Informe de Laboratorio 04

Tema: Sort y Listas Enlasadas

INFORMACIÓN BÁSICA					
ASIGNATURA:	**Estructura de Datos y Algoritmos**				
TÍTULO DEL TRABAJO:	Sort y Listas Enlazadas				
NÚMERO DE TRABAJO:	04	**AÑO LECTIVO:**	2023-A	**NRO. SEMESTRE:**	III
FECHA DE PRESENTA-CIÓN:	11/06/23	**HORA DE PRESENTA-CIÓN:**	23:59		
INTEGRANTE (s)				**NOTA (0-20)**	
Hidalgo Chinchay, Paulo Andre **Betanzos Rosas, Taylor Anthony** **Villafuerte Ccapira Frank Alexis**					
DOCENTE(s): Mg. Edith Giovanna Cano Mamani					

Tabla 1: Mi tabla extendida

INTRODUCCIÓN

En este trabajo presentaremos el método de ordenamiento por inserción aplicado sobre una lista enlazada normal y doble. Se tratará además la complejidad de éste método de ordenamiento. Usaremos la librería de GNUPlot para graficar una función que represente el tiempo que tarda el ordenador en ejecutar el ordenamiento por cantidad de datos en el peor escenario posible.

MARCO CONCEPTUAL

aqui ira la MARCO CONCEPTUAL

SOLUCIONES Y PRUEBAS

Ejercicio 1

Para dar solución al ejercicio 1 creamos la lista enlazada NodeLink esta tiene un nodo head y un contador de nodos.

Contamos con los métodos InserFist este permitiéndonos insertar en el inicio y el InserLast insertando al final en el caso la lista esta vacía se llamara a InserFist.

```
public void InserFist(E x){
    this.head=new Node<E>(x,this.head);
    this.count++;
}
public void InserLast(E x){
    if(IsEmpty())
        InserFist(x);
    else{
        Node<E> aux=this.head;
        while (aux.getNext()!=null)
            aux=aux.getNext();
        aux.setNext(new Node<E>(x));
        this.count++;
    }
}
```

Contamos con el método IsEmpty este con la capacidad de mostrar que la lista esta vacía o por lo menos tiene un elemento, además tenemos el método toString que nos permite mostrar los elementos de nuestra lista enlazada.

```
public boolean IsEmpty(){
    return this.head == null;
}
@Override
public String toString() {
    String str="";
    for(Node<E> aux= this.head;aux != null;aux=aux.getNext())
        str+= aux.toString()+", ";
    return str;
}
```

Dado que en la información que se nos muestra en el código que nos da a implementar el ejercicio 1 vemos la capacidad de manipular mediante índices.

```
List - Method remove()
public void remove(int indice) {
    if(indice<tamano) {
        if(indice==0)
            raiz=raiz.getNextNode();
        else {
            Node<T> anterior=this.get(indice-1);
            anterior.setNextNode(this.get(indice+1));
        }
        tamano--;
    }
}
```

En nuestra clase main implementamos insertionSort y generarPeorCaso. El método insertionSort solo se ejecutará siempre y cuando la lista tenga por lo menos un elemento dado a que se habilito la capacidad de usar índices modificamos el código dado en la guía del laboratorio para que pueda trabajar con una lista enlazada.

```
public static long insertionSort(NodeLink<Integer> list) {
    int key;
    int i;
    long nano_startTime = System.nanoTime();
    for(int j=1; j<list.getCount(); j=j+1) {
    key = list.get(j).getData();
    i = j-1;
    while(i>-1 && list.get(i).getData()>key) {
        list.get(i+1).setData(list.get(i).getData());
    i = i-1;
    }
    list.get(i+1).setData(key);

    long nano_endTime = System.nanoTime();
    return nano_endTime - nano_startTime;
}
```

Se modifica el método generarPeorCaso para que este retorne una lista enlazada.

