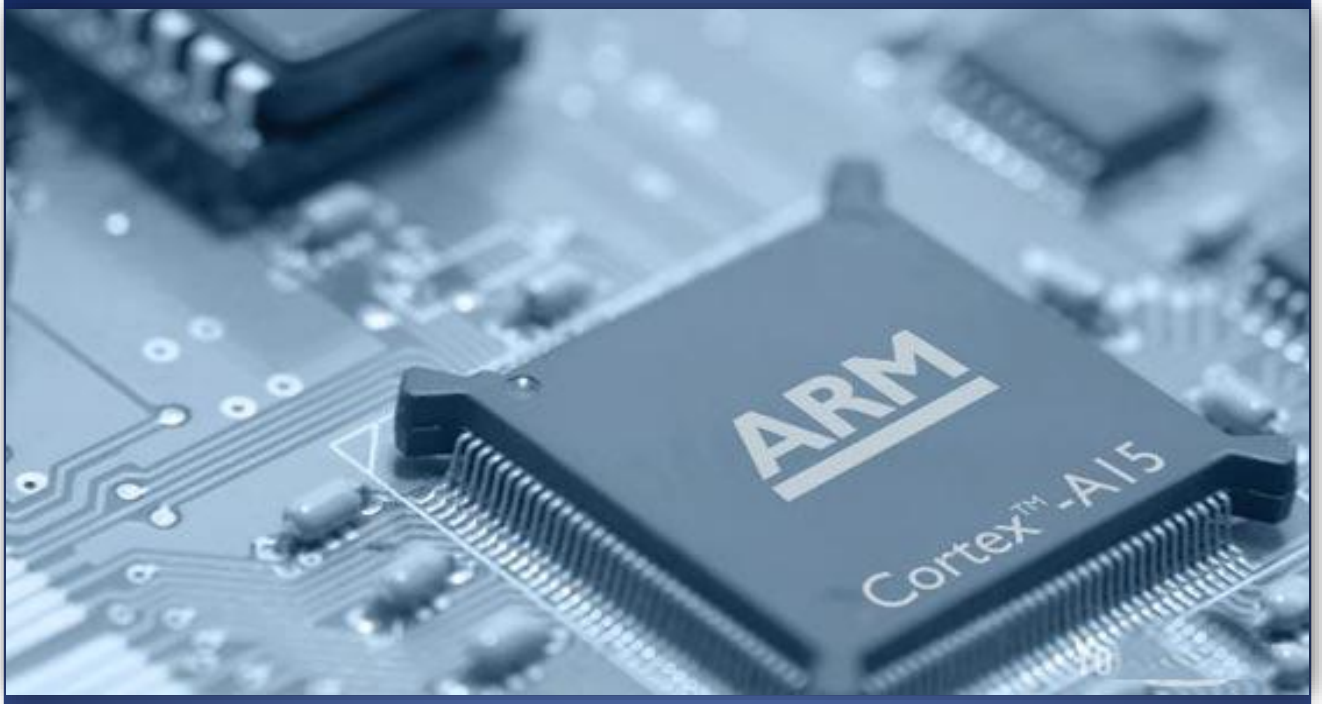
2014 -I




# WRITE-UP
# Proyecto Primer Parcial
# Grupo # 8

ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORES

**INTEGRANTES**

- José Romero Triviño
- Víctor Rodríguez Cabrera
- Fausto Mora Velásquez

# INDICE

## Contenido

**DESCRIPCION**.....2

**Parte 1: Generar Archivo de Números Aleatorios**.....2

**Parte 2: Ordenar los Grupos de Números Aleatorios del Archivo** .....4

**Parte 3: Interpretar los tiempos de los ordenamientos**.....9

**BIBLIOGRAFIA**.....11

# DESCRIPCION

Nuestro proyecto del primer parcial de la materia de Organización y Arquitectura de Computadores constaba de 4 literales que resumiremos en pocas palabras. Básicamente lo que se pedía en el proyecto era crear un programa en el que se pudiera realizar las siguientes acciones: Generar Archivo de Números Aleatorios, y Ordenar los Grupos de Números Aleatorios del Archivo utilizando algoritmos de ordenamiento. Adicional a esto se pidió también que los tiempos que se tomó en realizar el ordenamiento de los grupos sean interpretados mediante gráficas, además todas las acciones mencionadas anteriormente debían poder ser escogidas por el usuario mediante un menú.

