Trabajo Práctico 1
66.20 Organización de Computadoras

Nicolás Calvo, *Padrón Nro. 78.914*
nicolas.g.calvo@gmail.com
Nicolás Cisco, *Padrón Nro. 94.173*
n.cis_92@hotmail.com
Rodrigo Burdet, *Padrón Nro. 93.440*
rodrigoburdet@gmail.com

2do. Cuatrimestre de 2013

Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

En el presente se comparan dos algoritmos de ordenamiento: *Bubblesort*[1] vs *Heapsort*[2]. Bubblesort fue solo implementado en C mientras que Heapsort fue hecho en MIPS32 y en C. El parámetro de comparación es el tiempo de ordenamiento de archivos de texto. Todos los archivos y códigos fuente aquí mencionados, así como también el presente informe, pueden ser descargados del repositorio de trabajo [1].

[1] URL del Repositorio: https://github.com/NickCis/orga20132c/tree/master/tp1

1. Introducción

El ordenamiento por montículos (heapsort en inglés) es un algoritmo de ordenamiento no recursivo, no estable, con complejidad computacional $\Theta(n \log n)$. Este algoritmo consiste en almacenar todos los elementos del vector a ordenar en un montículo (heap), y luego extraer el nodo que queda como nodo raíz del montículo (cima) en sucesivas iteraciones obteniendo el conjunto ordenado. Basa su funcionamiento en una propiedad de los montículos, por la cual, la cima contiene siempre el menor elemento (o el mayor, según se haya definido el montículo) de todos los almacenados en él. La Ordenación de burbuja (Bubble Sort en inglés) es un sencillo algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada. Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas". También es conocido como el método del intercambio directo. Dado que solo usa comparaciones para operar elementos, se lo considera un algoritmo de comparación, siendo el más sencillo de implementar. Éste algoritmo es esencialmente un algoritmo de fuerza bruta lógica siendo su complejidad algorímica de $\Theta(n^2)$.

2. Objetivos

En el presente trabajo práctico se debe desarrollar una aplicación que implementa los algoritmos de ordenamiento Bubblesort y Heapsort en el lenguaje programación C y, en particular, el algoritmo Heapsort en assembler de arquitectura MIPS 32. Luego, utilizando dicha aplicación se debe ordenar una serie de archivos a fin de medir el tiempo de ejecución que insume.
Se calculó el speedup al reemplazar el algoritmo Bubblesort por el algoritmo Heapsort como así también al sustituir la versión del algoritmo de Heapsort implementado en C por el algoritmo implementado en assembler.
Con todo esto, lo que se busca es:
- Familiarizarse con la arquitectura MIPS 32 y su lenguaje ensamblador.
- Comparar el rendimiento de los distintos algoritmos de ordenamiento.
- Para Heapsort, comparar el rendimiento entre la versión codificada en lenguaje C y la codificada en lenguaje assembler.

3. Compilación

Para compilar el programa se uso un *make* que será presentado en el apéndice para automatizar el proceso. Se optó por usar el standard 99

std99[**?**] trantando a los warnings como errores y sin usar optimizaciones **-O0**. A saber:

- Antes de cada ejecución hacer : **make clean**
- Para compilar el código assembly se puede hacer: **make assembly**
- Para compilar el codigo en C se puede hacer: **make**

4. Programa

El programa consta de 3 secciones:

- Parser de argumentos (TDA)
- Métodos de ordenamiento
  - Bubblesort
  - Heapsort
- Main

Cada una de ellas sera explicada a continuación:

4.1. Parser

Para crear el TDA parseador de argumentos se le pasa la cantidad máxima de argumentos que el programa puede llegar a recibir, internamente se guarda un vector de estructuras *TArg*. Cada uno de estos *TArg* es un posible argumento, esta estructura guarda una función parseadora de char* provenientes de argv a un valor, un char representando el nombre corto (e.g: o de -o), un char* representando el nombre largo (e.g: output de –output), un void* representando un puntero a un valor por defecto, y dos valores que se asignan después de parsear argv (si es que se realiza algún match), un entero para saber si se encontró el valor y otro char* correspondiente al string del argv que se usó como argumento (e.g: -o output.pgm, se guardará output.pgm). En el .h se describe que hacen las primitivas, a modo de resumen, se carga el array de argumentos usando *ParseArg_addArg*. Despues, se llama a *ParseArg_parse(argc, argv)* que recorre el vector argv buscando los parámetros. Para obtener el valor de un argumento, se llama a *ParseArg_getArg(nombre_corto)* siendo nombre corto un char, esto devuelve un puntero al dato o null, internamente, se buscó en el vector de *TArg* aquel con nombre corto igual al pasado por argumento .Si $arg \rightarrow encontre == 0$, se devuelve el valor por defecto, sino, si $arg \rightarrow func == NULL$ se devuelve un 1 casteado a void*, si $arg \rightarrow buf == NULL$ se devuelve NULL, sino, se devuelve $arg \rightarrow func(arg \rightarrow buf)$ (buf siempre es una cadena de caracteres finalizada en 0).

