

TP0 Organización de las Computadoras (66.20)

Grupo Nro. X - 1er. Cuatrimestre de 2012
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires



Rodriguez Genaro, Leandro

Padrón Nro. 92.098

leandrorodriguezg@yahoo.com.ar

Reale, Tomás

Padrón Nro. 92.255

tomasreale@gmail.com

Piechotka, Federico

Padrón Nro. 92.126

f_piecho@hotmail.com

Índice

1. Introducción	3
2. <i>Previo a realizacion de medidas</i>	3
2.1. Merge Sort	3
2.2. Selection Sort	3
3. Desarrollo	4
3.1. Mediciones de tiempo	4
3.2. Analisis de las mediciones de tiempo	6
4. Profiling con Gprof	8
4.1. Mediciones de tiempo	8

Resumen

La idea principal de este trabajo practico es aprender a utilizar ciertas herramientas fundamentales para el análisis de software. Con tal fin, se implementa un programa simple cuya función es ordenar cadenas de caracteres. Este programa cuenta con 2 algoritmos de ordenamiento (Merge Sort o Selección Sort) y cuenta con la posibilidad de escoger 1 de estos enviándole un parámetro (-m o -s) a la aplicación.

Las herramientas de análisis de software a utilizar son: Gxemul (para simular una maquina MIPS usando un SO NetBSD, y chequear portabilidad). gprof (herramienta de profiling para ver tiempos de nuestro programa). También se utilizara "time" de la consola de Linux.

1. Introducción

Con ayuda de los links provistos por la cátedra (wikipedia) se programo en el lenguaje C la aplicación antes mencionada, responsable de ordenar cadenas de caracteres, ya sea leyendo los datos de 1 o varios archivos o por consola(lo que seria entrada standar). Todo esto bajo el SO Ubuntu 10.04.

Además de los algoritmos de ordenamiento, el programa cuenta con funciones para leer y manejar archivos, y un parser para interpretar los argumentos que se le pasan al ejecutarlo por consola. El parser de argumentos se implemento usando la útil librería getopt, y para manipular archivos, se siguió lo que dicta la ortodoxia de los manuales que se encuentran dando vuelta por la red. Se configuro el gxemul para correr NetBSD sobre una maquina mips, y se lo comunico con nuestro usuario de ubuntu a través de ssh.

Con el programa ya programado en C, se envió el código fuente a la maquina virtual del gxemul, se compilo por consola (cuidándose de generar el código .s), y se ejecuto allí para poder verificar que efectivamente el programa fuese portable.

2. *Previo a realizacion de medidas*

2.1. Merge Sort

El algoritmo de ordenamiento por mezcla (merge sort en inglés) es un algoritmo de ordenamiento externo estable basado en la técnica divide y vencerás. Es de complejidad $O(n * \log(n))$, ya sea para el mejor caso, el peor caso, o el caso promedio.

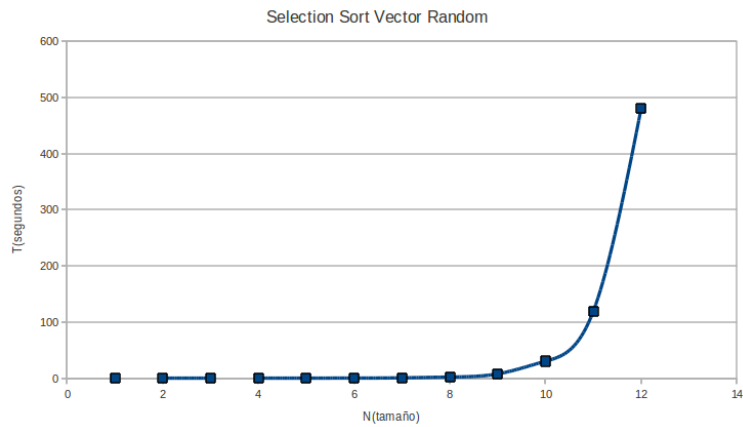
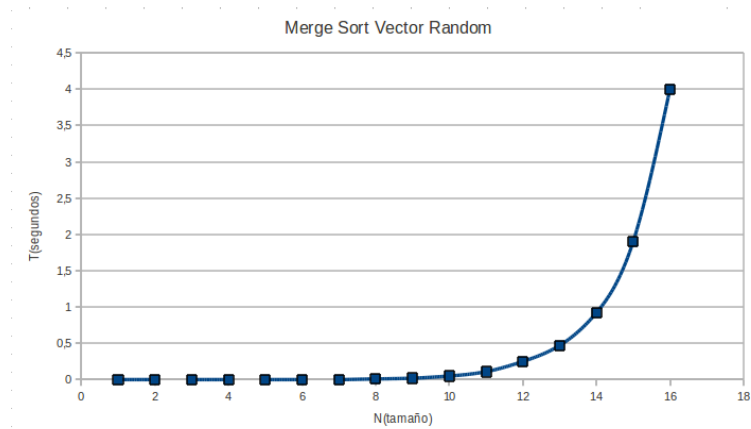
2.2. Selection Sort

