Reporte de resultados: Tarea Programada 1

Jeremy Rojas, B96804 <jeremy.rojascerna@ucr.ac.cr>

***Resumen*—En este trabajo se mide una serie de algoritmos con tal de comparar sus tiempos de ejecución entre ellos, y con sus cotas asintóticas. Para esto, cada algoritmo se implementó en C++ y se midió la duración de cada uno. Los resultados fueron tabulados y graficados para facilitar el análisis. Se concluye que las cotas asintóticas para los tiempos promedio se ven respaldados en la práctica.**

***Abstract*—In this paper, a series of algorithms were measured with the goal of comparing their runtimes between each other, and with their asymptotic bounds. For this, each algorithm was implemented in C++, and their runtimes were measured. The results were logged in a table, and then mapped in a graph for ease of análisis. It is concluded that the asymptotic bounds for the average runtimes are backed by data.**

Palabras clave—ordenamiento, selección, inserción, mezcla, montículos, rápido, residuos.

Keywords—sorting, selection, insertion, merge, heap, quicksort, radix.

I. Introducción

En este trabajo se mide y se compara los tiempos de ejecución de seis algoritmos vistos durante el curso CI-0116 Análisis de Algoritmos y Estructuras de Datos. Estos algoritmos son los siguientes:

- Ordenamiento por Selección (Selection Sort).
- Ordenamiento por Inserción (Insertion Sort).
- Ordenamiento por Mezcla (Merge Sort).
- Ordenamiento por Montículos (Heap Sort).
- Ordenamiento Rápido (Quicksort).
- Ordenamiento por Residuos (Radix Sort).

Con esto se busca comparar los tiempos de ejecución entre los algoritmos entre sí, y además comparar sus órdenes de crecimiento, según los datos obtenidos, con sus cotas asintóticas.

II. Metodología

Para lograr lo propuesto se implementó cada algoritmo en C++. El programa de prueba genera arreglos de enteros positivos de cuatro tamaños (mil, diez mil, cien mil, un millón). Cada arreglo es ordenado tres veces con cada algoritmo. Finalmente, el programa reporta el tiempo que tardó en ordenar el arreglo cada vez. Estos tiempos luego fueron anotados y graficados con tal de facilitar el análisis.

La generación de números aleatorios se realizó con el objeto *random_device* de la biblioteca estándar <random>. La medición del tiempo se realizó con el objeto *high_resolution_clock* de la biblioteca estándar <chrono>.

III. Resultados

Los tiempos de ejecución de las tres corridas de los algoritmos se muestran en el cuadro I.

Cuadro I Tiempo de ejecución de los algoritmos

		Tiempo (ms) *Corrida*			
Algoritmo	Tam. (k)	1	2	3	Prom.
Selección	1000	4.661097	1.573360	1.420824	2.5517603
	10 000	106.769312	103.453994	102.729384	104.317563
	100 000	10120.495933	10127.217507	10115.526653	10121.080031
	1 000 000	1041505.47	1162922.29	1120991.77	1108473.18
Inserción	1000	0.627467	0.617077	0.614201	0.6195817
	10 000	59.172352	59.967198	60.060248	59.733266
	100 000	6136.926509	6239.254123	6295.757521	6223.9793843
	1 000 000	644773.02	651448.56	669649	655290.1946
Mezcla	1000	0.125647	0.119973	0.118519	0.1213797
	10 000	1.536305	1.973854	1.514471	1.674877
	100 000	18.37174	17.930183	18.376615	18.2261793
	1 000 000	215.062076	214.229425	214.785866	214.6924557
Montículo	1000	0.192216	0.185172	0.19402	0.1904693
	10 000	2.618141	2.642735	2.602497	2.6211243
	100 000	34.42126	34.972582	35.244421	34.879421
	1 000 000	487.933053	491.169889	492.068778	490.390573
Rápido	1000	0.104821	0.102261	0.099937	0.1023397
	10 000	1.334245	1.403377	1.406245	1.381289
	100 000	16.494821	16.370678	16.45425	16.4399163
	1 000 000	205.786624	207.533294	208.773739	207.3645523
Residuos	1000	0.043416	0.034559	0.034402	0.037459
	10 000	0.283557	0.267296	0.262953	0.2712687
	100 000	3.298804	4.350832	3.104175	3.5846037
	1 000 000	26.695715	26.242958	27.270793	26.7364887

Los tiempos promedio se muestran gráficamente en las figuras 1 a 6. La figura 7 muestra los tiempos promedio de todos los algoritmos en una escala logarítmica de manera conjunta.

