Universidad de los Andes

Ingeniería de Sistemas y Computación ISIS1206 – Estructura de Datos Laboratorio 3

Laboratorio 3: Algoritmos de Ordenamiento

# Objetivos

* Entender la implementación y el funcionamiento de los algoritmos de ordenamiento básicos.
* Entender la complejidad de los algoritmos de ordenamiento básicos por sus tiempos de ejecución.
* Utilizar adecuadamente el ambiente de trabajo distribuido basado en Git y Github.

# Fecha Límite de Entrega

Este laboratorio se realizará en dos partes:

Parte 1. Comprensión y pruebas de los algoritmos de ordenamiento iterativos – 17 Febrero al 23 Febrero.

Parte 2. Comprensión y pruebas de los algoritmos de ordenamiento recursivos – 24 Febrero al 1 Marzo.

**Fecha límite Entrega Única**: Lunes 1 de Marzo antes de la media noche (11:59 p.m.)

Santiago Pulido Torres - 201730218

Juan David Briceño - 201812887

# Lo que usted debe hacer

## Parte 1 (Durante el laboratorio)

1. Estudiar las implementaciones de los algoritmos de ordenamiento vistos hasta el momento en clase y que se presentaron en las hojas de trabajo del tema de Ordenamiento.
2. Se va reutilizar el resultado del laboratorio 2 (TAD Lista y Estructuras de Datos ArregloDinamico y ListaEnlazada). Hacer *fork* del repositorio de su grupo T2\_202110 y colocarle nombre **T3\_202110**. En este repositorio se espera que se haya resuelto la lectura de los datos del Reto 1 del curso (archivo videos-large.csv) en ambas Estructura de Datos del TAD Lista.
3. Configure su nuevo repositorio para vincular a sus compañeros de grupo (en el rol de administrador o propietario), a los monitores y al profesor de su sección. Tenga presente que el repositorio debe ser configurado como privado.
4. Clonar el nuevo repositorio en los computadores de su grupo.
5. Descargue el archivo completo de videos del Reto 1 (videos-all.zip) y descomprímalo en la carpeta data del proyecto.
6. Completar el TAD ILista<T> para incluir el nuevo método:

/\*\*

* + Crear una sublista de la lista original (*this*).
  + Los elementos se toman en el mismo orden como aparecen en la lista original (*this*).
  + **@param** número de elementos que contendrá la sublista. Si el número es superior al tamaño
  + original de la lista, se obtiene una copia de la lista original.
  + **@return** sublista creada con la misma representación de la lista original (*this*).

\*/

### public ILista<T> sublista(int numElementos);

Implementar el nuevo método en las estructuras ArregloDinamico<T> y ListaEncadenada<T>.

1. Actualizar el Menú de Opciones de la Aplicación (archivo Controller.java) para ofrecer las siguientes opciones:

− Cargar el archivo completo de videos Youtube (videos-all.csv del Reto 1) en una Lista. Se debe dar la posibilidad al usuario de escoger si usa una representación Arreglo Dinámico o Lista Enlazada. Como resultado debe mostrarse el total de videos cargados en la lista.

− Obtener una muestra de videos de la lista cargada de un tamaño dado por el usuario. La muestra debe iniciar desde el primer elemento de la lista cargada. Si el tamaño de la muestra excede el tamaño de la lista, la muestra será la copia de la lista original. Como resultado debe confirmarse el número de videos que contiene la muestra.

**Ayuda**: Usar el nuevo método del TAD ILista subLista( … ) que crea una sublista con los elementos de una muestra. La muestra se obtiene en una lista del mismo tipo que la lista original.

− Aplicar uno de los algoritmos de ordenamiento sobre la **muestra de los videos**. Se debe dar la posibilidad al usuario de escoger el algoritmo a aplicar: Insertion sort, Shell sort, Merge sort o Quick sort.