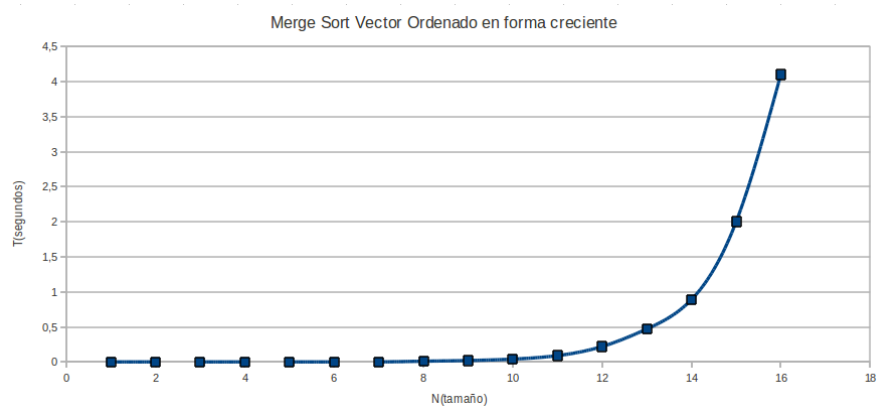
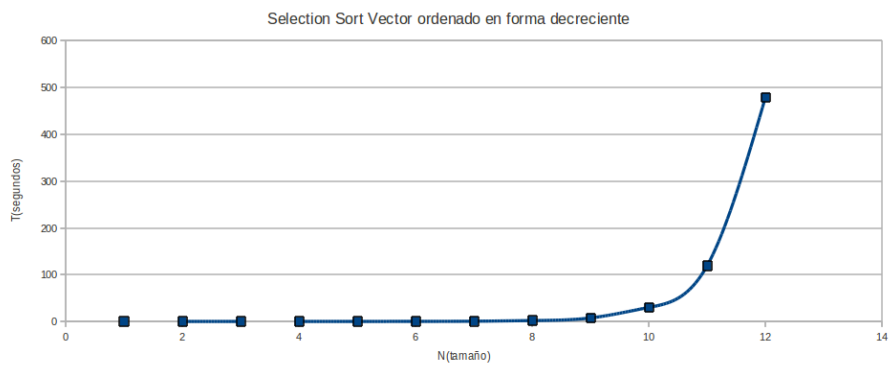
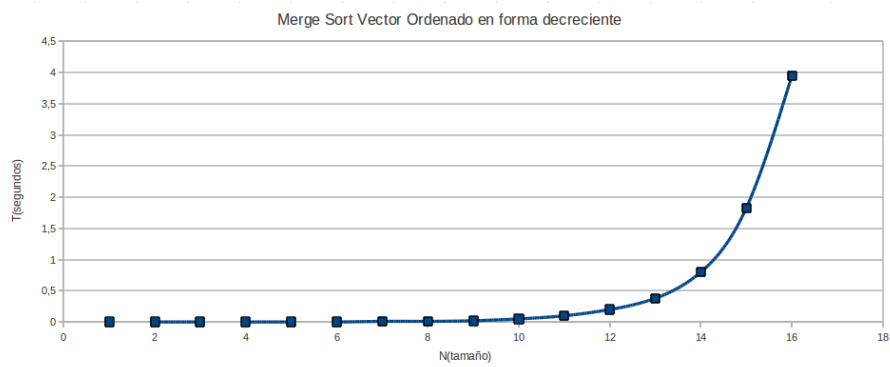
El ordenamiento por selección (Selection Sort en inglés) es un algoritmo de ordenamiento que requiere $O(n^2)$ operaciones para ordenar una lista de n elementos. Su complejidad no varia para el mejor, el peor, o el caso promedio.

