# 6620. Organización de Computadoras
# Trabajo Práctico 0:
# Infraestructura Básica

González, Juan Manuel, *Padrón Nro. 79.979*
juan0511@yahoo.com

Argüello, Osiris, *Padrón Nro. 83.062*
osirisarguello@yahoo.com.ar

Paez, Ezequiel Alejandro, *Padrón Nro. 84.474*
skiel85@gmail.com

1er. Cuatrimestre de 2012
66.20 Organización de Computadoras – Prática Jueves
Facultad de Ingeniería, Universidad de Buenos Aires

**Resumen**

El presente trabajo tiene como objetivo familiarizarse con las herramientas de software que serán usadas en los siguientes trabajos, implementando un programa (y su correspondiente documentación) que resuelva el problema piloto planteado, una implementación de los algoritmos de ordenamiento *Mergesort* y *Selectionsort* en lenguaje C. A su vez, se utilizará el programa GXemul para simular un entorno de desarrollo en una máquina MIPS, a fin de obtener el código en Assembly MIPS32 de la implementación realizada. Finalmente, procederemos a realizar mediciones para evaluar las posibilidades de mejora y el desempeño relativo entre ambos algorimos, utilizando los programas *time* y *gprof*.

# 1. Introducción

Al comenzar a utilizar nuevas herramientas, en cualquier ámbito, es necesaria una breve introducción al funcionamiento de las mismas: tener una noción de las prestaciones que ofrecen, asi también como de sus limitaciones.
Como primer objetivo en la materia, nos proponemos adentrarnos en el funcionamiento del emulador GXemul. Nuestra meta será emular una plataforma MIPS DECstation 5000/200 (ejecutando un sistema operativo NetBSD), para poder desde allí desarrollar programas en lenguaje C. Estos serán compilados y ejecutados haciendo uso de la herramienta GCC (GNU Compiler Collection), mediante el cual también será posible obtener, a posteriori, el código MIPS32 del programa.
Una vez cumplido este objetivo, experimentaremos con los programas *time* y *gprof*, a fin de evaluar las posibilidades de mejora y el desempeño relativo entre ambas implementaciones.
Como último punto, aprenderemos los rudimientos de LaTeX para generar la documentación relevante al trabajo práctico.

# 2. Programa a implementar

El problema a resolver consiste en un implementar programa que realice ordenamiento o *sorting* utilizando los algorimos de *mergesort* o *selectionsort*, según elija el usuario.

El programa debe leer los datos de entrada desde *stdin* o bien desde uno o más archivos. La salida del programa debe imprimirse por *stdout*, mientras que los errores deben imprimirse por *stderr*. El algoritmo de ordenamiento puede seleccionarse mediante las opciones **-m** o **-s** para *mergesort* o *selectionsort* respectivamente.

# 3. Consideraciones sobre el desarrollo

A continuación se detallan las consideraciones tenidas en cuenta para el desarrollo del trabajo práctico. Hemos decidido separar las que surgen del Diseño de las que surgen en la implementación.

## 3.1. Consideraciones de Diseño

El programa a desarrollar consta a grandes rasgos de un parser encargado de leer los parámetros de entrada, y las funciones que realizan la lectura de los archivos y las funciones encargadas de realizar el ordenamiento mediante los algoritmos solicitados. Dada la complejidad y extensión de algunas de las funciones desarrolladas, se decidió separar el programa en una serie de archivos, a modo de mejorar su legibilidad y organización. A continuación se presenta la lista de archivos que componen el proyecto:

| Archivo | Descripción |
|---|---|
| tp0.c | Función main e intérprete de línea de comandos. |
| manejoes.h | Encabezado correspondiente a 'manejoes.c'. |
| manejoes.c | Parser de entrada y función de escritura de salida. |
| sort.h | Encabezado correspondiente a 'sort.c'. |
| sort.c | Implementación de *mergesort* y *selectionsort*. |
| deferrores.h | Definición de los códigos de error internos del programa. |

El diseño de la aplicación se realizó pensando en varias «capas». Una capa es encargada de analizar y validar opciones de ejecución, otra capa, la encargada de realizar las operaciones principales de conversión y otras más pequeñas encargadas de emitir mensajes de error o información de ayuda en pantalla para el usuario.

Dado que la aplicación puede tomar parámetros variables de ejecución o adoptar valores por defecto, se realizó un análisis completo de todas las opciones que un usuario puede ingresar, aceptando aquellas que fueren válidas y rechazando las que no, con el consiguiente mensaje de error que el usuario visualizará en pantalla. En caso de no ser especificado un algoritmo de ordenamiento, el programa asumirá la opción **-m**, correspondiente a *mergesort*.

