

# Informe de Laboratorio 04

## Tema: Sort y Listas Enlazadas

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 PRESENTACIÓN:	11/06/23	HORA DE PRESENTACIÓN:	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
aquí ira la MARCO CONCEPTUAL
SOLUCIONES Y PRUEBAS
<p>Ejercicio 1</p> <p>Para dar solución al ejercicio 1 creamos la lista enlazada NodeLink esta tiene un nodo head y un contador de nodos.</p> <p>Contamos con los métodos InsertFist este permitiéndonos insertar en el inicio y el InsertLast insertando al final en el caso la lista esta vacía se llamara a InsertFist.</p> <pre> public void InsertFist(E x){     this.head=new Node&lt;E&gt;(x,this.head);     this.count++; } public void InsertLast(E x){     if(IsEmpty())         InsertFist(x);     else{         Node&lt;E&gt; aux=this.head;         while (aux.getNext()!=null)             aux=aux.getNext();         aux.setNext(new Node&lt;E&gt;(x));         this.count++;     } } </pre>

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.

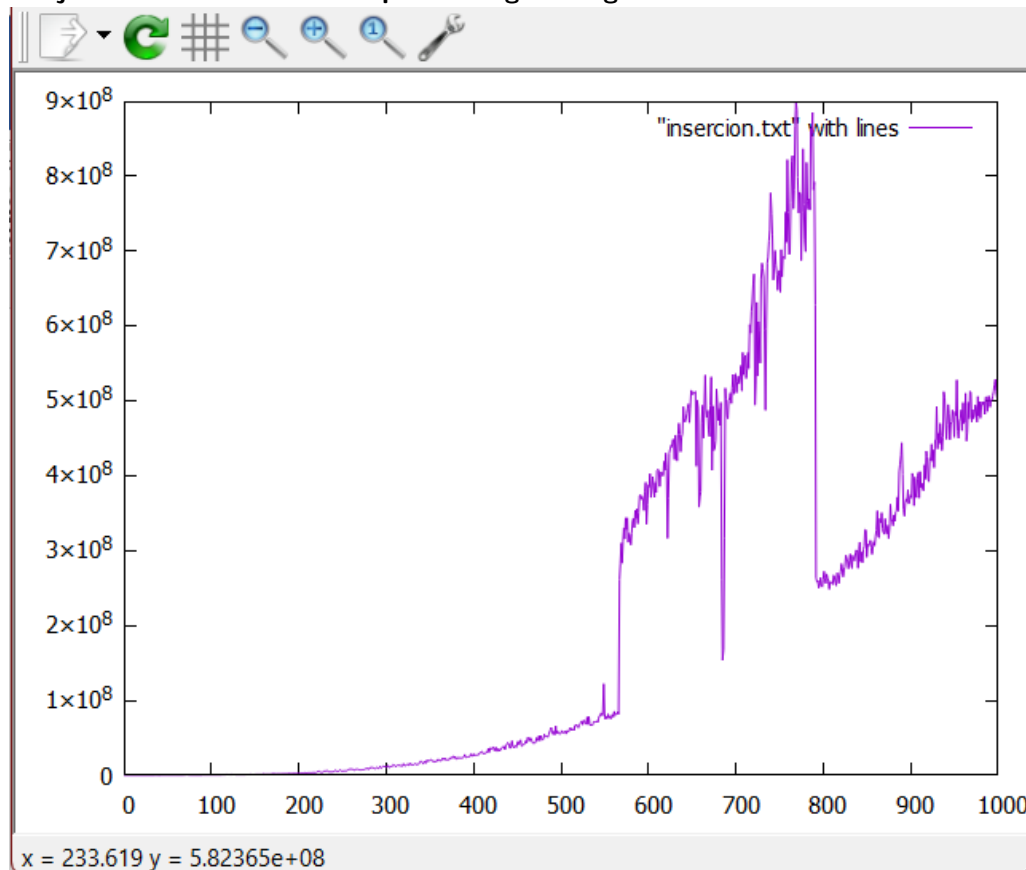
```
public static NodeLink<Integer> generarPeorCaso(int tamaño) {
    NodeLink<Integer> lista = new NodeLink<>();
    for (int i = 0; i < tamaño; i++)
        lista.InsertLast(tamaño - i);
    return lista;
}
```