Cada una de las partes del proyecto son detallas a continuación:

## Parte 1: Generar Archivo de Números Aleatorios

La primera era que se pueda generar un archivo de nombre **"aleatorios.txt"** con 10 grupos de números aleatorios que estuvieran en un rango entre el 0 y el 1000. En el menú de nuestro programa dicha acción está representada por la opción: *Generar Randoms*.

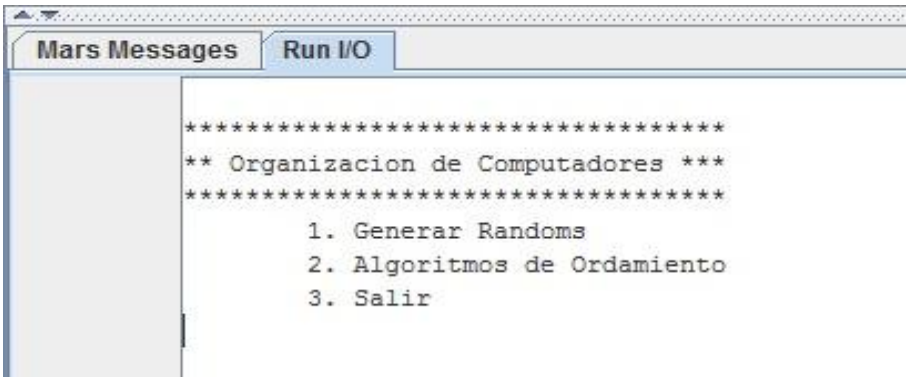



**Figura 1.a: Menú Principal del Programa.**

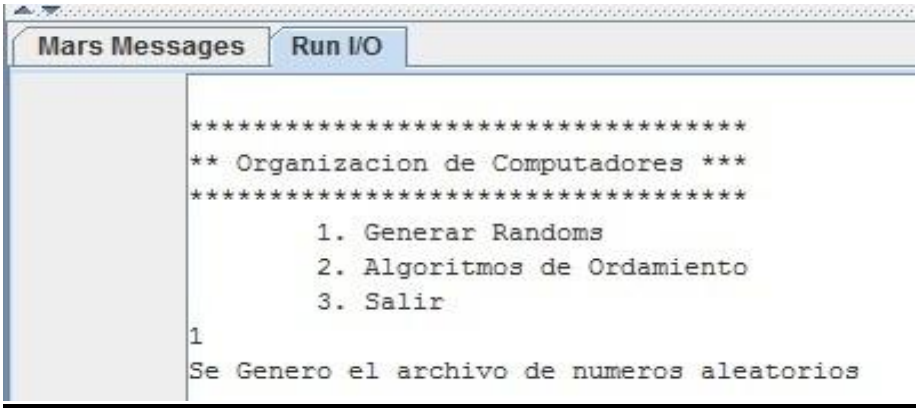



**Figura 1.b : Menú Principal luego de seleccionar la opción 1.**

El formato que escogimos para nuestro archivo que contiene los grupos de números aleatorios separamos los grupos por corchetes, y los números de cada grupo los separamos por comas. A continuación se muestra el formato:

[num1_grupo1,num2_grupo1,...,num10_grupo1][num1_grupo2,num2_grupo2,….,num20_grupo2]……..…[num1_grupo10,num2_grupo10,……, num400_grupo10].