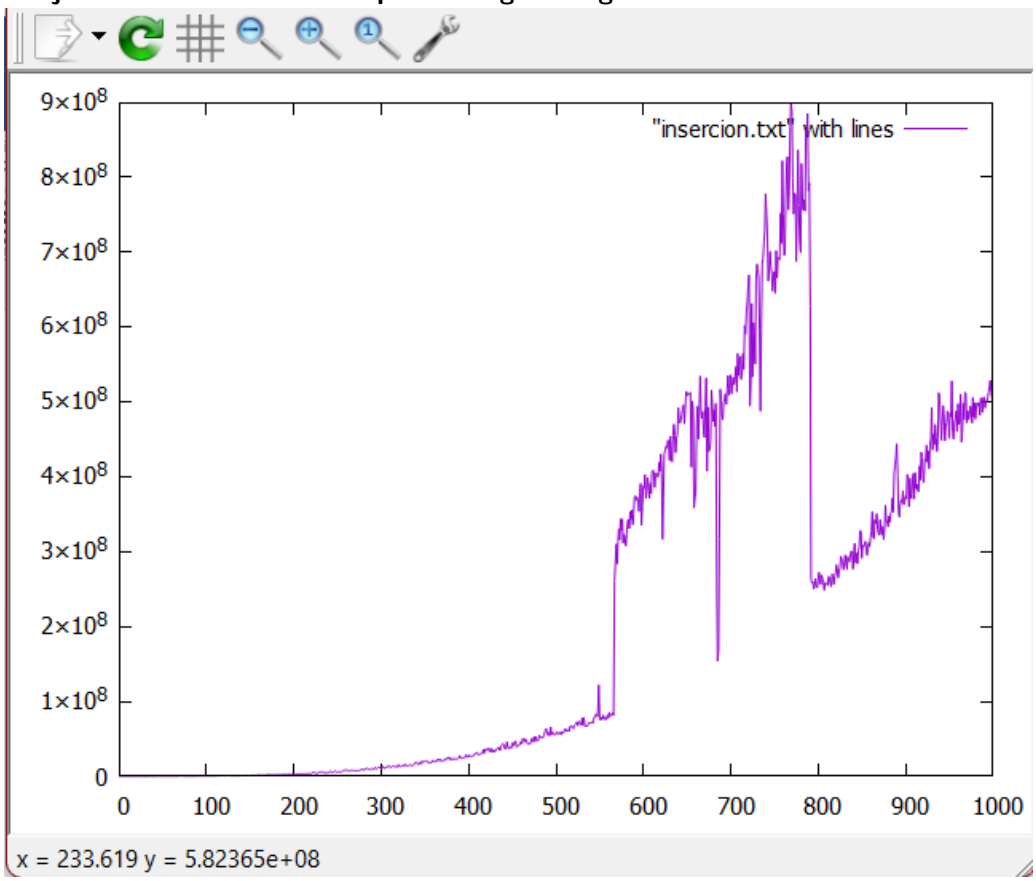
```java
public static NodeLink<Integer> generarPeorCaso(int tamaño) {
    NodeLink<Integer> lista = new NodeLink<>();
    for (int i = 0; i < tamaño; i++)
        lista.InserLast(tamaño - i);
    return lista;
}
```

Se descargó el proyecto java plot de https://javaplot.yot.is/. Donde se agregó el ejercicio 1 como eda_lab4_ej1 importando JavaPlot.

```java
package eda_lab4_ej1;

import com.panayotis.gnuplot.JavaPlot;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;
```

Al ejecutarlo con 1000 nodos quedo el siguiente gráfico.



Ejercicio 2

Para lograr ejecutar el algoritmo de ordenamiento por insercion primero fue necesario crear la clase ListaDoble el cual tenia 2 nodos, uno inicial y otro final. Con esto se lograba recorrer la lista de el fin al inicio y en viceverza.