La aplicación fue implementada en lenguaje ANSI C, es decir que está garantizada la compilación exitosa ante la utilización de la opción *-ansi* en GCC.

La siguiente capa importante, es la correspondiente a los algoritmos de ordenamiento. Se desarrollaron dos algoritmos de ordenamiento: *mergesort* y *selectionsort*. La implementación para ambos algoritmos básicamente consta de procesos similares, ligeramente diferenciados en el punto en que se invoca una función ANSI C distinta de acuerdo al tipo de ordenamiento que se esté realizando.

A grandes rasgos, la solución intenta leer el *stream* de entrada de datos que es especificado por el usuario. Puede constar de varios archivos o directamente ser la entrada estándar. De no resultar satisfactoria esta operación, ya sea porque el archivo no existe o se produjere un fallo en el disco en el momento de la lectura, se retorna de inmediato el correspondiente código de error. Si pueden tomarse los datos de entrada, el programa procede a su almacenamiento, si se trata de varios archivos estos son concatenados. Para resolver el almacenamiento de los datos se diseñó un tipo de dato especial, que consta de un char* y dos variables de control, a fin de ir reservando memoria en forma dínamica, a medida que es requerida. Como último paso, una vez que se cuenta con los datos cargados, se continua la ejecución realizando el ordenamiento indicado por el usuario, volcando el resultado de dicha operación por la salida estándar o *stdout*. Dado que el programa debe funcionar con archivos binarios, las lecturas y escrituras se realizan mediante las funciones *fread* y *fwrite*.

Por otro lado, se diseñó el programa de modo que todas las salidas por *stdout* o *stderr* se encuentren en la función main o en funciones dedicadas a ese propósi-

to. En el caso del mensaje de ayuda y el de versión del programa, se codificaron dos funciones específicas, mientras que los mensajes de error son mostrados justo antes de finalizar la ejecución del programa, en la función main. Para lograr esto, se definieron códigos de error internos al programa que son devueltos a main para su interpretación por cada una de las funciones que son llamadas. Dichos códigos, definidos en el archivo 'deferrores.h', son:

2: ERROR_RESERVA_INICIAL_MEMORIA
3: ERROR_RESERVA_MEMORIA
4: ERROR_ARCHIVO_ENTRADA

La ejecución del programa se interrumpe por cualquiera de estos errores, mostrando al usuario un mensaje afin. Los códigos 0 y 1 son utilizados para la devolución al SO del resultado de la ejecución, siendo:

0: Ejecución sin problemas.
1: Error de ejecución.

El procesamiento e interpretación de los comandos ingresados desde la consola es manejado mediante la biblioteca 'getopt', evitando así la implementación de un parser para este fin. Dado que el enunciado presenta un mensaje de ayuda en idioma inglés, se han codificado todas las salidas en este idioma. Asimismo, los mensajes de error fueron formateados para imitar la salida de 'getopt', con el fin de presentar los errores en forma consistente.

Finalmente, cabe destacar que todas las impresiones del programa son ruteadas por *stdout*, y los mensajes de error a través de *stderr*.

## 3.2. Consideraciones de Implementación

Cabe destacar que para implementar el trabajo práctico se partió de la totalidad del programa implementado en lenguage C (se ha respetado el estándar ANSI) y luego se procedió a traducir a assembly de MIPS el código final, con el uso de GCC.

### 3.2.1. Portabilidad de la solución

El programa está diseñado para poder ser ejecutado en diferentes plataformas, como por ejemplo NetBSD (PMAX) y la versión de Linux usada para correr el simulador, GNU/Linux (i686). El haber codificado el proyecto en lenguaje ANSI C garantiza que pueda ser compilado en ambas plataformas sin problemas, dotando al programa de un grado mínimo de portabilidad.
Por otro lado, se destaca que se entrega junto con el presente informe dos versiones del código fuente, una en formato C y la otra en código assembly MIPS.

Las principales funciones y estructuras en nuestra implementación son las siguientes:

#### 3.2.2. Cadena de bytes *tDynArray*

Para implementar la cadena de bytes que utiliza el programa desarrollado (*tDynArray*), se ha utilizado un arreglo en memoria dinámica, que crece reservando memoria automáticamente al agrandarse la cadena cuando es necesario.