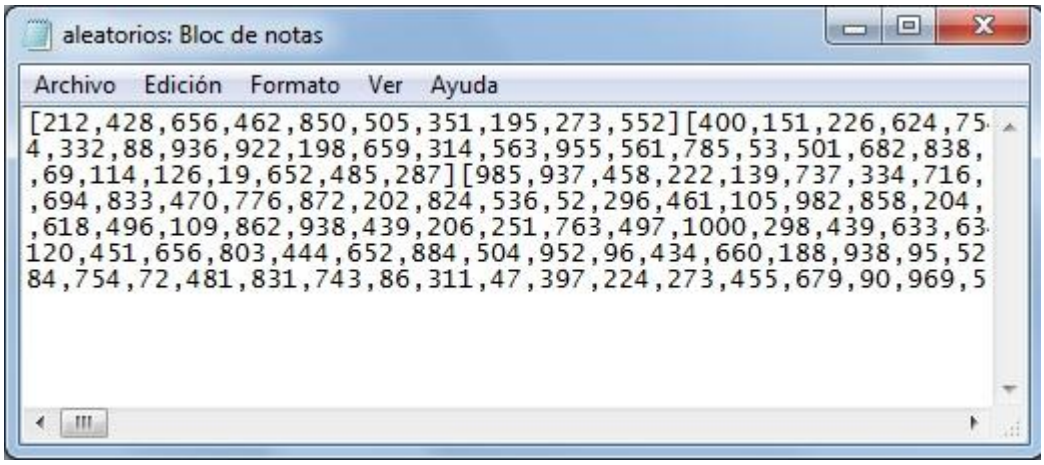



**Figura 1.c: Archivo "aleatorios.txt" generado.**

## Parte 2: Ordenar los Grupos de Números Aleatorios del Archivo

La segunda parte consistía en que el archivo generado en la primera parte debería poder ser leído por el programa, para posteriormente poder ordenar cada uno de los 10 grupos de números aleatorios, esto utilizando un algoritmo de ordenamiento seleccionado por el usuario. Los algoritmos que se pidió que estuvieran disponibles para ser seleccionados por el usuario son los siguientes: Bubble Sort, Insertion Sort, y Quick Sort.

Una vez ordenados los grupos de números se debían generar dos archivos: uno que contenga los grupos ya ordenados, y otro que contenga los tiempos que se tomó hacer el ordenamiento seleccionado por el usuario.

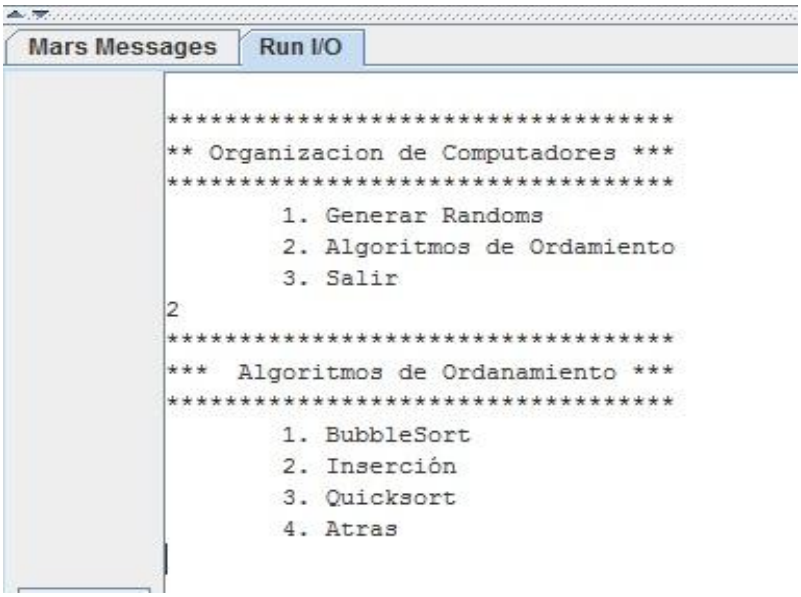



**Figura 2.a : Menú Principal luego de seleccionar la opción 2.**

Luego de seleccionar en el menú principal la opción 2, nos aparecerá un nuevo menú donde podremos seleccionar el ordenamiento que queramos.

```
**********************************
***  Algoritmos de Ordanamiento ***
**********************************
        1. BubbleSort
        2. Inserción
        3. Quicksort
        4. Atras
1




Ordenamiento por BubbleSort Completado, Se han generado los archivos bur_aleatorios.txt y bur_tiempos.t
**********************************
***  Algoritmos de Ordanamiento ***
**********************************
        1. BubbleSort
        2. Inserción
        3. Quicksort
        4. Atras
```

