Informe Prática 01

Análisis de los algoritmos de ordenamiento InsertionSort y QuickSort, y el algoritmo de búsqueda binaria

Nota

Estudiantes	Escuela	Asignatura
Arias Quispe, Jhonatan David Mamani Huarsaya, Jorge Luis Mollo Chuquicaña, Dolly Yadhira Quispe Condori, Alvaro Raul Velarde Saldaña, Jhossep Fabritzio jariasq@unsa.edu.pe jmamanihuars@unsa.edu.pe dmolloc@unsa.edu.pe aquispecondo@unsa.edu.pe jvelardesa@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de Programación 2 Semestre: II Código: 17013

Teoría	Tema	Duración
Prática 01	Análisis de los algoritmos de ordenamiento InsertionSort y QuickSort, y el algoritmo de búsqueda binaria	04 horas

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

1. Actividad

- 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 ArchCraft GNU Linux 64 bits Kernell, Sistema Operativo Ubuntu GNU Linux 22.04, Sistema operativo windows 11
- NeoVim, vs code
- OpenJDK 64-Bit 20.0.1
- Gnuplot 5.4.9
- Git 2.42.0
- Cuenta en GitHub con el correo institucional.
- Programación Orientada a Objetos.
- Algoritmos de ordenamiento y búsqueda.

3. URL de Repositorio Github

- URL para acceder a la Prática 01 en el Repositorio GitHub.
- `https://github.com/JhonatanDczel/prac01`

4. Documentacion

5. Desarrollo de la actividad

5.1. Lectura y almacenamiento de datos

5.1.1. Descripción

La clase `Reader` se encarga de leer los datos de un archivo CSV y almacenarlos en un arreglo de objetos `Student`. Los pasos principales son:

- Abrir el archivo `data.csv`
- Leer linea por linea con `BufferedReader`
- Separar cada linea por comas para obtener los campos
- Crear un nuevo objeto `Student`
- Setear los campos leídos en el objeto `Student`
- Agregar el objeto `Student` a un `ArrayList`
- Convertir el `ArrayList` a un arreglo al final

5.1.2. Código relevante

Listing 1: Método para leer archivo CSV

```java
private void readDataFile(){

  BufferedReader reader = null;

  String line = "";

  String[] parts;

  try {

    FileReader fileReader = new FileReader("./reader/data.csv");

    reader = new BufferedReader(fileReader);

    while ((line = reader.readLine()) != null) {

      parts = line.split(",");

      //... creacin y seteo de Student

    }

  }catch (Exception e){

    e.printStackTrace();

  }finally{

    //...cerrar reader

  }

}
```

Listing 2: Conversión de ArrayList a arreglo

```java
classmates = students.toArray(new Student[students.size()]);
```

5.1.3. Explicación

Se utiliza `BufferedReader` para leer el archivo línea por línea de forma eficiente. Cada línea se divide en las partes correspondientes a cada campo, delimitadas por comas. Luego se crea un objeto `Student`, se setean sus campos y se agrega al `ArrayList`. Al final se convierte el `ArrayList` a arreglo para retornarlo.

5.2. Algoritmo de insercion

El ordenamiento por inserción es un algoritmo de ordenamiento simple y eficiente que funciona de la siguiente manera:

- Comienza con un arreglo desordenado.
- Divide el arreglo en dos partes: una parte ordenada y una parte desordenada.
- En cada iteración, toma el primer elemento de la parte desordenada y lo compara con los elementos en la parte ordenada.