#### 3.2.3. procesarEntrdada

Esta función se encarga de tomar los archivos de entrada (o *stdin* según corresponda), abrirlos y operar con el *tDynArray* para ir guardando el contenido de todos los archivos especificados por el usuario. Los archivos se van abriendo dentro de un bucle *for*, y la lectura se realiza mediante la función *fread*.

#### 3.2.4. procesarSalida

Esta función tiene como objetivo presentar el resultado del ordenamiento, para lo que utiliza la función *fwrite*. La salida es únicamente por salida estándar o *stdout*.

#### 3.2.5. mergeSort

Este método toma el array de caracteres leido de los archivo, lo divide en 2 partes e invoca recursivamente la funcion **mergeSort** sobre cada parte. Como ultimo paso fusiona ambas partes, ordenandolas.

#### 3.2.6. selectionSort

Este método recorre el array de caracteres, buscando el menor caracter en cada iteración y ubicándolo al principio del array.

## 4. Generación de ejecutables y código assembly

Para generar el ejecutable del programa, debe correrse la siguiente sentencia en una terminal:

```
$ gcc -Wall -o tp0 tp0.c manejoes.c sort.c
```

Para generar el código MIPS32, debe ejecutarse lo siguiente:

```
$ gcc -Wall -S -mrnames tp0.c manejoes.c sort.c
```

Nótese que para ambos casos se han activado todos los mensajes de 'Warning' (-Wall). Además, para el caso de MIPS, se han habilitado otras dos banderas: '-S' que detiene al compilador luego de generar el assembly y '-mrnames' que tiene como objetivo generar la salida utilizando los nombres de los registros en vez de los números de registros.

# 5. Corridas de prueba

En esta sección se presentan algunas de las distintas corridas que se realizaron para probar el funcionamiento del trabajo práctico.

En primer lugar se mostró el mensaje de ayuda, y luego se imprimió la versión del programa:

```
juan0511@VM15:~/Desktop/TP0_SVN/tp0v2/src$ ./tp0 -h
tp0 [OPTIONS] [file...]
-h, --help       display this help and exit.
-V, --version    display version information and exit.
-m, --merge      use mergesort algorithm.
-s, --sel        use selectionsort algorithm.

juan0511@VM15:~/Desktop/TP0_SVN/tp0v2/src$ ./tp0 -V
TP0: v2.0.
```

Luego, se ejecutó la prueba provista por la cátedra en el enunciado del trabajo práctico:

```
juan0511@VM15:~/Desktop/TP0_SVN/tp0v2/src$ echo -n "9876543210" > digits.txt
juan0511@VM15:~/Desktop/TP0_SVN/tp0v2/src$ ./tp0 -s digits.txt
0123456789juan0511@VM15:~/Desktop/TP0_SVN/tp0v2/src$ cat letters.txt
aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZjuan0511@VM15:~/Desktop/TP0_
SVN/tp0v2/src$ ./tp0 < letters.txt
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzjuan0511@VM15:~/Desktop/TP0_
SVN/tp0v2/src$ ./tp0 letters.txt digits.txt
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzjuan0511@VM15:~/De
sktop/TP0_SVN/tp0v2/src$
```

```
juan0511@VM15:~/Desktop/TP0_SVN/tp0v2/src$ head -c 64 /dev/urandom > random.txt
juan0511@VM15:~/Desktop/TP0_SVN/tp0v2/src$ hexdump -C random.txt
00000000  54 7f 39 cf 20 35 b4 07  20 08 2b 79 31 d1 98 22  |T.9. 5.. .+y1.."|
00000010  33 8c 8b c7 82 ae 1e f7  d4 5f 8c 09 a2 ee 41 58  |3........_....AX|
00000020  ba 72 40 b6 20 bb ec e7  2f 3d 8b 6e ab 43 05 5d  |.r@. .../=.n.C.]|
00000030  a8 5b 5d 29 2f 20 e7 e0  2e 59 e4 5d fb 80 2c c7  |.[])/ ...Y.]..,.|
00000040
juan0511@VM15:~/Desktop/TP0_SVN/tp0v2/src$ ./tp0 < random.txt > sorted.txt
juan0511@VM15:~/Desktop/TP0_SVN/tp0v2/src$ hexdump -C sorted.txt
00000000  80 82 8b 8b 8c 8c 98 a2  a8 ab ae b4 b6 ba bb c7  |................|
00000010  c7 cf d1 d4 e0 e4 e7 e7  ec ee f7 fb 05 07 08 09  |................|
00000020  1e 20 20 20 20 22 29 2b  2c 2e 2f 2f 31 33 35 39  |.    ")+,.//1359|
00000030  3d 40 41 43 54 58 59 5b  5d 5d 5d 5f 6e 72 79 7f  |=@ACTXY[]]]_nry.|
00000040
juan0511@VM15:~/Desktop/TP0_SVN/tp0v2/src$
```