4.2. Main

El main consta de todas las funciones auxiliares que son necesarias para los ordenadores. Estos necesitan una lista de palabras que serán provistas por *fillWords*. A su vez, esta, necesita todas las palabras del archivo en un solo buffer, tarea de *getData*.

4.3. Métodos de ordenamiento

4.4. Bubblesort

La llamada al método

```
1    void bubbleasort(char ** words, int size)
```

Genera un ordenamiento básico de bubblesort de forma ascendente.

4.5. Heapsort

La llamada al método

```
1    void heapsort(char ** words, int size)
```

Genera un ordenamiento de heapesort de forma ascendente.

5. Mediciones

Para tomar el tiempo de ejecución del programa, se utilizó el comando time*TIME*[3] del sistema. El comando time (del inglés time , tiempo) es un comando del sistema operativo Unix y derivados. Se utiliza para determinar la duración de ejecución de un determinado comando o proceso. Para utilizar el comando, simplemente hay anteponer la palabra time a lo que se quiere ejecutar. Al ejecutar el comando, time informará de cuanto tiempo llevó ejecutar el proceso en términos de tiempo de CPU de usuario, tiempo de CPU del sistema y tiempo real. El tiempo que hemos tomado es el de usuario ya que contempla solo el tiempo usado por el programa. En caso del tiempo de sistema, contempla los llamados al sistema. Para el caso del tiempo real, se refiere al tiempo transcurrido como si fuese un cronómetro.

Los siguientes gráficos muestran el tiempo de ejecución con los distintos algoritmos y en base al tamaño del archivo a ordenar.