- Inserta el elemento en la posición correcta en la parte ordenada, desplazando los elementos mayores a la derecha.
- Repite este proceso hasta que la parte desordenada esté vacía y todos los elementos estén en la parte ordenada.

```
INSERTION-SORT(A, n)
1  for i = 2 to n
2      key = A[i]
3      // Insert A[i] into the sorted subarray A[1 : i − 1].
4      j = i − 1
5      while j > 0 and A[j] > key
6          A[j + 1] = A[j]
7          j = j − 1
8      A[j + 1] = key
```

Figura 1: Pseudocódigo del ordenamiento por inserción

El ordenamiento por inserción es eficiente para arreglos pequeños o parcialmente ordenados, pero puede volverse ineficiente para arreglos grandes debido a su complejidad cuadrática en el peor de los casos. Sin embargo, es estable y fácil de implementar.

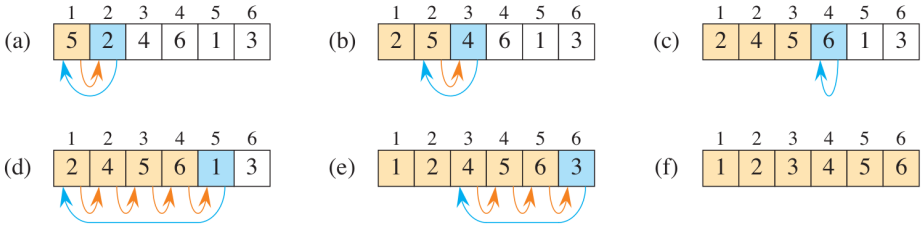

Figura 2: Iteraciones del ordenamiento por inserción

Entendido el funcionamiento de este algoritmo, podemos aplicarlo para resolver esta actividad.

- Comprendamos que nosotros estamos trabajando con un arreglo de objetos, es decir, trabajamos con referencias a objetos que se estan almacenando en cada posición del arreglo.
- Las comparaciones se hacen según los atributos de la clase Student, por lo tanto, tenemos la necesidad de usar sus métodos accesores; en este caso solo usamos los métodos getters.
- La claridad es mejor que la astucia.

5.2.1. Commits principales

5.3. Quicksort

- El método «Quicksort», también conocido como ordenación rápida, es uno de los algoritmos de ordenación más eficientes y ampliamente utilizados en la informática.
- Se basa en el paradigma de divíde y vencerás. Consiste en seleccionar un elemento de la lista, llamado "pivote,ƨ reorganizar los elementos en la lista de manera que los elementos menores que el pivote estén a su izquierda, y los elementos mayores estén a su derecha.

- Cuando se elige un pivote adecuado, «Quicksort» puede alcanzar su rendimiento óptimo, que es O(n log n) en promedio. Sin embargo, en el peor de los casos, se puede degradar a O(n ^{2}).

5.3.1. Commits de la implementación de Quicksort

- **Hash: afde7b901d3ac3c8414a3d6976fde97bcbe9729d**
- En el presente commit se implementó el quicksort para ordenar los CUI, sin embargo cuenta con un error. Cambia la referencia del atributo cui de la clase Student, sin embargo, no cambia la la referencia del objeto mismo, es un error que se solucionará más adelante.

Listing 3: Commit: Se implementó Quicksort para cui

```
public static void Cui (Reader.Student[] s, int left, int right) {
    int i = left;
    int j = right;
    int aux;
    while (i < j) {
        while (s[i].getCui() <= piv && i < j) i++;
        while (s[j].getCui() > piv) j--;
        if (i < j) {
            aux = s[i].getCui();
            s[i].setCui(s[j].getCui());
            s[j].setCui(aux);
        }
     }
    s[left].setCui(s[j].getCui());
    s[j].setCui(piv);
    if (left < j - 1) Cui(s, left, j -1);
    if (j + 1< right) Cui(s, j + 1, right);

 }
```

- **Hash: c9e70896b8edb82f963375cac9a4cb72d67ed6fb**
- En el presente commit de manera similar se implementó el método email para ordenar según la primera letra a los correos, que están representados mediante Strings, sin embargo, presenta el error de solo ordenar la primera letra. Más adelante se parchará el error.