Se descargó el proyecto java plot de <https://javaplot.yot.is/>. Donde se agregó el ejercicio 1 como eda\_lab4\_ej1 importando JavaPlot.

```
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 viceversa.

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

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

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

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

```
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.

```
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;
```

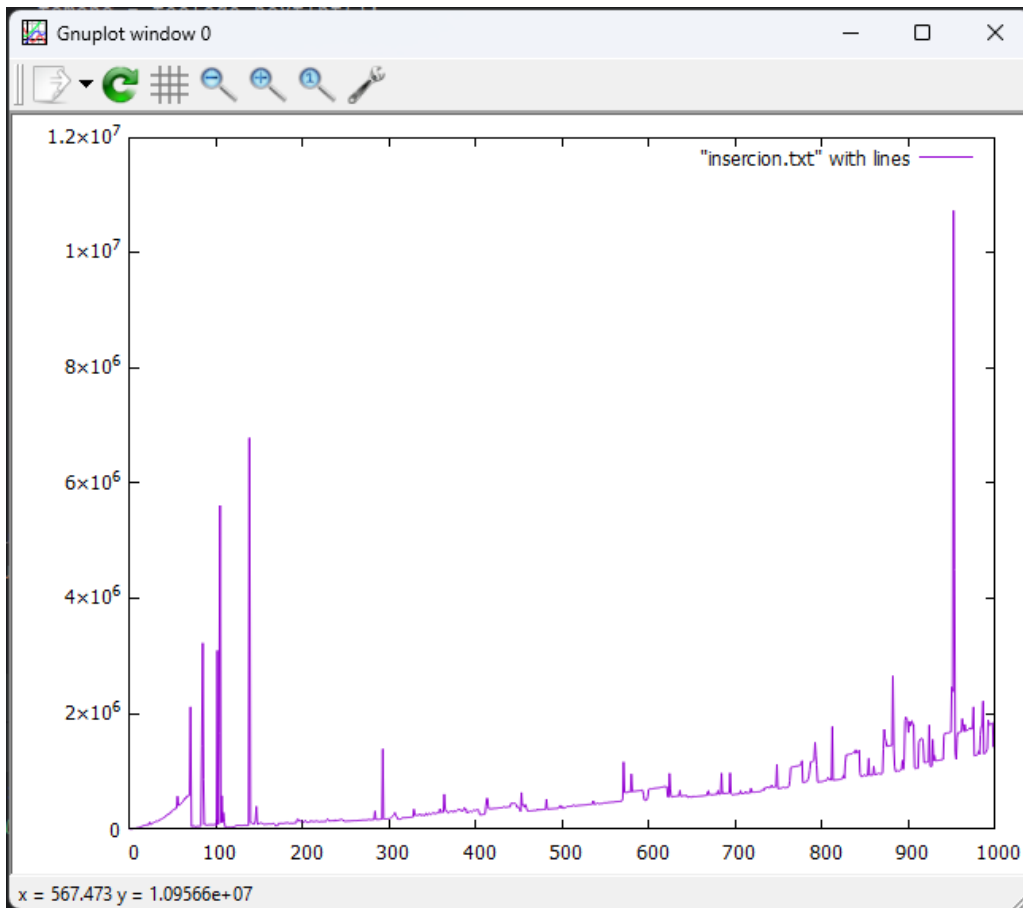
El metodo generarPeorCaso se modifiko para que retornara una ListaDoble donde su primer elemento fuera el tamaño-1 y el ultimo 1, donde tamaño era el valor que se recibia por argumento, quedando asi el peor caso para hacer insertionSort.