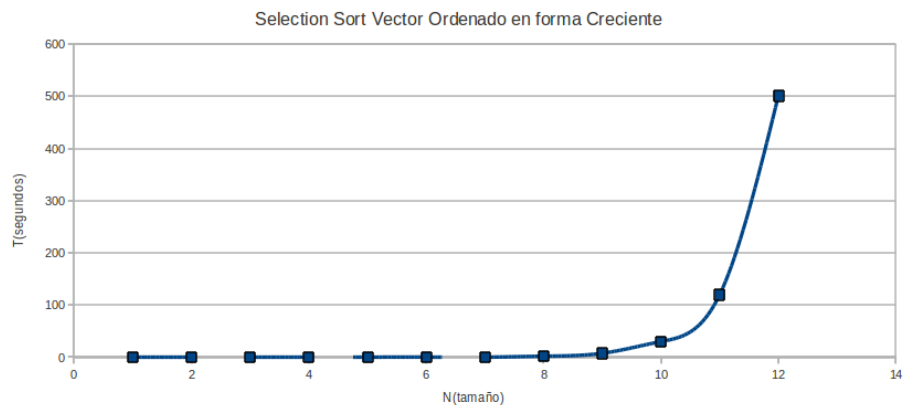
3. Desarrollo

3.1. Mediciones de tiempo

Siguiendo las instrucciones del tp, se crearon varios archivos de caracteres aleatorios, con un tamaño dado por $Size = 100 * 2^N$, $N = 0, 1, 2, 3, \dots, 16$. Antes de empezar a medir, se hicieron 3 copias para cada tamaño. Una se dejó tal cual fue creada. Otra se ordenó de menor a mayor. Otra se ordenó de mayor a menor. Con esas copias, se procedió a construir tablas del tiempo medido en función del tamaño, para cada uno de los algoritmos de ordenamiento. A continuación mostramos los gráficos obtenidos. Los datos se consiguieron ejecutando el programa para todos los distintos archivos, con los 2 métodos, usando la función time de la consola de Ubuntu para registrar los tiempos.







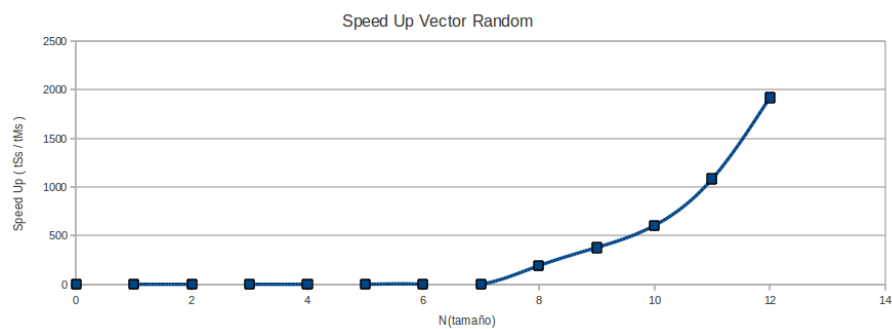
3.2. Analisis de las mediciones de tiempo

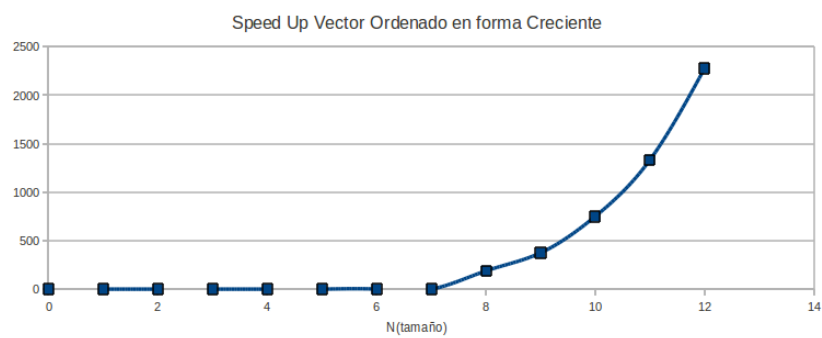
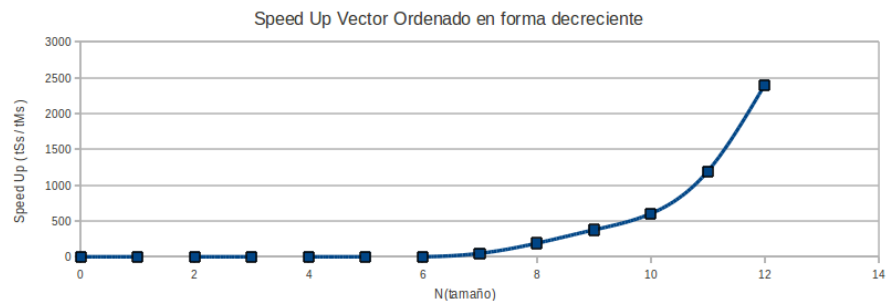
Es fácil ver que se verifica lo que se esperaba : que los 2 métodos de ordenamiento mantienen su orden de complejidad independientemente de como se encuentra el vector. Es fácil observar también que el Merge Sort es muchísimo mas rápido que el Selection Sort.