El metodo addFinal permitia insertar un nodo al final y el otro metodo addInicio, al inicio, dependeiendo si es que estaba vacia o no la lista.

```java
public void addInicio(E x){
    if(!isEmpty()){
        inicio=new Node<>(x,inicio,null);
        inicio.getNextNode().setPreviousNode(inicio);
    }else {
        inicio=fin= new Node<>(x);
    }
}
```

```java
public void addFinal(E x){
    if(!isEmpty()){
        fin=new Node<>(x,null,fin);
        fin.getPreviousNode().setNextNode(fin);
    }else {
        inicio=fin= new Node<>(x);
    }
}
```

Asismimo se agrego el metodo isEmpty para saber si estaba vacia y se sobreescribio el metodo toString para mostrar retornar la data del 1er al ultimo nodo. Los getters y setters se omitieron en la imagen.

```java
public boolean isEmpty() {
    return inicio == null;
}
//getters y setters de inicio y fin
@Override
public String toString() {
    String text="";
    if (!isEmpty()){
        Node<E> aux = inicio;
        while (aux!=null){
            text += aux.getData()+" ";
            aux = aux.getNextNode();
        }
    }
    return  text;
}
```

Se creo la clase Test para implementar el insertionSort y generarPeorCaso. El metodo insertionSort se modifico haciendo que se verificara si es que la lista no estuviera vacia y en caso fuera asi recorria con un for en ves de un while hasta que el nodo siguiente fuera nulo. Dentro de este while habia otro que intercambiaba la data, en este caso Integer, de los nodos haciendo que los menores quedaran al inicio y los mayores al final. Retornaba el tiempo que se demoraba haciendo la operacion.

```java
if (!lista.isEmpty()) {
    keyNode = lista.getInicio().getNextNode();
    while (keyNode != null) {
        Integer key = keyNode.getData();
        currentNode = keyNode.getPreviousNode();
        while (currentNode != null && currentNode.getData() > key) {
            currentNode.getNextNode().setData(currentNode.getData());
            currentNode = currentNode.getPreviousNode();
        }
        if (currentNode == null) lista.getInicio().setData(key);
        else currentNode.getNextNode().setData(key);
        keyNode = keyNode.getNextNode();
    }
}
```

Listing 1: Retorno de insertionSort

```
        return nano_endTime - nano_startTime;
```

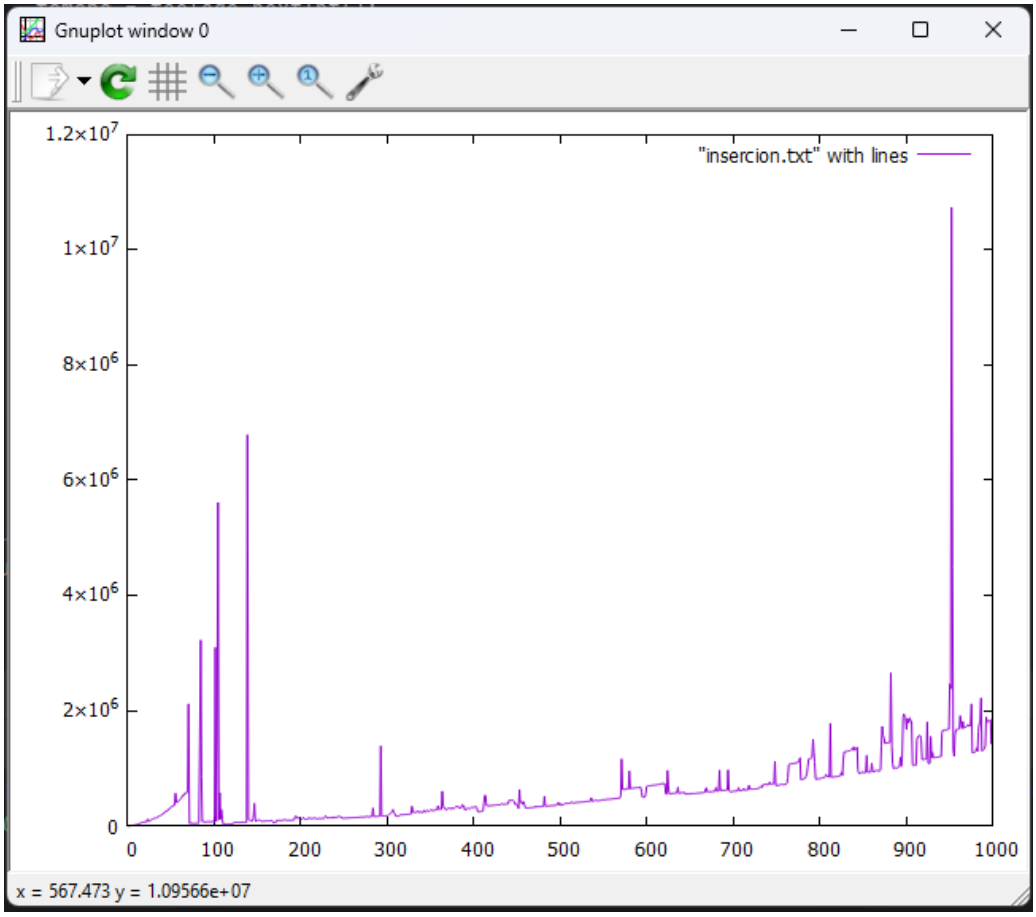
El metodo generarPeorCaso se modifico para que retornara una ListaDoble donde su primer elemento fuera el tamanio-1 y el ultimo 1, donde tamanio era el valor que se recibia por argumento, quedando asi el peor caso para hacer insertionSort.

```java
ListaDoble<Integer> listaDoble = new ListaDoble<>();
for (int i = 0; i < tamanio; i++)
    listaDoble.addFinal(tamanio - i);
return listaDoble;
```