Como resultado debe mostrarse el tiempo de ejecución en mili-segundos del algoritmo de ordenamiento, los primeros diez videos (título y valor del criterio de ordenamiento) y los últimos diez videos (título y valor del criterio de ordenamiento) resultado del ordenamiento.

1. Actualizar el avance realizado en el repositorio local en el repositorio remoto en Github.

## Parte 2 (Trabajo extra-laboratorio) – Implementación para el Ordenamiento de una Lista de Videos

1. En la clase VideoYoutube definir una clase interna que implemente Comparator<VideoYoutube> que permita comparar dos objetos de la clase por su número de likes así:

### public class VideoYoutube implements Comparable<VideoYoutube>

{

// Definicion de atributos

…

/\*\* Comparación natural de acuerdo a algún atributo con identificación única

* **@return** valor 0 si *this* y otro son iguales. Numero negativo si *this* es menor a otro.
* Numero positivo *this* es mayor a otro \*/

### public int compareTo(VideoYoutube otro) {

…

return resultado;

}

/\*\* Comparador alterno de 2 videos por número de likes

### public static class ComparadorXLikes implements Comparator<VideoYoutube> {

/\*\* Comparador alterno de acuerdo al número de likes

\* **@return** valor 0 si video1 y video2 tiene los mismos likes. valor negativo si video1 tiene menos likes que video2. valor positivo si video1 tiene más likes que video2. \*/

### public int compare(VideoYoutube video1, VideoYoutube video2) {

…

return …;

}

}

1. Crear el paquete **utils** al interior del paquete **model** de la aplicación. Al interior de **utils** crear la clase Ordenamiento que contendrá la implementación de los diferentes algoritmos de ordenamiento. Cada algoritmo recibe una ILista<T> con elementos a ordenar, un Comparator<T> de una clase que define el criterio de comparación alterno y el indicador si el ordenamiento se hace ascendentemente o descendentemente así:

### public final class Ordenamiento<T extends Comparable<T>>

{

### public void ordenarInsercion(ILista<T> lista, Comparator<T> criterio, boolean ascendente)

{

// implementación del algoritmo de Ordenamiento por Inserción

…

}

### public void ordenarShell(ILista<T> lista, Comparator<T> criterio, boolean ascendente)

{

// implementación del algoritmo de Ordenamiento Shell

…

}

}

1. Combinar la ejecución de un algoritmo de ordenamiento (ordenarInsercion) a una sublista de videos (subListaVideos) usando un criterio de comparación alterno para la clase VideoYoutube (comparadorXLikes) de forma ascendente (de menor a mayor por el criterio de comparación) así:

ILista<VideoYoutube> subListaVideos = listaVideos.subLista(100000); Comparator<VideoYoutube> comparadorXLikes = new VideoYoutube.ComparadorXLikes(); Ordenamiento<VideoYoutube> algsOrdenamientoVideos = new Ordenamiento<VideoYoutube>(); algsOrdenamientoVideos.ordenarInsercion(subListaVideos, comparadorXLikes, true);

1. Cronometrar el tiempo de ejecución de una Instrucción y/o método se hace así:

El método System.currentTimeMillis() del paquete java.lang.System retorna el tiempo transcurrido en mili-segundos (mseg). La medición del tiempo de duración en mili-segundos de una instrucción y/o función se realiza de la siguiente forma:

import java.lang.System.\*;

…

long start\_time = System.currentTimeMillis(); # tiempo de referencia inicial (mseg)

Instrucción o funcion a cronometrar

long stop\_time = System.currentTimeMillis(); # tiempo de referencia final (mseg) long elapsed\_time = stop\_time - start\_time; # tiempo transcurrido en mseg

1. Hacer los cambios necesarios en las componentes View, Controller y Model para poder ejecutar las nuevas opciones del menú (lectura de datos, generación de muestra y ejecución de algoritmos de ordenamiento).