Finalmente, se presenta un ejemplo más mostrando alguna de las salidas posibles, obtenida para casos en los trabaja con archivos binarios y cuando se intenta abrir un archivo inexistente:

```
juan0511@VM15:~/Desktop/TP0_SVN/tp0v2/src$ ./tp0 digits.txt text.in
tp0: could not read input file `text.in'.
juan0511@VM15:~/Desktop/TP0_SVN/tp0v2/src$ head -c 1048576 /dev/urandom > random
1M.in
juan0511@VM15:~/Desktop/TP0_SVN/tp0v2/src$ wc -c random1M.in
1048576 random1M.in
juan0511@VM15:~/Desktop/TP0_SVN/tp0v2/src$ ./tp0 -m random1M.in | wc -c
1048576
juan0511@VM15:~/Desktop/TP0_SVN/tp0v2/src$
```

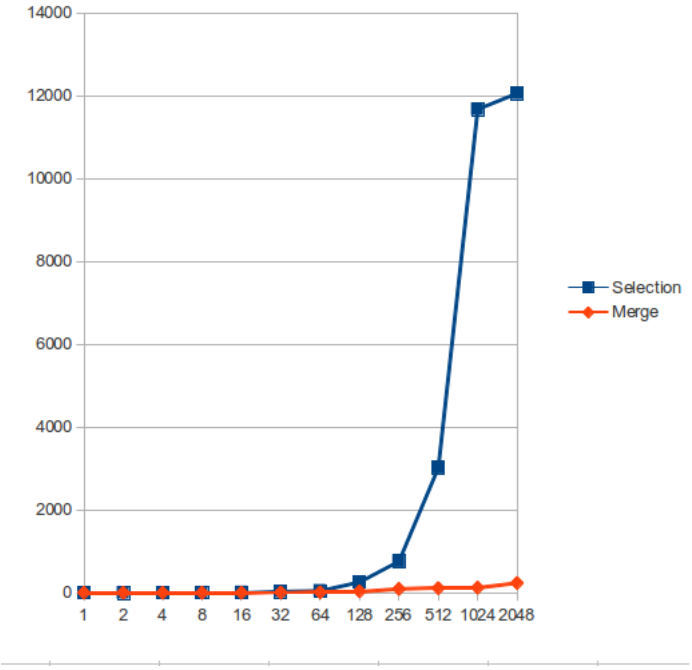
# 6. Mediciones

Se comparó la performance de cada uno de los algoritmos realizando pruebas con el comando time. Se tomó el tiempo insumido por cada uno de los algoritmos tomando archivos de diferentes tamaños y de diferente aleatoridad (completamente aleatorio, aleatorio ordenado y aleatorio inversamente ordenado).

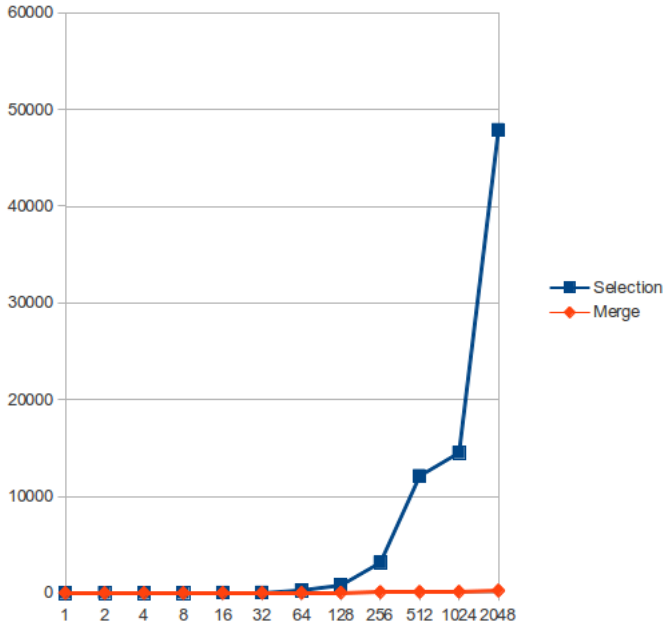
Los archivos completamente aleatorios fueron generados leyendo bytes de /dev/urandom de un sistema linux. Los archivos de prueba aleatorios ordenados se generaron ordenando los archivos anteriores con el propio programa desarrollado. Y los archivos inversamente ordenados fueron generados invirtiendo estos últimos archivos con un simple algoritmo hecho en C.