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

```
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?

Primero nos dirigimos a la página de JavaPlot y descargamos el archivo necesario

JavaPlot is a pure Java programming interface library for GNUPlot. It can be used as a way to create gnuplot plots on the fly through pure Java commands. In contrast with other common gnuplot java libraries, this library uses java structures to store the various plot parameters, including data-sets. Moreover, is flexible enough to give special parameters to gnuplot, even if the library does not support it (yet). Moreover, it uses Java's Exceptions to inform the user if something went wrong.

**Links**

- [Example](#)
- [Download](#)
- [Documentation](#)

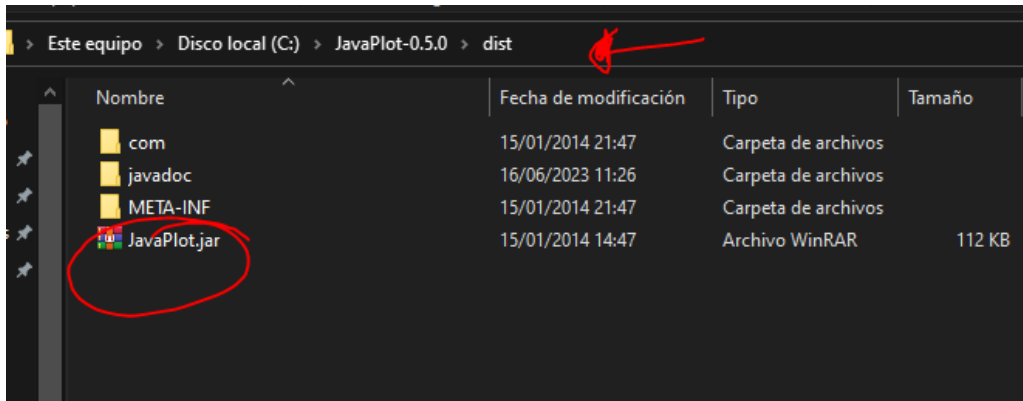
**Requirements**

- Java 1.5 (or better)
- GNUPlot 4.2

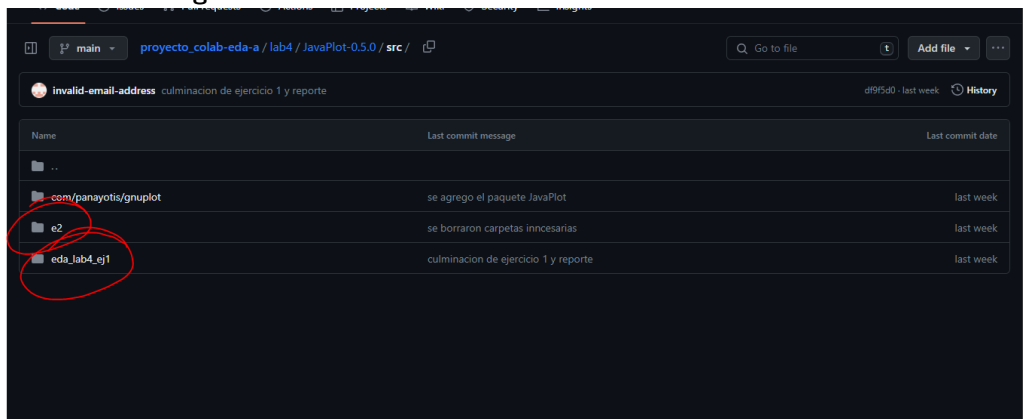
**Tested on the following operation systems:**

- Linux

Luego procedemos a descomprimirlo y ubicarlo en una carpeta. Debemos obtener la dirección del archivo JavaPlot.jar

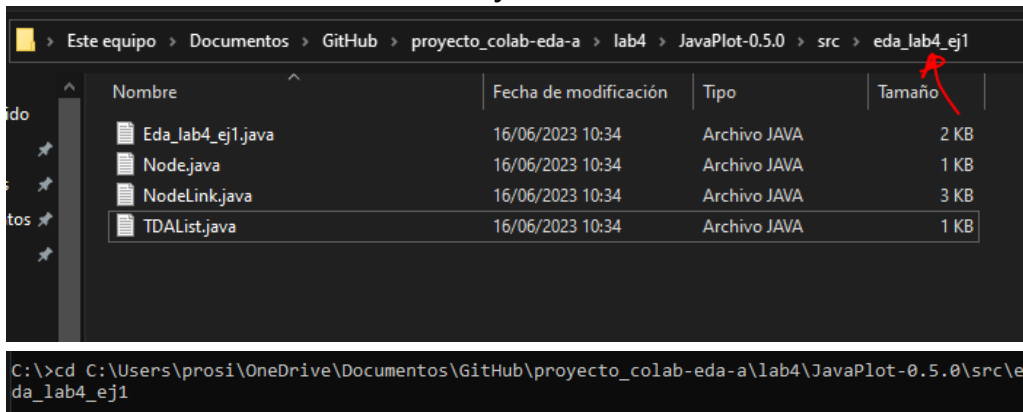


Pasamos a ingresar a Github para obtener los recursos java de los ejercicios resueltos y procedemos a descargarlos.

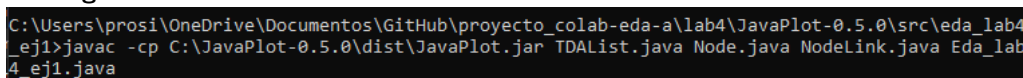


## Ejercicio 1

Obtenemos la ubicación de los archivos y nos ubicamos desde el terminal



Procedemos a compilar los archivos java, además agregamos el classpath del JavaPlot.jar que descargamos antes



Nos muestra un error debido a problemas con la codificación