**Figura 2.b: Menú Algoritmos de Ordenamiento luego de seleccionar la opción 1.**

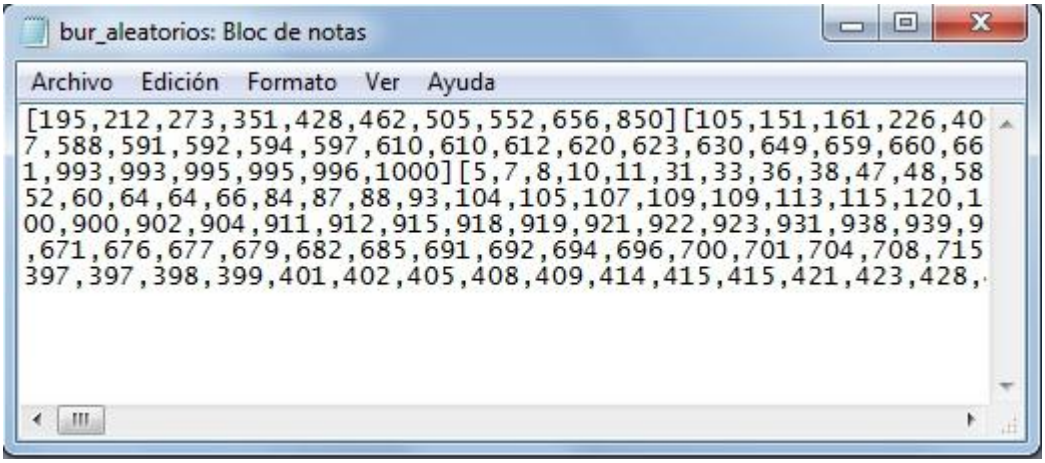



**Figura 2.c : Archivo "bur_aleatorios.txt" generado**

En el formato que escogimos para el archivo que contienen los tiempos separamos el tiempo de cada grupo con una barra vertical.