Se realizaron gráficos comparativos para cada una de las situaciones con curvas que representan a cada algoritmo, representando el tiempo insumido (milisegundos) en relación con el tamaño del archivo de entrada.

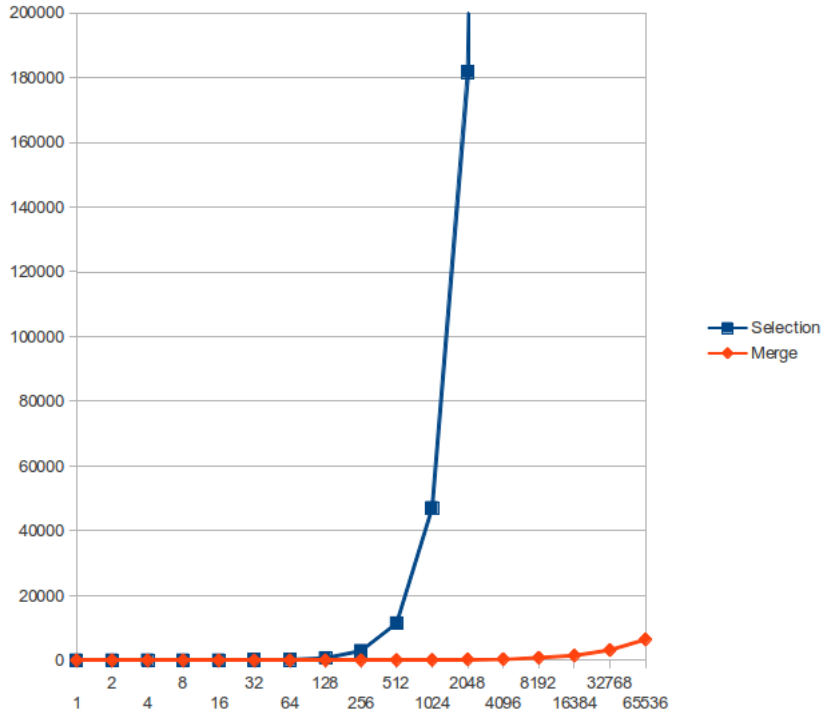
Este gráfico representa los dos métodos para el caso de un *array* ya ordenado:




Este gráfico representa los dos métodos para el caso de un *array* en orden inverso:



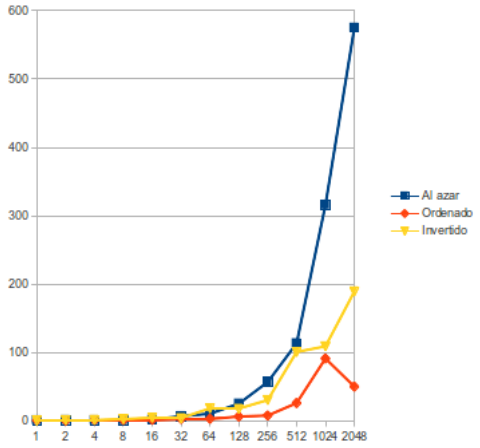

Este gráfico representa los dos métodos para el caso del promedio de diez *arrays* con datos tomados al azar:




Observamos en estos gráficos como la curva de crecimiento del tiempo insumido del *selectionsort* es casi exponencial, mientras que la curva del *mergesort*

es mucho más estable.

También se presenta un gráfico que muestra el speedup de la mejora de pasar de *selectionsort* a *mergesort* para cada una de las tres situaciones. Observamos en este gráfico que la mejora es muy grande para arreglos de gran tamaño, pero para arreglos pequeños casi no hay diferencia:



# 7. Profiling

Con el objetivo de proponer una mejora en la implementación realizada, se utilizó el programa *gprof* para obtener la distribución de tiempos durante la ejecución de la aplicación. Se presentan dos tablas con los resultados más significativos, para cada algoritmo de ordenamiento:

## 7.1. Mergesort

| percent | cumulative seconds | self seconds | calls | self s/call | total s/call | name |
|---|---|---|---|---|---|---|
| 96.74 | 3.26 | 3.26 | 1 | 3.26 | 3.26 | mergeSort |
| 3.26 | 3.37 | 0.11 | 1 | 0.11 | 0.11 | procesarEntrada |
| 0.00 | 3.37 | 0.00 | 1 | 0.00 | 0.00 | imprimirSalida |

Utilizo un archivo con 6553600 bytes para que el procesamiento dure tiempo suficiente para notar las diferencias de tiempo entre cada método y minimizar el impacto del tiempo de lectura de disco.

