Escuela de Ingeniería
Departamento de Computación
Análisis y Diseño de Algoritmos (TC2017)
Profesor: Dr. Vicente Cubells Nonell

Tecnológico de Monterrey

Tarea No. 2

Título	**Análisis de algoritmos de ordenamiento**
Aprendizaje esperado (objetivo)	El alumno demostrará su capacidad para programar diferentes algoritmos de ordenamiento y medir el tiempo de ejecución de los mismos en dos arquitecturas de hardware diferentes, así como analizar e interpretar los resultados obtenidos, comparando los diferentes algoritmos.
Instrucciones	Programa los algoritmos de ordenamiento indicados utilizando C++ (Xcode u otra IDE). Genere un arreglo de números enteros de manera aleatoria y calcule el tiempo de ejecución de cada algoritmo. Debe hacer las mediciones en una BeagleBone o una Raspberry Pi y en su laptop o computadora de escritorio. Calcule los tiempos para N = 10, 100, 1 000, 10 000, 100 000, 1 000 000 ó el número máximo posible (según su arquitectura), y rellene la tabla que aparece en la orientaciones de la actividad. Para que la comparación sea correcta, cada vez que genere un arreglo de números, debe ejecutar cada algoritmo tomando como entrada el mismo arreglo desordenado. Para cada población (10, 100, etc.), genere una gráfica (en Google Sheets) comparativa de todos los algoritmos y sus tiempos de ejecución en la BB / RPi y en la laptop / escritorio. Analice e interprete los resultados alcanzados. Confeccione un documento en Google Docs donde incluya la tabla de resultados, las gráficas comparativas y su interpretación personal de los resultados obtenidos. Suba a la plataforma el archivo con los resultados e incluya en el mismo la liga al repositorio de GitHub que contenga todos los códigos de los algoritmos programados. No se aceptan trabajos fuera de fecha ni por correo electrónico. En ambos casos la calificación de la tarea será 0 puntos.
Lugar en que se llevará a cabo	Casa
Forma de trabajo	Individual

Recursos	Foros de información en Internet Wikipedia (http://www.wikipedia.org) Códigos de algoritmos vistos en la materia Estructura de Datos Computadora
Tiempo estimado	5 horas
Criterios de evaluación	El código de cada algoritmo (12 en total) tendrá un valor de 3 puntos. Medir el tiempo de ejecución de cada algoritmo (12) en cada plataforma (2), para cada población (6) tendrá un valor de 0.25 puntos. Generar cada gráfica (6) que incluya los tiempos de ejecución de todos los algoritmos (12) en cada plataforma (2) tendrá un valor de 1 puntos. Analizar e interpretar los resultados correctamente tendrá un valor de 16 puntos.
Valor de la actividad	10% de la calificación del primer parcial

Algoritmos de ordenamiento a considerar

1. Ordenamiento de burbuja (*Bubble Sor*t)
2. Ordenamiento de burbuja bidireccional (*Cocktail Sort*)
3. Ordenamiento por inserción (*Insertion Sort*)
4. Ordenamiento por casilleros (*Bucket Sort*)
5. Ordenamiento por cuentas (*Counting Sort*)
6. Ordenamiento por mezcla (*Merge Sort*)
7. Ordenamiento con árbol binario(*Binary tree Sort*)
8. Ordenamiento Radix (*Radix Sort*)
9. Ordenamiento Shell (*Shell Sort*)
10. Ordenamiento por selección (*Selection Sort*)
11. Ordenamiento por montículos (*Heap Sort*)
12. Ordenamiento rápido (*Quick Sort*)

Respuestas

Repositorio de GitHub:

https://github.com/tec-csf/TC2017-T2-Otono-2019-dvigleo

Tabla con los resultados de las mediciones:

Tabla de resultados a completar (*tiempo en ms*)								
No.	**Algoritmo**		**10**	**100**	**1000**	**10 000**	**100 000**	**200 000**
1	**Bubble Sort**	*RPi*	0	0	51	1906	229076	774907
		PC	0	0	2	263	30333	193758
2	**Cocktail Sort**	*RPi*	0	0	42	1287	182688	633196
		PC	0	0	1	437	66391	231
3	**Insertion Sort**	*RPi*	0	0	18	835	77378	316601
		PC	0	0	0	112	7629	190361
4	**Bucket Sort**	*RPi*	0	0	1	7	82	112
		PC	0	0	0	0	5	47
5	**Counting Sort**	*RPi*	0	0	0	8	37	136
		PC	0	0	0	1	6	41
6	**Merge Sort**	*RPi*	0	0	1	18	81	237
		PC	0	0	1	1	13	39
7	**Binary tree Sort**	*RPi*	0	0	7	6	303	96
		PC	0	0	1	5	54	45
8	**Radix Sort**	*RPi*	0	0	1	11	47	175
		PC	0	0	0	0	5	40
9	**Shell Sort**	*RPi*	0	0	0	18	52	259
		PC	0	0	0	1	6	174
10	**Selection Sort**	*RPi*	0	0	23	1008	103571	420953
		PC	0	0	0	131	13993	103116
11	**Heap Sort**	*RPi*	0	0	2	34	170	477
		PC	0	0	0	2	30	94
12	**Quick Sort**	*RPi*	0	0	4	17	805	1523
		PC	0	0	1	7	124	439

Interpretación de los resultados:

Lamentablemente la arquitectura de mi computadora no soportaba un tamaño de arreglo mayor a 200,000 elementos, pues el código se detenía de pronto y dejaba de ejecutarse. Dicho esto, viendo las gráficas comparativas y analizando los resultados obtenidos en la tabla de arriba, cuando un arreglo es de tamaño 10 o 100, no tiene ningún impacto en el tiempo de ejecución de un algoritmo. No es hasta que el n crece a partir de 1000 que empezamos a ver cambios:

<u>Cuando n = 1 000 elementos</u>
En Raspberry Pi

- El tiempo de ejecución es mayor, superando los 40 milisegundos en los algoritmos de ordenamiento de Cocktail Sort y Bubble Sort. Aún existen dos algoritmos que tienen como tiempo de ejecución 0 ms.

En PC

- El tiempo de ejecución es mucho más rápido que en la Raspberry Pi. El algoritmo más lento siendo Bubble Sort (2 ms), seguido de Cocktail Sort (1 ms). Existen siete algoritmos que tienen como tiempo de ejecución 0 ms.

Estos resultados tienen sentido, pues se sabe que Bubble Sort es realmente ineficiente en el peor de los casos $O(n^2)$. Y dado que Cocktail Sort es una modificación de Bubble Sort, igual tiene sentido que no sea el algoritmo más eficiente $O(n^2)$.

<u>Cuando n = 10 000 elementos</u>
En Raspberry Pi

- El tiempo de ejecución para el algoritmo más lento (Cocktail Sort) está por encima de los 1000 ms. El más rápido (Bucket Sort) realizó el ordenamiento en 6 ms.

En PC

- El tiempo de ejecución es mucho más rápido que en la Raspberry Pi. El algoritmo más lento siendo Cocktail Sort (2 ms), seguido de Bubble Sort (1 ms). Existen siete algoritmos que tienen como tiempo de ejecución 0 ms.

Aquí los resultados de los algoritmos más lentos difieren mucho de la entrada anterior, con Cocktail Sort. Esto nos hace pensar que depende mucho el tiempo de ejecución de los elementos que componen el arreglo, pues el orden de los elementos pueden acelerar o alentar el ordenamiento de estos. Pareciera que en este caso, los elementos aleatorios que se generaron estuvieron en pro de Bubble Sort y en contra de Cocktail Sort.

<u>Cuando n = 100 000</u>
En Raspberry Pi

- El tiempo de ejecución para el algoritmo más lento (Bubble Sort) es considerablemente más lento: pasando de los 200 000 ms (aproximadamente 3.33 minutos). El algoritmo más rápido fue counting (37 ms).

En PC

- El algoritmo más lento vuelve a ser Bubble Sort, pero sigue siendo mucho más rápido ejecutarlo en la PC que en el RPi (60 000 ms = 1 minuto). El más rápido Bucket Sort (5 ms)

A pesar de que Bucket Sort trabaja mejor cuando están los datos están uniformemente distribuidos sobre un rango, en este caso, sorprendentemente, al ejecutarlo en la PC fue el algoritmo más rápido, lo que simplemente nos hace pensar que los datos de entrada estaban casualmente ordenados de manera que favoreciera a Bucket Sort, otorgando una complejidad de O(n).