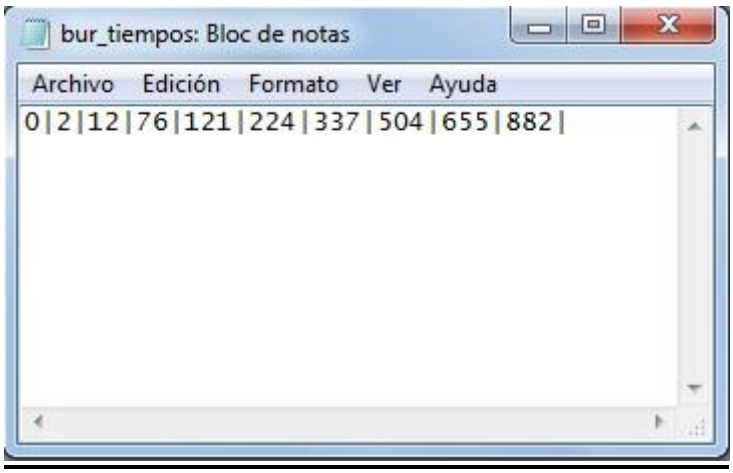



**Figura 2.d : Archivo "bur_tiempos.txt" generado**

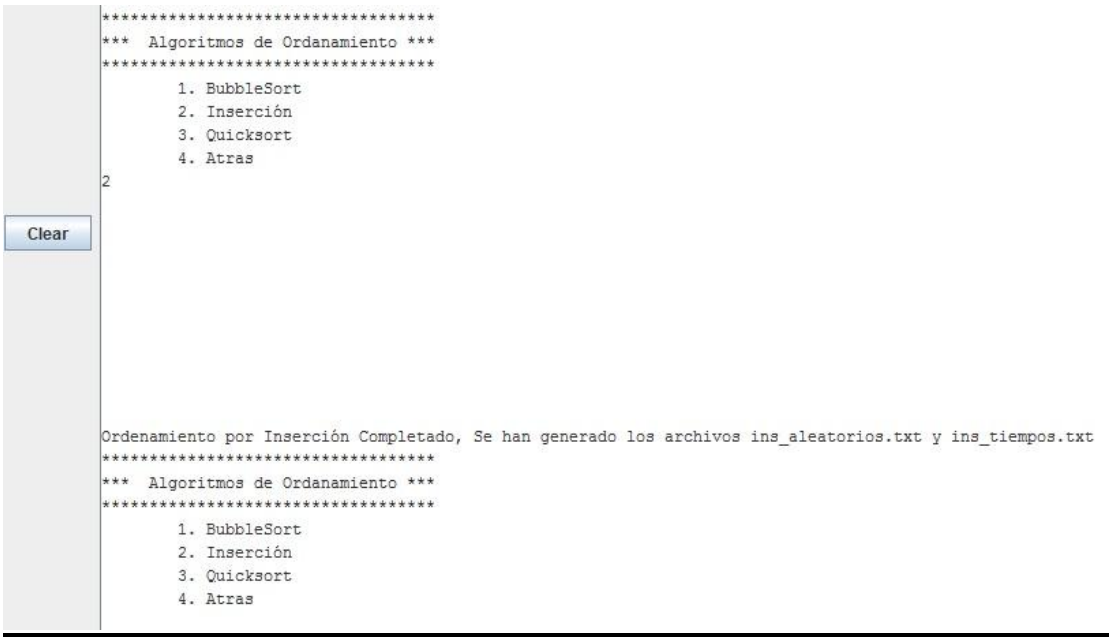



**Figura 2.e : Menú Algoritmos de Ordenamiento luego de seleccionar la opción 2.**

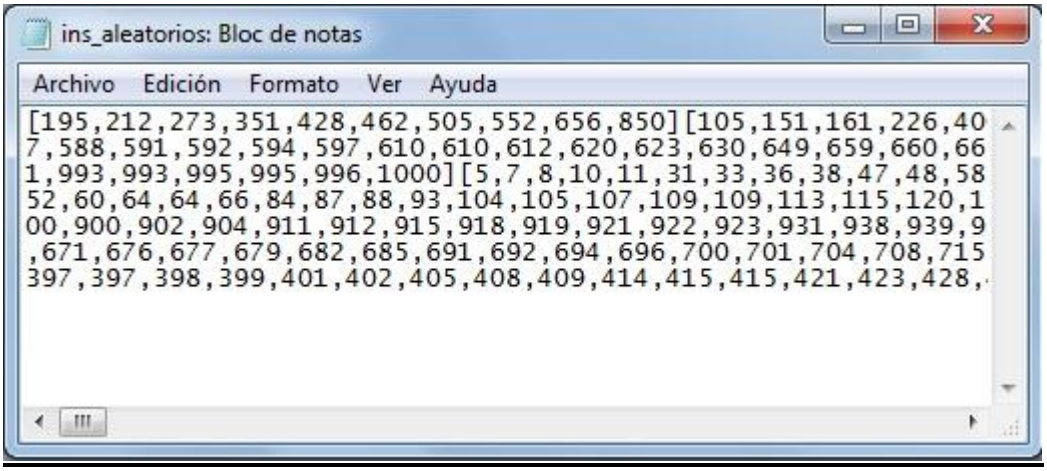



**Figura 2.f : Archivo "ins_aleatorios.txt" generado.**

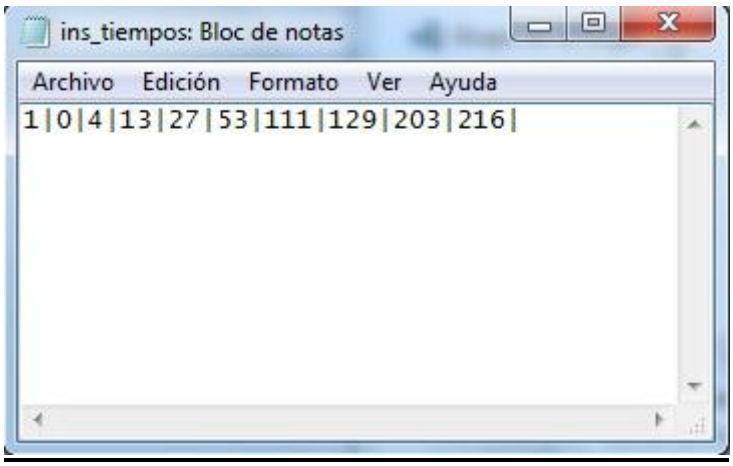



**Figura 2.g : Archivo "ins_tiempos.txt" generado.**

```
**********************************
***  Algoritmos de Ordanamiento ***
**********************************
        1. BubbleSort
        2. Inserción
        3. Quicksort
        4. Atras
3



Ordenamiento por QuickSort Completado, Se han generado los archivos qui_aleatorios.txt y qui_tiempos.txt
**********************************
***  Algoritmos de Ordanamiento ***
**********************************
        1. BubbleSort
        2. Inserción
        3. Quicksort
        4. Atras
```

