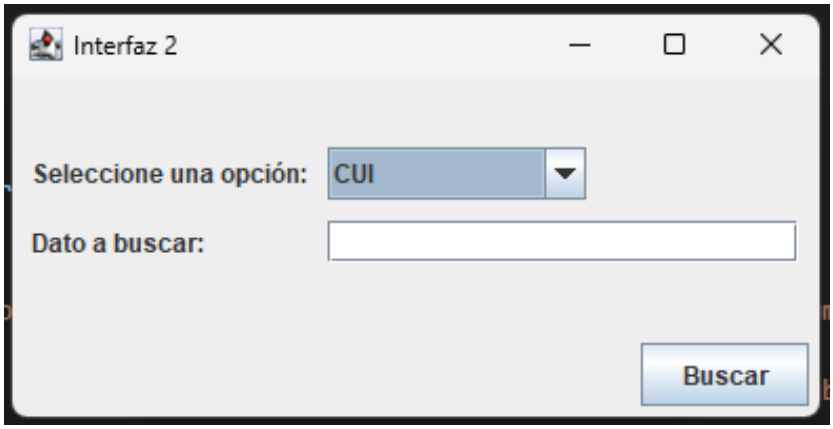

Figura 6: Interfaz 2

- Luego de hacer click en el botón "Buscar", automáticamente se abre esta interfaz, la cual le permite al usuario elegir el atributo e ingresar el dato a buscar.
- Al hacer click en el botón seguidamente se muestra una nueva ventana especificando si el resgistro del estudiante fue encontrado o si no existe, en caso de ser mostrado se imprimirán los datos del estudiante.
- Por otro lado, también se mostrará en pantalla el gráfico lineal de los dos algoritmos de ordenamiento.

4.9.4. Gráfico lineal de algoritmos de búsqueda

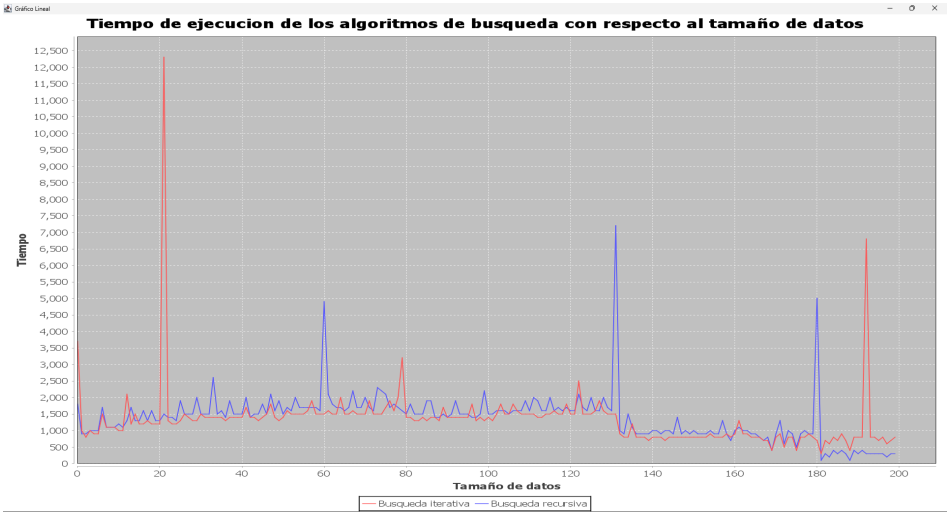

Figura 7: Gráfico lineal de algoritmos de búsqueda

- En la gráfica se puede apreciar los tiempos que le toma a cada algoritmo de búsqueda encontrar el dato ingresado por el usuario, teniendo en cuenta la variación del tamaño de datos entre los cuales se realizará la búsqueda.

- Realizando un análisis podemos concluir que el algoritmo de búsqueda binaria (recursiva) resulta ser el mas eficiente y rápido.

4.9.5. Ventana de resultado de búsqueda

Figura 8: Ventana de resultado de búsqueda

- Luego de haberse realizado la búsqueda, en la última ventana son mostrados los resultados de esta.
- En caso se encuentre al estudiante en el registro, se mostrarán todos sus datos, como se muestra el la figura 8, caso contrario se le informará al usuario que el dato ingresado no corresponde a ningún estudiante.

4.10. Estructura de práctica 01

- El contenido que se entrega en este laboratorio es el siguiente:

```
prac01/
|--- jcommon-1.0.23.jar
|--- jfreechart-1.0.19.jar
|--- Student.java
|--- StudentRegistration.java
|--- StudentData.csv
|--- latex
   |--- img
   |   |--- Interfaz1.png
   |   |--- Interfaz2.png
   |   |--- logo_abet.png
   |   |--- logo_episunsa.png
   |   |--- logo_unsa.jpg
   |   |--- Result_Search.png
   |   |--- Search_Grafic.png
   |   |--- Sort_Search.png
   |--- prac01_informe.pdf
   |--- prac01_informe.tex
   |--- src
       |--- StudentRegistration.java
       |--- Student.java
```

5. Rúbricas

5.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y fácil de leer.

5.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplió con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 3: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	3	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente estan dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	3	
Total		20		18	

6. Referencias

- https://parzibyte.me/blog/2020/08/30/java-ordenamiento-seleccion/
- http://puntocomnoesunlenguaje.blogspot.com/2015/02/ordenamiento-insercion-directa-java.html
- https://www.geeksforgeeks.org/bubble-sort/
- https://www.geeksforgeeks.org/insertion-sort/
- https://www.geeksforgeeks.org/selection-sort/
- https://www.geeksforgeeks.org/java-program-for-merge-sort/
- https://www.geeksforgeeks.org/binary-search/
- https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html
- https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html
- https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html
- https://docs.oracle.com/javase/8/docs/api/java/lang/Exception.html
- https://www.jfree.org/jfreechart/
- https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F%2F/javax/swing/package-summary.html