Listing 4: Commit: Se añadió el método email para ordenar los correos según quicksort

```

public static void email (Reader.Student[] s, int left, int right) {
    char piv = s[left].getEmail().charAt(0);
    int i = left;
    int j = right;
    String aux;
    while (i < j) {
        while (s[i].getEmail().charAt(0) <= piv && i < j) i++;
        while (s[j].getEmail().charAt(0) > piv) j--;
        if (i < j) {
            aux = s[i].getEmail();
            s[i].setEmail(s[j].getEmail());
            s[j].setEmail(aux);
        }
    }
    aux = s[left].getEmail();
    s[left].setEmail(s[j].getEmail());
    s[j].setEmail(aux);
    if (left < j - 1) email(s, left, j -1);
```

```
20        if (j + 1< right) email(s, j + 1, right);
21     }
```

- **Hash: cb4418c7361dc0e2da32042698b2400e79beebfd**
- Se creó la fución «smallerThan()» en la cual se usa el método String.compareTO(String), la función de esta misma es restar los caracteres según su orden ascii. Si w1 - w2 es positivo, significa que w1 debería estar después de w2.
- Esta función se utiliza cada vez que se quiere comparar dos String, solucionando el error mencionado anteriormente.

Listing 5: Commit: Se mejoró los métodos que ordenan Strings

```
1     public static boolean smallerThan(String w1, String w2) {
2         w1.toUpperCase();
3         w2.toUpperCase();
4         if (w1.compareTo(w2) <= 0) return true;
5         return false;
6
7     }
8
9  // Uso de smallerThan en otras funciones:
10
11    public static void lastNameM (Reader.Student[] s, int left, int right) {
12        String piv = s[left].getLastNameM();
13        int i = left;
14        int j = right;
15        String aux;
16        while (i < j) {
17            while (smallerThan(s[i].getLastNameM(), piv) && i < j) i++;
18            while (!smallerThan(s[j].getLastNameM(), piv)) j--;
19            if (i < j) {
20                aux = s[i].getLastNameM();
21                s[i].setLastNameM(s[j].getLastNameM());
22                s[j].setLastNameM(aux);
23            }
24        }
25        aux = s[left].getLastNameM();
26        s[left].setLastNameM(s[j].getLastNameM());
27        s[j].setLastNameM(aux);
28        if (left < j - 1) lastNameM (s, left, j -1);
29        if (j + 1< right) lastNameM(s, j + 1, right);
30    }
```

- **Hash: 0c89058320e8f2dc1981a4666c3deb86df450702**
- Se solucioó el primer error mencionado, en esta oportunidad, el código intercambia la referencia de los objetos en el array en vez de sus atributos.

Listing 6: Commit: Se solucionó error de cambio de valores de atributos en vez de las referencias de los objetos

```
1     public static void email (Reader.Student[] s, int left, int right) {
2          String piv = s[left].getEmail();
3          int i = left;
4          int j = right;
5          Reader.Student aux;
6          while (i < j) {
```

```
            while (smallerThan(s[i].getEmail(), piv) && i < j) i++;
            while (!smallerThan(s[j].getEmail(), piv)) j--;
            if (i < j) {
                aux = s[i];
                s[i] = s[j];
                s[j] = aux;
            }
        }
        aux = s[left];
        s[left] = s[j];
        s[j] = (aux);
        if (left < j - 1) email(s, left, j -1);
        if (j + 1< right) email(s, j + 1, right);
    }
```

- Hash:
-

Listing 7: TITULOOOOOO

5.4. Busqueda Binaria

- La búsqueda binaria es un algoritmo de búsqueda eficiente utilizado para encontrar un elemento específico en una lista ordenada.
- Tanto la versión recursiva como la iterativa de la búsqueda binaria se basan en el mismo principio: dividir y vencerás.
- En la búsqueda binaria recursiva, se divide repetidamente la lista en dos mitades y se compara el elemento buscado con el elemento en el medio. Este proceso se repite de manera recursiva hasta que se encuentre el elemento deseado
- La búsqueda binaria iterativa se implementa mediante un bucle en lugar de una función recursiva.