**Figura 2.h : Menú Algoritmos de Ordenamiento luego de seleccionar la opción 3.**

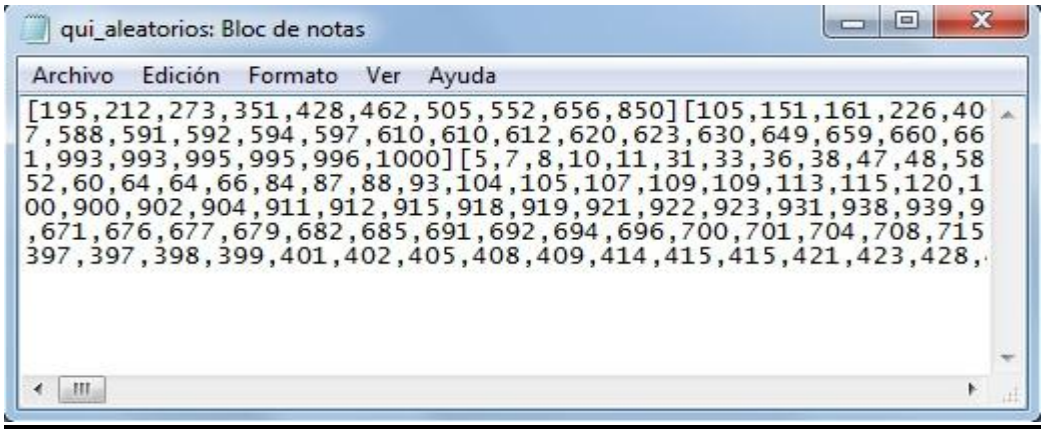



**Figura 2.i : Archivo "qui_aleatorios.txt" generado.**

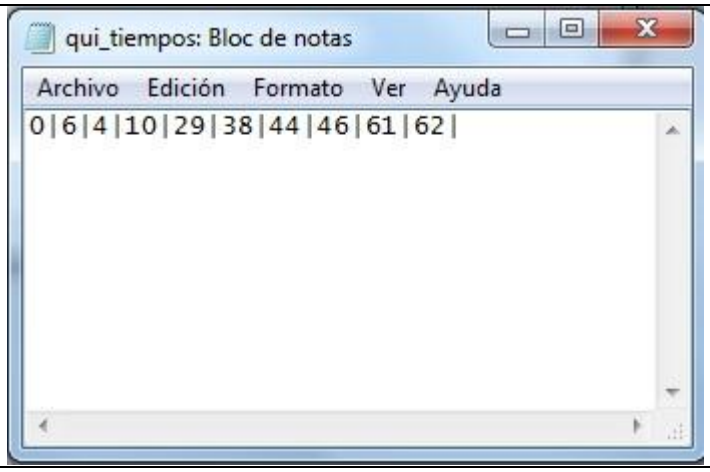



**Figura 2.j : Archivo "qui_tiempos.txt" generado.**

## Parte 3: Interpretar los tiempos de los ordenamientos

Como parte final del proyecto se solicitó que los datos de tiempo guardados en los distintos archivos, sean interpretados mediante gráficas, utilizando algún software que permita modelar los datos obtenidos, para eso utilizamos **Scilab 5.4.1** que un software libre y de código abierto para calculo numérico que proporciona un entorno de desarrollo de gran alcance para aplicaciones de ingeniería y científicas.