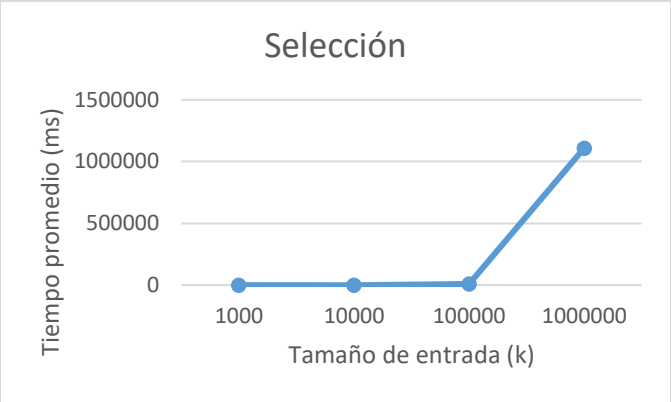

Figura 1 Tiempos promedio de ejecución de ordenamiento por selección

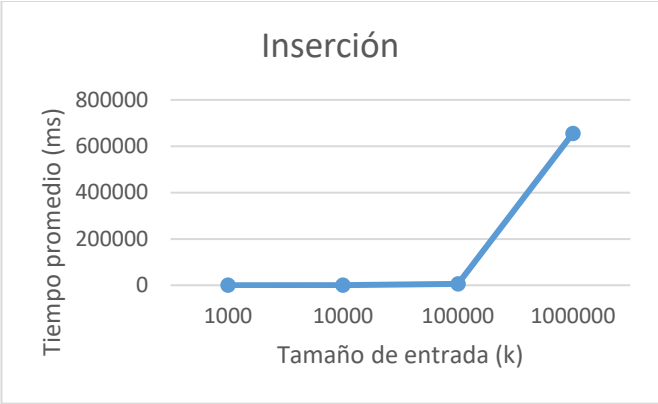

Figura 2 Tiempos promedio de ejecución de ordenamiento por inserción

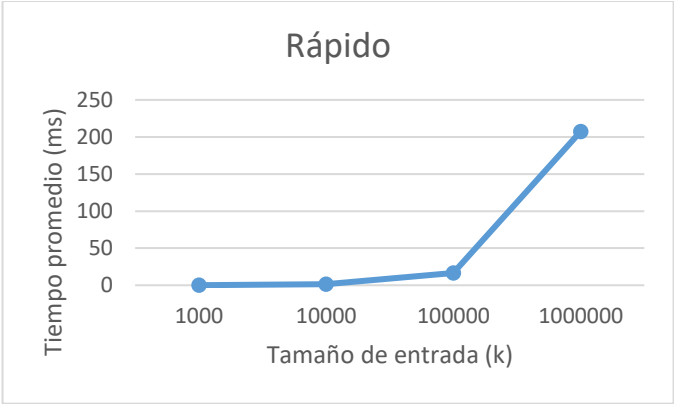

Figura 5 Tiempos promedio de ejecución de ordenamiento rápido

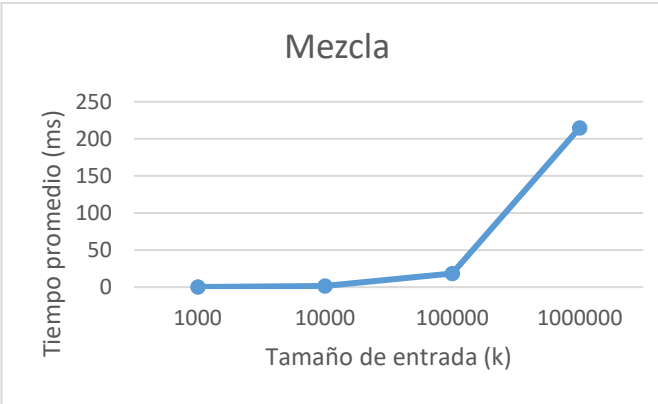

Figura 3 Tiempos promedio de ejecución de ordenamiento por mezcla

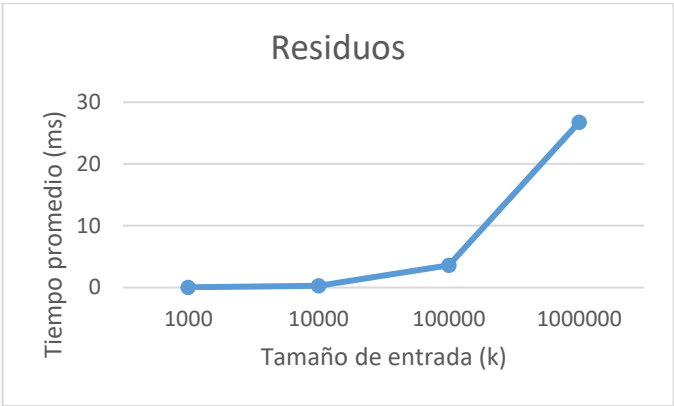

Figura 6 Tiempos promedio de ejecución de ordenamiento por residuos

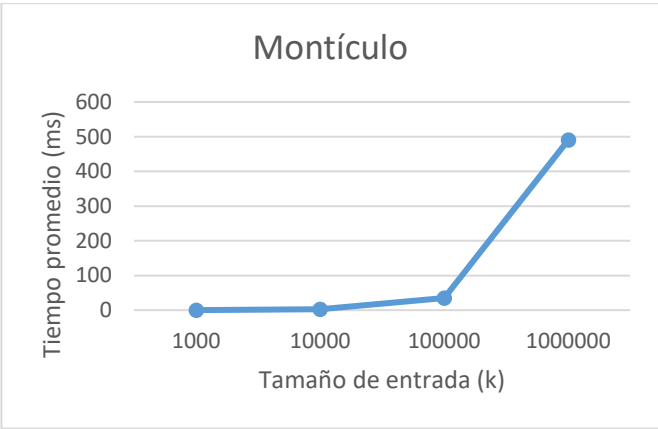

Figura 4 Tiempos promedio de ejecución de ordenamiento por montículo

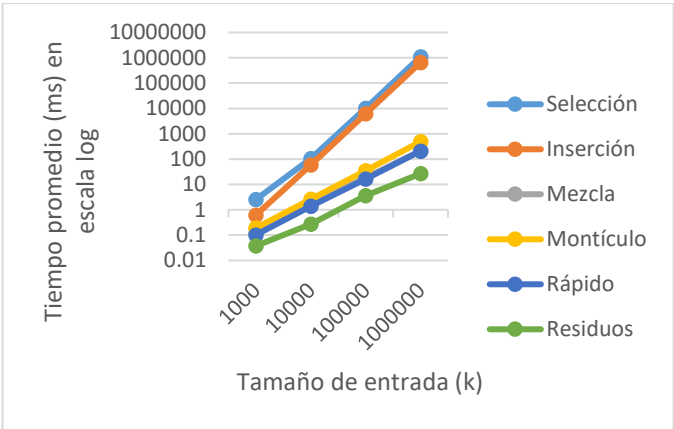

Figura 7 Gráfica comparativa de los seis tiempos promedio

IV. Discusión

Como primer punto, con base en los resultados presentados, se puede observar que el algoritmo más lento, de lejos, es el de selección, seguido del de inserción. En un tercer lugar les sigue ordenamiento por montículo, y en cuarto lugar hay un empate entre ordenamiento por mezcla y ordenamiento rápido. Finalmente, el algoritmo más rápido es el ordenamiento por residuos.

Más allá de eso, interesa observar que las gráficas individuales de cada algoritmo no son muy distintas en forma. Si bien los saltos en el eje Y cambian —por las diferencias de tiempo que presentan—, todas describen, en mayor o menor medida, una gráfica logarítmica. Esto se puede achacar a los tamaños de entrada elegidos; con tamaños de entrada más cercanos entre sí se podría describir mejor la curva, de manera que cada gráfica se acercaría más a las funciones asintóticas correspondientes.

Aún con lo similares que son las gráficas en forma, los datos permiten realizar un análisis más detallado —y más importante aún, reflejan lo esperado dado las cotas asintóticas.