```
Eda_lab4_ej1.java:17: error: unmappable character (0xF1) for encoding UTF-8
    System.out.print("Introduzca el tama?o m?ximo de la ultima lista: ");
                        ^
Eda_lab4_ej1.java:17: error: unmappable character (0xE1) for encoding UTF-8
    System.out.print("Introduzca el tama?o m?ximo de la ultima lista: ");
                        ^
Eda_lab4_ej1.java:33: error: unmappable character (0xF1) for encoding UTF-8
    public static NodeLink<Integer> generarPeorCaso(int tama?o) {
                                                ^
Eda_lab4_ej1.java:35: error: unmappable character (0xF1) for encoding UTF-8
    for (int i = 0; i < tama?o; i++)
        ^
Eda_lab4_ej1.java:36: error: unmappable character (0xF1) for encoding UTF-8
    lista.InserLast(tama?o - i);
                    ^
Eda_lab4_ej1.java:35: error: not a statement
    for (int i = 0; i < tama?o; i++)
        ^
Eda_lab4_ej1.java:35: error: ';' expected
    for (int i = 0; i < tama?o; i++)
        ^
Eda_lab4_ej1.java:36: error: not a statement
    lista.InserLast(tama?o - i);
    ^
```

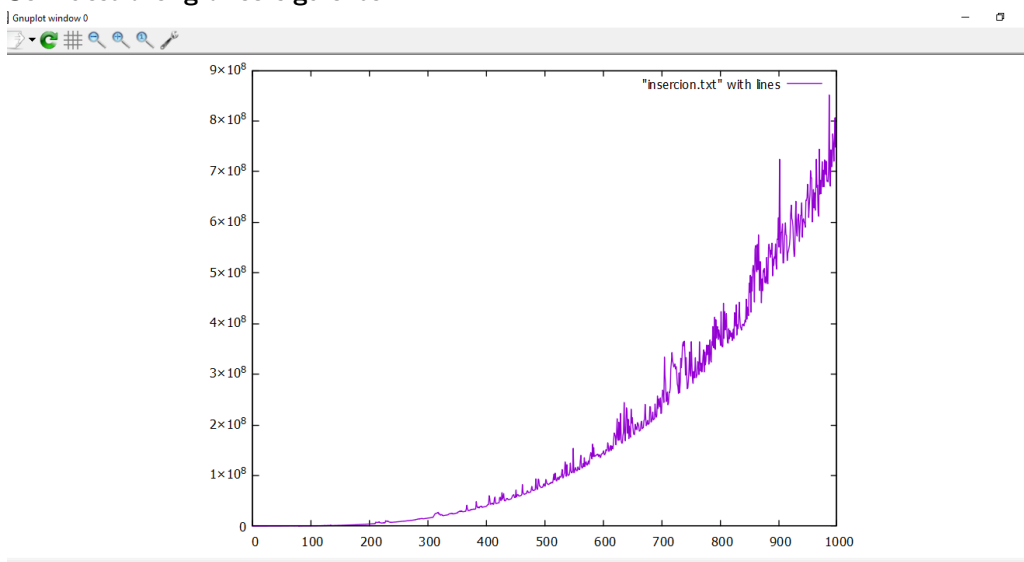
**Repetimos la compilación, esta vez cambiando el sistema de codificación**

```
C:\Users\prosi\OneDrive\Documentos\GitHub\proyecto_colab-eda-a\lab4\JavaPlot-0.5.0\src\eda_lab4_ej1>javac -encoding ISO-8859-1 -cp C:\JavaPlot-0.5.0\dist\JavaPlot.jar TDAList.java Node.java NodeLink.java Eda_lab4_ej1.java
C:\Users\prosi\OneDrive\Documentos\GitHub\proyecto_colab-eda-a\lab4\JavaPlot-0.5.0\src\eda_lab4_ej1>
```

**Ejecutamos el programa. este proceso llama tambien usa classpath para JavaPlot. Retrocedemos una carpeta para llamar al paquete de la clase main**