Por ultimo se descargo el proyecto java plot de `https://javaplot.yot.is/`**. Donde se agrego ejercicio 2 como e2 importando JavaPlot con ayuda de [,] para saber coo usarla y agregarlo a mi ide respectivamente.**

```java
package e2;
import com.panayotis.gnuplot.JavaPlot;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;
```

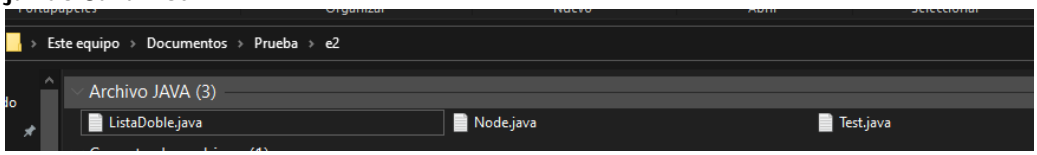
Al ejecutarlo con 1000 nodos quedo el siguiente grafico, donde en el eje x es el numero de nodos y en el y el tiempo en nanosegundos que tomo ordenarlo. Nanosegundos = 10 elevado a -7 segundos.

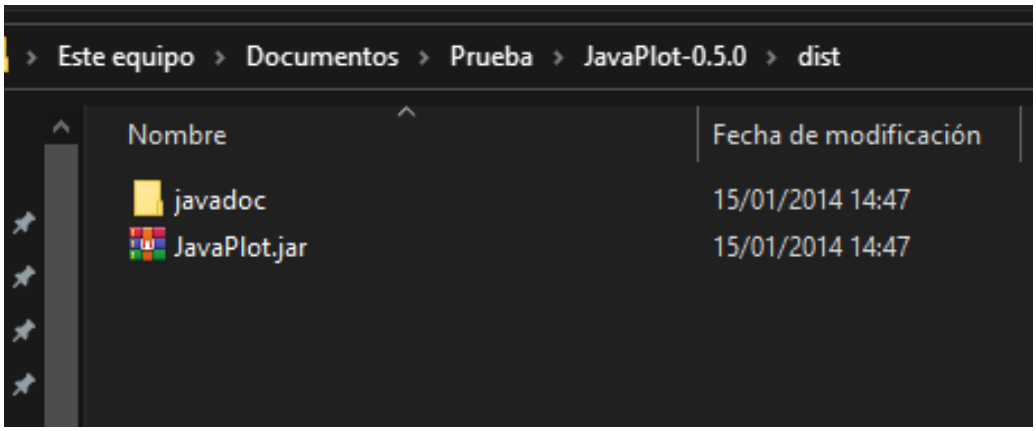


CUESTIONARIO

¿Como se ejecutarían las implementaciones desde terminal?