## Parte 3 (Trabajo extra-laboratorio) –Resultados de los Algoritmos de Ordenamiento (Documentación de Pruebas)

1. Resultados Teóricos de Complejidad Temporal en el Peor Caso

Complete la tabla de complejidades teóricas en el peor caso para cada algoritmo aplicado a una lista de N datos en representación de Arreglo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Insertion sort | Shell sort | Merge sort | Quick sort |
| Complejidad O( … en el peor caso para una muestra  de N datos | O(n^2) | O(n^2) | O(n log (n)) | O(n^2) |

**Tabla 1. Tabla de Complejidad Temporal Teórica en el peor caso en representación Arreglo.**

1. Ejecución, Pruebas y Obtención de Tiempos de Ejecución de los Algoritmos de Ordenamiento aplicado en Arreglos

Complete la siguiente tabla de tiempos (en mseg) para la ejecución de cada algoritmo de ordenamiento aplicado a diferentes tamaños de muestras de videos representadas en Arreglos.

**Nota**: Solo se deben reportar tiempos de ejecución para muestras donde se obtiene un resultado de ordenamiento en 10 minutos o menos (10 min = 600000 mseg). Para las pruebas donde el tiempo excede de 10 minutos anotar “excedió limite”.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tamaño de la  Muestra | Insertion sort  (tiempos en mseg) | Shell sort  (tiempos en mseg) | Merge sort  (tiempos en mseg) | Quick sort  (tiempos en mseg) |
| 1000 | 2.22 | 1.768 | 0.47 | 0.874 |
| 2000 | 9.48 | 6.51 | 0.8225 | 0.655 |
| 4000 | 28.32 | 7.26 | 1.702 | 1.28 |
| 8000 | 158.63 | 9.09 | 2.27 | 2.74 |
| 16000 | 558.35 | 10.7 | 6.20 | 3.82 |
| 32000 | 2378.93 | 18.89 | 11.08 | 9.978 |
| 64000 | 10369.8762 | 51.89 | 22.421 | 18.83 |
| 128000 | 69947.25 | 150.12 | 47.28 | 48.85 |
| 256000 | Excedió limite | 379.10 | 135.04 | 223.02 |
| 512000(375942) | Excedió limite | 655.07 | 214.98 | 398.21 |

**Tabla 2. Tabla de rendimiento para los distintos algoritmos de ordenamiento en Arreglo.**

Comparando los resultados del tamaño de la muestra más grande donde se lograron obtener los tiempos de ejecución de al menos tres algoritmos, indique el orden de los algoritmos del más eficiente al menos eficiente en Arreglos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Alg. Más eficiente  en tiempo | 2o Alg. Más  eficiente | 3er Alg. Más  eficiente | Alg. Menos  eficiente en tiemp |
| **Algoritmo** | Merge Sort | Quick Sort | Shell Sort | Insertion Sort |

**Tabla 3. Tabla de comparación de algoritmos de ordenamiento en Arreglo.**

1. Ejecución, Pruebas y Obtención de Tiempos de Ejecución de los Algoritmos de Ordenamiento aplicado en Listas Enlazadas

Complete la siguiente tabla de tiempos (en mseg) para la ejecución de cada algoritmo de ordenamiento aplicado a diferentes tamaños de muestras de videos representadas en Listas Enlazadas.