En esta prueba el tiempo total insumido fue de 3,37 segundos.

La mayor parte del tiempo se la lleva la ejecución del método merge con un 97 % del tiempo (3,26 segundos), y por eso es el que conviene optimizar. Si este método fuera optimizado al punto de costar un tiempo despreciable, el tiempo total del programa sería de 0,11 segundos, entonces el speedup mÃ¡ximo sería de 3,37/0,11 = 30,6.

## 7.2. Selectionsort

La distribución de tiempos para este algoritmo es:

| percent | cumulative seconds | self seconds | calls | self s/call | total s/call | name |
|---|---|---|---|---|---|---|
| 100.00 | 268.50 | 268.50 | 1 | 268.50 | 268.50 | selectionSort |
| 0.00 | 268.50 | 0.00 | 203999 | 0.00 | 0.00 | swap |
| 0.00 | 268.50 | 0.00 | 1 | 0.00 | 0.00 | imprimirSalida |
| 0.00 | 268.50 | 0.00 | 1 | 0.00 | 0.00 | procesarEntrada |

Fue analizado con un archivo de 204800 bytes por las mismas razones que elegimos el archivo del mergeSort. Pero en este caso es un archivo más pequeño porque el selectionSort es más lento. El método selectionSort es el que conviene optimizar, ya que es el que se lleva casi el 100 % del tiempo. Si este método se optimizara al punto de ser despreciable el tiempo que consume, el speedup máximo tendería a infinito.

# 8. Código fuente C y MIPS32

Tanto el código C como el código MIPS32 que fue generado en el trabajo práctico ha sido incluido en el CD entregado junto al presente informe.

# 9. Conclusiones

El trabajo práctivo motivo de este informe ha presentado al equipo diversos desafíos. En primer lugar cabe mencionar la adaptación a un ambiente basado en GNU/Linux, no dominado por todos sus integrantes en igual manera. También es destacable la dificultad inicial que trajo la correcta configuración del ambiente virtual utilizado para emular la máquina MIPS, aunque esto fue paliado en gran parte por la documentación y explicaciones provistas por la cátedra. Finalmente, también fue invertida una cantidad considerable de tiempo en aprender a utilizar e investigar sobre LaTeX, ya que si bien permite obtener muy buenos resultados, se requiere de mucha lectura para poder aprovechar todo su potencial. En este sentido, el modelo de informe suministrado también fue muy util para dar los primeros pasos.
A su vez, hemos comprobado la utilidad de utilizar la ley de Amdahl para comparar la performance de algoritmos, y su utilidad para comprender como realizar optimizaciones de los mismos, utilizada en conjunto con utilidades como *time* y *gprof*.
La ley de Amdahl en este caso no es de mucha ayuda porque los algoritmos implementados constan de una sola función y es obvio que todo el peso recae sobre la función de ordenamiento. Pero en algoritmos más complicados con más funciones, el profiling nos puede ayudar a determinar cual función es la más pesada y entonces si poner todo el esfuerzo en optimizarla.
Por las razones expuestas, consideramos muy necesario un trabajo práctico introductorio de esta naturaleza, sin gran dificultad algorítmica o teórica pero justamente orientado a nivelar e introducir los elementos a utilizar en las demás actividades prácticas que se desarrollarán en el curso.
Como conclusión final, podemos considerar que hemos logrado obtener un buen manejo de las herramientas introducidas en este primer proyecto.

# Referencias


[1] Merge-sort, http://en.wikipedia.org/wiki/Merge_sort

[2] Selection-sort, http://en.wikipedia.org/wiki/Selection_sort

[3] *time* man page, http://unixhelp.ed.ac.uk/CGI/man-cgi?time

[4] GNU *gprof*, http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html

[5] GXemul, http://gxemul.sourceforge.net/

[6] The NetBSD Project, http://www.netbsd.org/

[7] Kernighan, Brian W. / Ritchie, Dennis M. El Lenguaje De Prorgamación C. Segunda Edición, PRENTICE-HALL HISPANOAMERICANA SA, 1991.

[8] Oetiker, Tobias, "The Not So Short Introduction To LaTeX2", http://www.physics.udel.edu/∼dubois/lshort2e/