Primero, los dos algoritmos más lentos: inserción y selección. Lo importante, tanto de las gráficas como de la tabla, es que estos crecen más rápido que los demás: a mayor tamaño de entrada, mayor duración. Esto se ve particularmente bien en la figura 7. Cuando se observa su cota, esto tiene sentido: tienen, de entre todos los algoritmos aquí estudiados, la cota más lenta, en $\Theta(n^2)$.

Ahora, al analizar los siguientes tres (ordenamiento por montículo, ordenamiento rápido, y ordenamiento por mezcla), el que estén de segundos y en una dimensión relativamente cercana (la mayor diferencia entre ellos es de poco menos de trescientos cuando la entrada es del millón, mientras que inserción y selección se separan de ellos, como mínimo, en más de seiscientos mil con esa misma entrada) tiene sentido: sus cotas en los casos promedios es de $\Theta(n \lg n)$. Como bien se sabe, $n \lg n$ es menor que n^2, de manera que el crecimiento de estos algoritmos debe de ser menor al de los dos primeros, por bastante. Y, efectivamente, lo es.

Finalmente, el ordenamiento por residuos es el más rápido y, de nuevo, esto calza con su cota promedio: $\Theta(d(n + k))$. Si se ignora d y k porque se toman como constantes del algoritmo, y se analiza su tiempo asintótico exclusivamente en términos de su entrada, pues es el valor más variable, el ordenamiento por residuos es, esencialmente, lineal. Esto significa que su crecimiento debería ser menor a cualquiera de los dos descritos por las cotas de los algoritmos anteriores (logarítmica y cuadrática); y, una vez más, lo es.

V. Conclusiones

A partir de los resultados se puede concluir que la teoría se ve respaldada por los datos: aún con tan pocas pruebas, y con unas entradas relativamente pequeñas (pues aún los algoritmos más lentos no tardaron más de veinte minutos en una máquina personal con un procesador y una memoria no muy espectaculares), cada algoritmo crece en el orden que se esperaría según sus respectivas cotas dadas para los casos promedio.