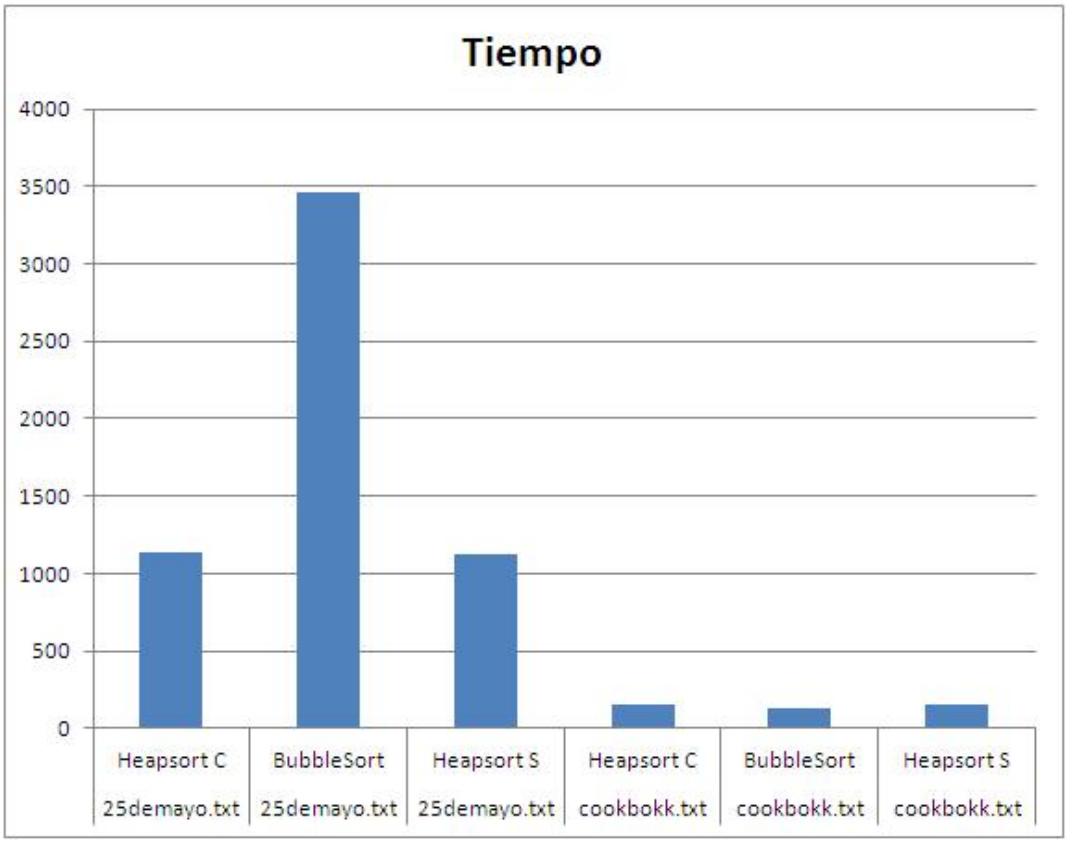


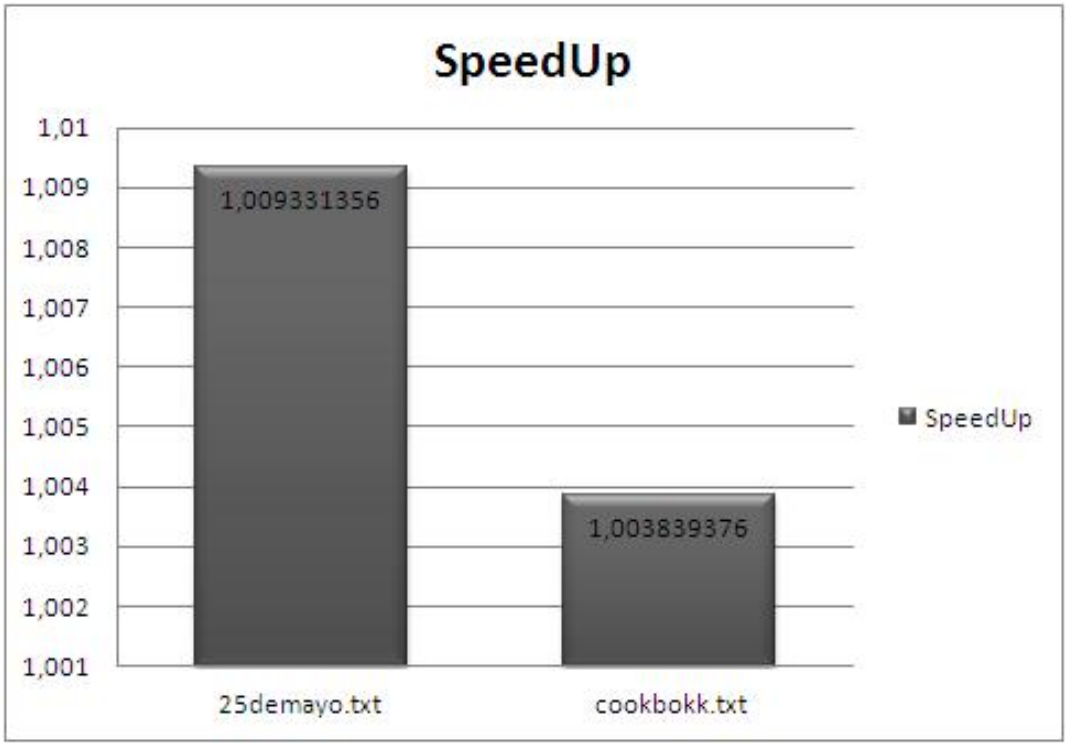
Figura 1: Tiempo de ejecución de los algoritmos.

5.1. Análisis de los resultados

A continuación se expone una tabla con los valores obtenidos de las mediciones medidas en segundos:

Archivo a ordenar	Heapsort C	Heapsort assembler	Bublesort
25demayo.txt	1136,82	1126,31	3456,84
cookbokk.txt	159,49	158,88	131,12

Para calcular el speedup, se utiliza la fórmula de tiempo viejo sobre tiempo nuevo.
En base a esta fórmula, se obtuvieron los siguientes resultados:



6. Código

El código se encuentra adjunto en el disco, tanto su versión en C como en MIPS.

7. Conclusión

El speedup reflejan cual de todos los algoritmos es mejor. En este caso, se ve reflejado como el Heapsort supera en desempeño al BubbleSort.
Para el caso de la versión de assembler diera mejores resultados que la versión realizada en código C, por tratarse de un diseño a más bajo nivel. Como pudo apreciarse en los resultados, los tiempos no fueron los esperados dando tiempos muy cercanos entre ambas versiones. Con lo que podríamos concluir, que no es conveniente invertir tanto tiempo en desarrollar código a bajo nivel. El código en lenguajes a alto nivel es mucho más fácil de desarrollar, mantener y detectar posibles fallos.

Referencias

[1] Bubble sort. http://en.wikipedia.org/wiki/Bubble_sort

[2] Heap sort http://es.wikipedia.org/wiki/Heapsort

[3] time man page, http://www.linuxmanpages.com/man1/time.1.php