Cuando N tiende a números grandes, fue imposible medir el tiempo para terminar de ordenar un vector con Selection Sort, se detuvo la medición al pasar mas de 10 minutos. Para N chicos, los 2 métodos son similares. Usando estos valores, se consiguieron los Speed Up de Merge Sort vs Selection Sort.

$$SpeedUpMs = \frac{TiempoMergeSort}{TiempoSelectionSort}$$

Vale mencionar que para N chicos, donde los tiempos de ambos métodos son 0 o casi 0, se considera el Speed Up nulo, indicando que los 2 métodos tardaron lo mismo. A continuacion, los graficos de los Speed Up para los distintos N, segun la forma en la que se encuentra el vector a ordenar.





Para los casos donde $N > 12$, el tiempo del Merge Sort es mas de 2 ordenes de magnitud menor que el de Selection Sort, el Speed Up crece vertiginosamente. Se ve que los speed up varian segun los distintos casos. Esto no deberia suceder, y se debe hay otras aplicaciones corriendo mientras se mide la performance de los algoritmos de ordenamiento, lo cual introduce algo de error en las mediciones.

4. Profiling con Gprof

4.1. Mediciones de tiempo

Con lo dicho en los puntos anteriores, vemos que:

1. Merge Sort es mucho mas rapido que Selection Sort
2. Para casos muy grandes, no se pueden estimar los tiempos del Selection Sort, ya que da valores cercanos a las horas
3. Para casos muy chicos, los algoritmos se comportan de forma similar

Debido a esto, elegimos un caso medio, donde $N = 10$. Vamos a usar un vector random y la herramienta Gprof. Vale aclarar que para poder usar esta herramienta, tuvimos que haber compilado nuestro programa con el flag -pg . Mas adelante se vera en mas profundidad los comandos usados para compilar el codigo source en C .

El gprof nos entrega un texto. Aqui se reproducen solo las partes importantes.

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
100.27	30.00	30.00	1	30.00	30.00	SelectionSort
0.00	30.00	0.00	1	0.00	0.00	ExportarCadena
0.00	30.00	0.00	1	0.00	0.00	LeerArchivoDeCaracteres
0.00	30.00	0.00	1	0.00	0.00	SubLeer
0.00	30.00	0.00	1	0.00	0.00	SumarCadenas

index	% time	self	children	called	name
		30.00	0.00	1/1	main [2]
[1]	100.0	30.00	0.00	1	SelectionSort [1]

					<spontaneous>
[2]	100.0	0.00	30.00		main [2]
		30.00	0.00	1/1	SelectionSort [1]
		0.00	0.00	1/1	LeerArchivoDeCaracteres [4]
		0.00	0.00	1/1	SumarCadenas [6]
		0.00	0.00	1/1	ExportarCadena [3]

		0.00	0.00	1/1	main [2]
[3]	0.0	0.00	0.00	1	ExportarCadena [3]

		0.00	0.00	1/1	main [2]
[4]	0.0	0.00	0.00	1	LeerArchivoDeCaracteres [4]
		0.00	0.00	1/1	SubLeer [5]

		0.00	0.00	1/1	LeerArchivoDeCaracteres [4]
[5]	0.0	0.00	0.00	1	SubLeer [5]


```

-----
          0.00    0.00    1/1          main [2]
[6]      0.0    0.00    0.00    1          SumarCadenas [6]
-----

```

Index by function name

```

[3] ExportarCadena          [1] SelectionSort          [6] SumarCadenas
[4] LeerArchivoDeCaracteres [5] SubLeer

```

Se puede ver que practicamente el 100 por ciento del tiempo que se ejecuta el programa, este se encuentra ejecutando la funcion Selection Sort

Referencias

- [1] Tobias Oetiker, *The Not So Short Introduction to L^AT_EX 2_ε*. USA, Version 5.01, 2011.
- [2] Merge-sort, http://en.wikipedia.org/wiki/Merge_sort
- [3] Selection-sort, http://en.wikipedia.org/wiki/Selection_sort
- [4] GXemul, <http://gavare.se/gxemul/>
- [5] The NetBSD project, <http://www.netbsd.org/>
- [6] time man page, <http://unixhelp.ed.ac.uk/CGI/man-cgi?time>
- [7] GNU gprof, <http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html>