Cuando n = 200 000

En Raspberry Pi

- Aquí una vez más, Bubble Sort (774907 ms ~ 13 minutos) es el algoritmo más lento y el más rápido fue Binary Tree Sort (96 ms).

En PC

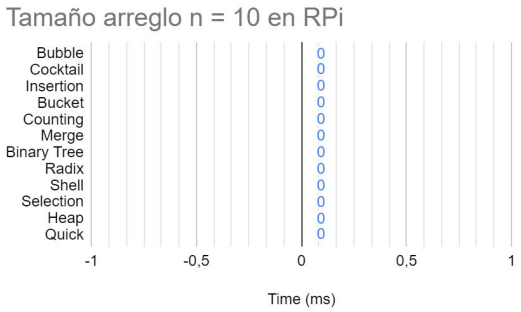
- Bubble Sort sigue siendo el algoritmo más lento (193758 ms ~ 3.22 minutos) y el más rápido con 39 ms (Merge Sort).

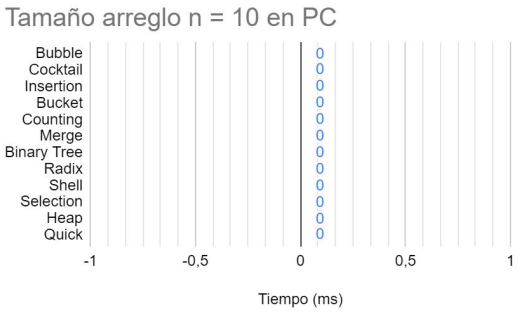
Es importante notar que otro de los algoritmos lentos en todas las rondas, fue Selection Sort con una complejidad de $O(n^2)$. Selection sort en, en el mejor de los casos, 60% más rápido que Bubble Sort, pero aún así destaca entre los peores algoritmos de ordenamiento. La complejidad de Binary Tree Sort y de Merge Sort, es O(nlogn); en ambos, en sus respectivos dispositivos de ejecución, fueron los más rápidos.

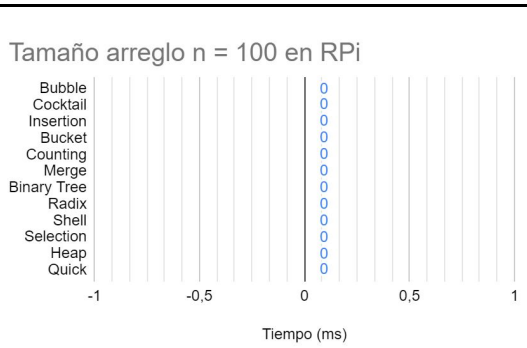
En general, Bubble Sort es el algoritmo de ordenamiento más lento de todos, seguido de Cocktail sort y después Selection Sort; los tres con una complejidad de $O(n^2)$. Por otro lado, entre los algoritmos más rápidos, se encuentran Binary Tree Sort, Merge Sort y Counting Sort, con complejidades más favorecientes que los otros tres.

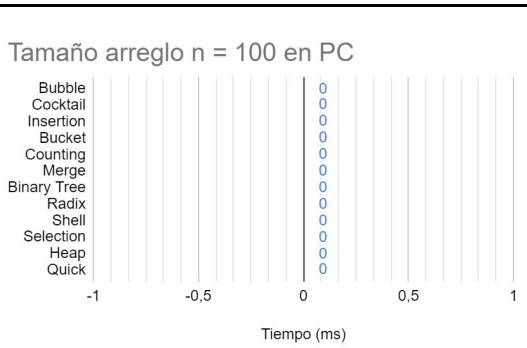
Cabe mencionar, que el tiempo de ejecución de un algoritmo, varía mucho por la entrada del arreglo que tiene; pues algún acomodo aleatorio de *n* números puede favorecer o no a un cierto algoritmo. Además, ejecutar un programa en una PC con especificaciones decentes, es considerablemente más rápido que ejecutarlos dentro de un RaspberryPi, esto es debido a la arquitectura y poca potencia con la que cuenta el RPi.