Para realizar esta tarea necesitaremos tener los archivos java y conocer la ubicacion del archivo jar de JavaPlot.





Abrimos el terminal dentro de la carpeta de los archivos java y los compilamos junto al archivo JavaPlot.jar, el cual buscaremos mediante -cp

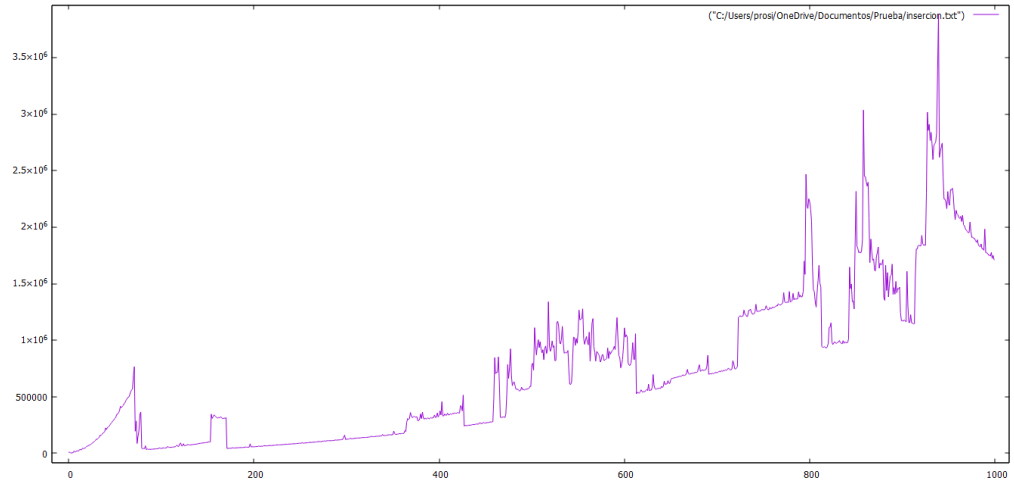
```
C:\Users\prosi\OneDrive\Documentos\Prueba\e2>javac -cp C:\Users\prosi\OneDrive\Documento
s\Prueba\JavaPlot-0.5.0\dist/JavaPlot.jar Node.java ListaDoble.java Test.java

C:\Users\prosi\OneDrive\Documentos\Prueba\e2>
```

Al terminar la compilación, procedemos a ejecutar la clase principal Test. Este proceso se debe realizar en conjunto con el archivo JavaPlot.jar para que pueda obtener las librerías necesarias

```
C:\Users\prosi\OneDrive\Documentos\Prueba\e2>java -cp C:\Users\prosi\OneDrive\Documento
\Prueba\JavaPlot-0.5.0\dist/JavaPlot.jar Test
```

Debería mostrarse el gráfico siguiente (Ejercicio 2)



LECCIONES APRENDIDAS Y CONCLUSIONES

Se aprendio que es el metodo de ordenamiento por insercion, al igual que como implementarlo para Listas simples y dobles. Por otro lado se a crear los peores casos para este ordenamiento y a como guardar los resultados hasta n casos. Con esto se uso JavaPlot para graficar los resultados,como se muestran en el ejercicio 1 y 2.

REFERENCIAS Y BIBLIOGRAFÍA

[]https://www.youtube.com/watch?v=35zTmB9HB6g
[]https://www.youtube.com/watch?v=7wDeHDASoSw
[]https://javaplot.yot.is/example.html