5.4.1. Forma iterativa

- **Hash: 55e501872e486ef65d414a41aeb9f673c6b8318e**
- Se creo «IterativeBinarySearch.java» donde se implementó el método cui() para realizar la búsqueda binaria de manera iterativa.

Listing 8: Commit: Se creó el método cui para la búsqueda binaria

```
public static int cui (Reader.Student [] s, int x) {
  int l, r;
  l = 0;
  r = s.length - 1;
  while (l <= r) {
    int m = l + ( r - l ) / 2;
    if (s[m].getCui() == x) return m;
    if (s[m].getCui() < x) l = m + 1;
    else r = m - 1;
  }
  return -1;
}
```

- **Hash: b90e4fbe4c4373de351edf016e255e9534088251**
- De manera similar, se implementó el método « email() » usando como comparador la función «smallerThan()».
- Usando esta última función sabemos si la palabra buscada debería estar al delante o atrás.
- Usando este método que funciona con Strings, también se implementó el resto de funciones según los atributos de la clase Student.

Listing 9: Commit: Implementación para busqueda según email

```
 public static int email (Reader.Student [] s, String x) {
   int l = 0, r = s.length - 1;
   while (l <= r) {
     int m = l + (r - l) / 2;
     if (s[m].getEmail().equals(x)) return m;
     if (smallerThan(s[m].getEmail(), x)) l = m + 1;
     else r = m - 1;
   }
   return -1;
 }

 public static boolean smallerThan(String w1, String w2) {
     w1.toUpperCase();
     w2.toUpperCase();
     if (w1.compareTo(w2) <= 0) return true;
     return false;

   }

}
```

5.4.2. Forma recursiva

- **Hash: 2318dc5647d46b889c72f61a09a9fd7b8f8f2f5a**
- Posteriormente se creo el archivo «RecursiveBinarySearch.java» donde se implementaron las siguientes clases: «cui()» y «email()».
- Estas clases usan el algoritmo de la búsqueda binaria recursiva. Se reutiliza la función «smallerThan()» para comparar Strings.
- De manera análoga se implementa al resto de atributos de la clase Student que están basados en Strings, los cuales no se muestran en el presente informe para evitar una extensión innecesaria del mismo.

Listing 10: Commit: Se creó la búsqueda binaria con los métodos de cui y email

```
package algorithms;

import reader.Reader;

public class RecursiveBinarySearch {
 public static int cui (Reader.Student [] s, int x, int l, int r) {
   if (r >= l) {
     int m = l + (r - l) / 2;
     if (s[m].getCui() == x) return m;
     if (s[m].getCui() > x) return cui(s, x, l, m -1);
     return cui(s, x, m + 1, r);
```

```
    }
    return -1;
  }
  public static int email (Reader.Student [] s, String x, int l, int r) {
    if (r >= l) {
      int m = l + (r - l) / 2;
      if (s[m].getEmail().equals(x)) return m;
      if ( smallerThan(s[m].getEmail(), x) ) return email(s, x, m + 1, r);
      return email(s, x, l, m - 1);
    }
    return -1;
  }
  public static boolean smallerThan(String w1, String w2) {
      w1.toUpperCase();
      w2.toUpperCase();
      if (w1.compareTo(w2) <= 0) return true;
      return false;

    }

}
```

5.5. Parte grafica

5.5.1. Commits principales

6. Ejecucion del programa

- A continuacion veremos la ejecucion del programa en la terminal:

7. Rúbricas

7.1. Rúbrica para el contenido del Informe y demostración

- El equipo debe marcar o dejar en blanco las celdas de la columna **Checklist** si se cumplio con el ítem correspondiente.
- El equipo debe autocalificarse en la columna **Equipo** de acuerdo a la siguiente tabla:

Tabla 1: 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 2: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio de trabajo en el 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	