**Nota**: Solo se deben reportar tiempos de ejecución para muestras donde se obtiene un resultado de ordenamiento en 10 minutos o menos (10 min = 600000 mseg). Para las pruebas donde el tiempo excede de 10 minutos anotar “excedió limite”.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tamaño de la muestra | Insertion sort (tiempos en  mseg) | Shell sort (tiempos en mseg) | Merge sort (tiempos en mseg) | Quick sort (tiempos en mseg) |
| 1000 | 574.15 | 32.07 | 25.86 | 26.31 |
| 2000 | 5559.5097 | 188.28 | 58.83 | 146.62 |
| 4000 | 64212.46 | 1228.34 | 255.65 | 875.60 |
| 8000 | Excedió limite | 6240.45 | 1095.28 | 3977.2051 |
| 16000 | Excedió limite | 64000.96 | 4784.21 | 35070.70 |
| 32000 | Excedió limite | 426789 | 20700.92 | 239091.713 |
| 64000 | Excedió limite | Excedió limite | 87543.38 | Excedió limite |
| 128000 | Excedió limite | Excedió limite | 375336.60 | Excedió limite |
| 256000 | Excedió limite | Excedió limite | Excedió limite | Excedió limite |
| 512000(375942) | Excedió limite | Excedió limite | Excedió limite | Excedió limite |

**Tabla 4. Tabla de rendimiento para los distintos algoritmos de ordenamiento en Lista Enlazada.**

Comparando los resultados del tamaño de la muestra más grande donde se lograron obtener los tiempos de ejecución de al menos tres algoritmos, indique el orden de los algoritmos del más eficiente al menos eficiente en Listas Enlazadas.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Alg. Más eficiente  en tiempo | 2o Alg. Más  eficiente | 3er Alg. Más  eficiente | Alg. Menos  eficiente en tiemp |
| **Algoritmo** | Merge Sort | Quick Sort | Shell Sort | Insertion Sort |

**Tabla 5. Tabla de comparación de algoritmos de ordenamiento en Lista Enlazada.**

1. Comparación de los Algoritmos de Ordenamiento usando Arreglos y Listas Enlazadas

Comparando los resultados de un mismo algoritmo para el tamaño de muestra de videos más grande donde se tienen tiempos en Arreglo y en Lista Enlazada, indique la Estructura de Datos donde cada Algoritmo es más eficiente.

|  |  |  |
| --- | --- | --- |
|  | Estructura de Datos Más  eficiente en tiempo | Estructura de Datos Menos  eficiente en tiempo |
| **Insertion sort** | Arreglo dinamico | Lista encadenada |
| **Shell sort** | Arreglo dinámico | Lista encadenada |
| **Merge sort** | Arreglo dinámico | Lista encadenada |
| **Quick sort** | Arreglo dinamico | Lista encadenada |

**Tabla 6. Tabla de comparación de cada algoritmo de ordenamiento de acuerdo a la Estructura de Datos utilizada (Arreglo y Lista Enlazada).**

**Pregunta 1**: A partir de la tabla anterior, en el caso general de ordenamiento ¿Cuál Estructura de Datos es mejor utilizar si solo se tiene en cuenta los tiempos de ejecución de los algoritmos?

1. Preparar Reporte de Resultados Preparar un documento Word (.docx) con:
   * Identificación de los estudiantes que participaron activamente en el desarrollo del laboratorio
   * Incluir los resultados reportados en las tablas 1, 2, 3, 4, 5 y 6.
   * Incluir la respuesta a la pregunta 1.

Colocar este documento en la carpeta **docs** del Proyecto Eclipse/Java del laboratorio 3.

# Lista de Chequeo y Entregable Final

Para entregar exitosamente sus resultados de este laboratorio, recuerde las siguientes indicaciones:

1. Dar acceso a su repositorio del laboratorio al profesor y monitores de su sección.
2. Incluir en el README del repositorio los datos completos de los integrantes del grupo (nombre completo, correo Uniandes y código de estudiante).
3. Incluir el documento Word de resultados en la carpeta docs del proyecto Eclipse/Java del laboratorio 3.
4. Garantizar la actualización de su repositorio del laboratorio en una cuenta Github del grupo donde se incluyan las modificaciones realizadas al proyecto Java y los documentos solicitados.
5. Entregar el enlace (URL) del repositorio por Bloque Neón.
6. Recuerden que cualquier documento solicitado durante las actividades debe incluirse en el repositorio GIT y que solo se calificara hasta el último COMMIT realizado dentro de la fecha límite.