Para ello, por comodidad y facilidad en el análisis de los datos, desarrollamos un script en Scilab llamado **graficador.sci** que se encargue de realizar las gráficas requeridas. La graficación se la realiza a través de la función **`plottingFiles([vector])`** el cual recibe como parámetro un vector con los nombres de los archivos a analizar, tal como se visualiza en la **Figura 3.a.**

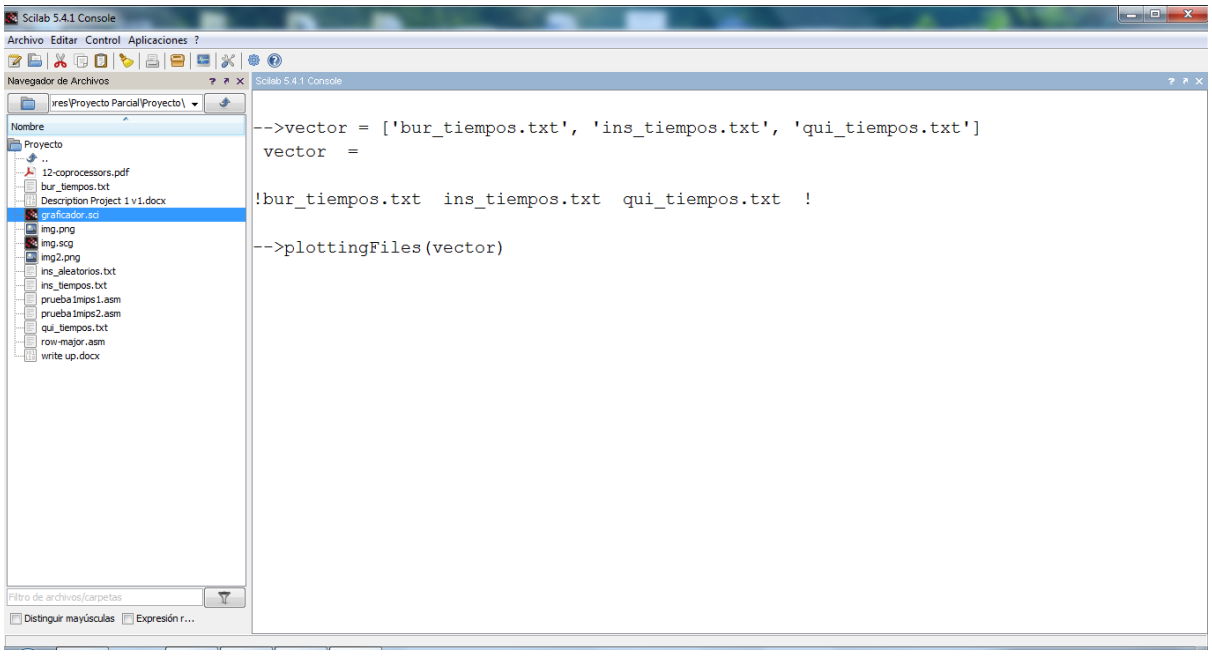



**Figura 3.a : Ejecución del Script en Scilab 5.4.1**

Al ejecutar el script, nos mostrará una ventana con la graficas solicitadas.