```
C:\Users\prosi\OneDrive\Documentos\GitHub\proyecto_colab-eda-a\lab4\JavaPlot-0.5.0\src\eda_lab4_ej1>cd ..
C:\Users\prosi\OneDrive\Documentos\GitHub\proyecto_colab-eda-a\lab4\JavaPlot-0.5.0\src>java -cp C:\JavaPlot-0.5.0\dist\JavaPlot.jar; eda_lab4_ej1.Eda_lab4_ej1
Introduzca el tamaño máximo de la ultima lista: 1000
```

**Se muestra el gráfico siguiente**



## Ejercicio 2

**Al igual que el anterior, accedemos a la carpeta y compilamos los archivos llamando a classpath para usar JavaPlot.jar**



Este equipo > Documentos > GitHub > proyecto\_colab-eda-a > lab4 > JavaPlot-0.5.0 > src > e2

Nombre	Fecha de modificación	Tipo	Tamaño
ListaDoble.java	16/06/2023 10:34	Archivo JAVA	2 KB
Node.java	16/06/2023 10:34	Archivo JAVA	2 KB
Test.java	16/06/2023 10:34	Archivo JAVA	3 KB

```
C:\Users\prosi\OneDrive\Escritorio\probar\e2>cd C:\Users\prosi\OneDrive\Documentos\GitHub\proyecto_colab-eda-a\lab4\JavaPlot-0.5.0\src\e2
```

```
C:\Users\prosi\OneDrive\Documentos\GitHub\proyecto_colab-eda-a\lab4\JavaPlot-0.5.0\src\e2>javac -cp C:\JavaPlot-0.5.0\dist\JavaPlot.jar Node.java ListaDoble.java Test.java
```

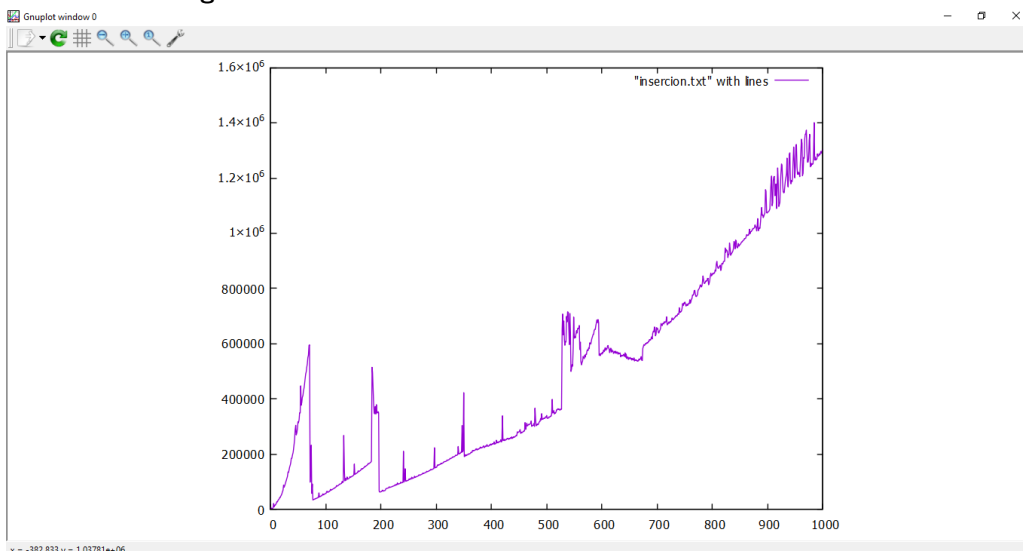
Debemos tener en cuenta que para ejecutar el main en esta situación, debemos retroceder una carpeta, ya que la clase está dentro de un paquete.

```
C:\Users\prosi\OneDrive\Documentos\GitHub\proyecto_colab-eda-a\lab4\JavaPlot-0.5.0\src\e2>cd ..
```

```
C:\Users\prosi\OneDrive\Documentos\GitHub\proyecto_colab-eda-a\lab4\JavaPlot-0.5.0\src>java -cp C:\JavaPlot-0.5.0\dist\JavaPlot.jar; e2.Test
```

Introduzca el tamaño máximo de la ultima lista doble: 1000

Nos muestra el gráfico



## LECCIONES APRENDIDAS Y CONCLUSIONES

Se aprendió 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>