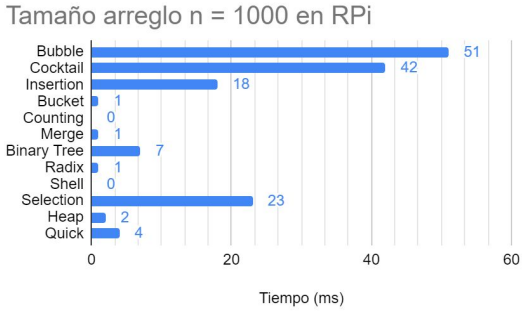
Gráficas comparativas

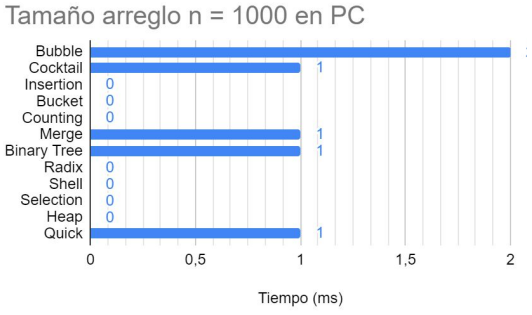


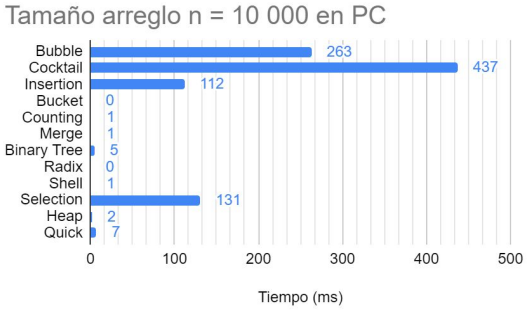


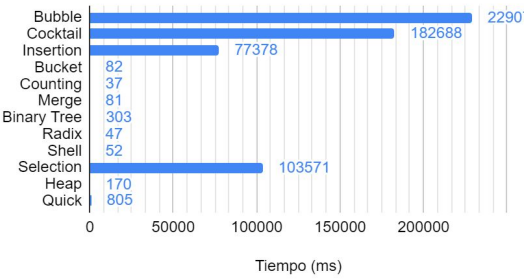


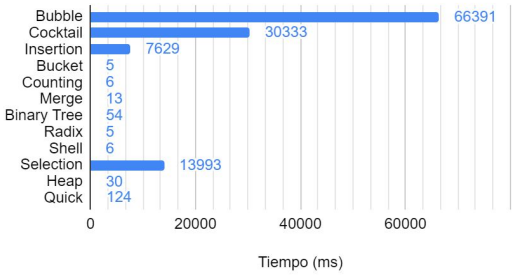




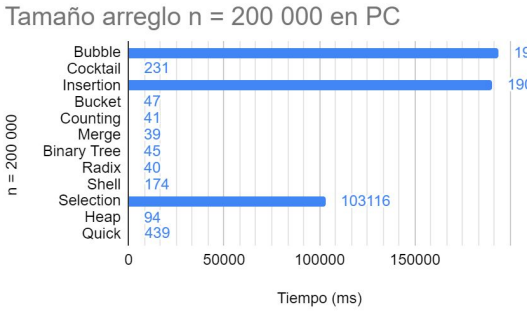





Tamaño arreglo n = 100 000 en RPi
Bubble 2290
Cocktail 182688
Insertion 77378
Bucket 82
Counting 37
Merge 81
Binary Tree 303
Radix 47
Shell 52
Selection 103571
Heap 170
Quick 805
0
50000
100000
150000
200000
Tiempo (ms)


Tamaño arreglo n = 100 000 en PC
Bubble 66391
Cocktail 30333
Insertion 7629
Bucket 5
Counting 6
Merge 13
Binary Tree 54
Radix 5
Shell 6
Selection 13993
Heap 30
Quick 124
0
20000
40000
60000
Tiempo (ms)


Tamaño arreglo n = 200 000 en RPi
Bubble 77
Cocktail 633196
Insertion 316601
Bucket 112
Counting 136
Merge 237
Binary Tree 96
Radix 175
Shell 259
Selection 420953
Heap 477
Quick 1523
0
200000
400000
600000
Tiempo (ms)


Tamaño arreglo n = 200 000 en PC
n = 200 000
Bubble 19
Cocktail 231
Insertion 190
Bucket 47
Counting 41
Merge 39
Binary Tree 45
Radix 40
Shell 174
Selection 103116
Heap 94
Quick 439
0
50000
100000
150000
Tiempo (ms)