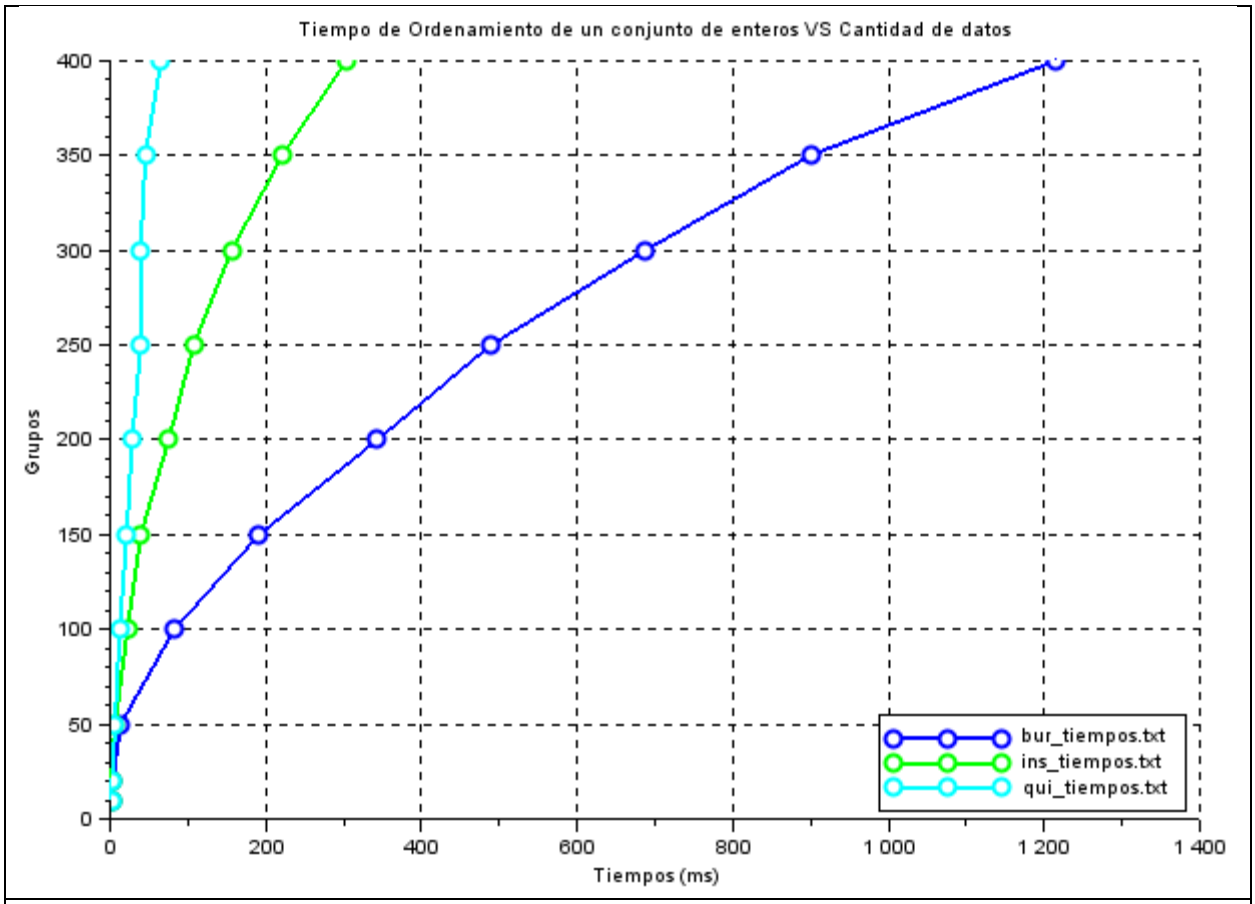



**Figura 3.b : Grafica Tiempo de Ordenamiento vs Cantidad de datos**

Como podemos apreciar en la gráfica **Figura 3.a** existe una notoria diferencia de los tiempos entre los algoritmos estudiados, pero la diferencia más clara es del algoritmo de Bubble Sort y es porque que ocupa la mayor cantidad de tiempo en el ordenamiento de los datos, para un mismo conjunto de datos en comparación a los otros 2 algoritmos medidos.

En cambio, si comparamos entre el algoritmo de Insertion Sort y Quick Sort, la diferencia en el ordenamiento para el mismo conjunto de datos es muy poca pero para conjuntos de datos pequeños, esto no sucede para conjuntos de datos mucho más grande, la cual su diferencia en el tiempo de ordenamiento es mucho mas evidente.

Además, según esta gráfica, el algoritmos que ocupa el menor tiempo en el ordenamiento de los datos, es el algoritmo de Quick Sort ya que la diferencia de los tiempos para diferentes conjuntos de datos es muy poca en comparación a Bubble Sort que tiene una mayor diferencias de los tiempos para cada conjunto diferente de datos.

Finalmente podemos decir que de los algoritmos más eficiente **(más rápido)** para el ordenamiento de un conjunto discreto de datos, es el algoritmo de Quick Sort debido a que la diferencia del tiempo en ordenar el conjunto es poca y el algoritmo de Bubble Sort es el menos eficiente **(más lento)** debido a que a diferentes conjuntos de datos, tiene una tendencia exponencial en el tiempo de ordenamiento es similar.

# BIBLIOGRAFIA


**Convertir de Entero a String**

- http://www.daniweb.com/software-development/assembly/code/435631/integer-to-string-in-mips-assembly

**Bubble Sort**

- http://codecellar.99k.org/BubbleSortMIPS.html

**Quick Sort**

- http://www.cs.uni.edu/~fienup/courses/copy-of-computer-organization/homework-solutions/hw6f98/hw6f98.mips

**Scilab IDE 5.4.0**

- http://www.scilab.org/download/5.4.1