**División de Ingeniería**

**Departamento de Computación**

**Análisis y Diseño de Algoritmos (TC2017)**

**Profesor: Dr. Vicente Cubells Nonell**

**Jorge A. Barrios Mendoza**

**A01020781**

### 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/JorgeBarMza/Sorting-Algorithms*](https://github.com/JorgeBarMza/Sorting-Algorithms)

**Tabla con los resultados de las mediciones:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tabla de resultados a completar (***tiempo en \*micro\* segundos***)** | | | | | | | | |
| **No.** | **Algoritmo** |  | **10** | **100** | **1000** | **10 000** | **100 000** | **1 000 000 ó máx** |
| **1** | **Bubble Sort** | *RPi* | 7 | 581 | 60370 | 3305133 | 336574473 | 41014223839 |
| *PC* | 1 | 32 | 3494 | 970318 | 44770593 | 4534561979 |
| **2** | **Cocktail Sort** | *RPi* | 10 | 529 | 57034 | 4397507 | 411277798 | 3848219835 |
| *PC* | 1 | 30 | 3328 | 1185493 | 56859381 | 367823564 |
| **3** | **Insertion Sort** | *RPi* | 6 | 434 | 54359 | 5832358 | 244184442 | 25932889724 |
| *PC* | 0 | 22 | 2294 | 917747 | 34515321 | 2268291545 |
| **4** | **Bucket Sort** | *RPi* | 126 | 715 | 9674 | 110094 | 1105977 | 12216392 |
| *PC* | 10 | 53 | 4885 | 59614 | 221078 | 2913316 |
| **5** | **Counting Sort** | *RPi* | 420 | 890 | 963 | 2213 | 2236 | 15237 |
| *PC* | 44 | 50 | 54 | 97 | 631 | 7065 |
| **6** | **Merge Sort** | *RPi* | 59 | 1025 | 10658 | 129053 | 1376866 | 11460854 |
| *PC* | 12 | 198 | 842 | 81917 | 227770 | 2227117 |
| **7** | **Binary tree Sort** | *RPi* | 15 | 106 | 1356 | 17194 | 162706 | 11334121 |
| *PC* | 2 | 10 | 117 | 5748 | 154198 | 11147518 |
| **8** | **Radix Sort** | *RPi* | 27 | 119 | 1074 | 10472 | 106989 | 80304 |
| *PC* | 2 | 6 | 50 | 485 | 5158 | 71918 |
| **9** | **Shell Sort** | *RPi* | 4 | 135 | 10989 | 978411 | 82741424 | 6168766319 |
| *PC* | 1 | 9 | 656 | 134636 | 7953811 | 605114111 |
| **10** | **Selection Sort** | *RPi* | 5 | 235 | 20976 | 1347914 | 161205888 | 11836518639 |
| *PC* | 1 | 13 | 1295 | 366319 | 22242204 | 983519898 |
| **11** | **Heap Sort** | *RPi* | 1 | 155 | 2430 | 33750 | 450217 | 5817354 |
| *PC* | 1 | 13 | 179 | 10398 | 47805 | 1246063 |
| **12** | **Quick Sort** | *RPi* | 6 | 58 | 820 | 34099 | 55992 | 3369578 |
| *PC* | 1 | 7 | 104 | 2597 | 119926 | 819037 |

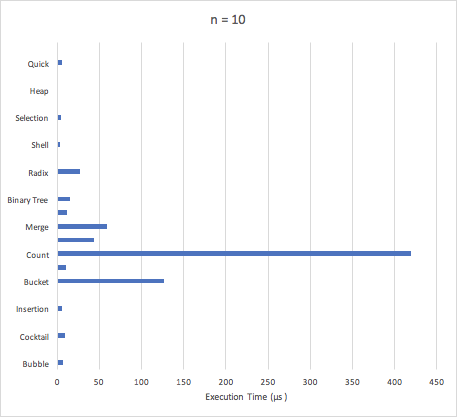
**Interpretación de los resultados:**

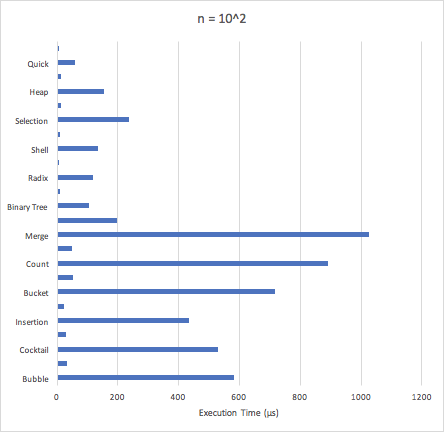
Las variaciones en el tiempo de ejecución de los 12 algoritmos de ordenamiento demuestra que la eficiencia de cada uno es drásticamente dependiente del tamaño “n” de la cantidad de datos a ordenar. De acuerdo a su tiempo de ejecución promedio, los algoritmos analizados se pueden clasificar en 3 tipos: comparaciones en tiempo cuadrático (Bubble, Cocktail, Selection, Shell, Insertion), comparaciones en tiempo cuasi-lineal (Merge, Heap, Binary Tree, Quick) y por llaves (Bucket, Radix, Count). Algunos algoritmos que dependen de un rango (i.e. Count, Shell) tienen desempeño temporal afectado por esta variable adicional.

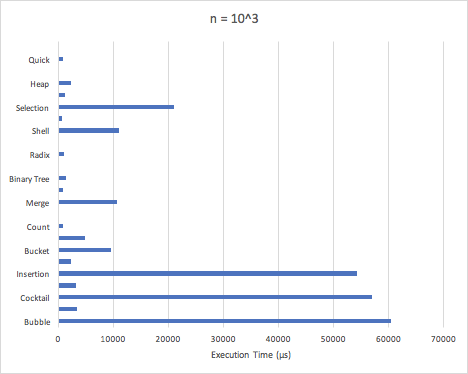
En las primeras dos gráficas, se muestra que los algoritmos por comparaciones cuadráticas son más eficientes. Esto ocurre porque los otros tipos de algoritmos invierten tiempo al inicio para estructurar los datos. Sin embargo, cuando “n” es mayor a 100, las graficas se invierten, de manera que los algoritmos logarítmicos se ejecutan en un tiempo significantemente menor al de complejidad cuadrática. Este comportamiento continua eternamente mientras “n” crece. Esto ocurre porque O(nlogn) esta acotado por O(n^2). Por lo tanto, si se utiliza una gran cantidad de datos, es óptimo usar algoritmos cuasi-lineales.

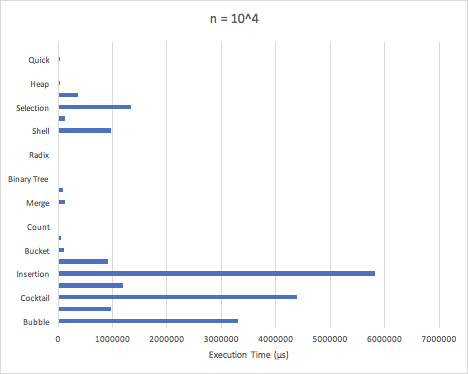
Sorprendentemente, el poder de procesamiento demostro ser de poca influencia, a comparación del algoritmo usado. Esto se observa al notar en todas las gráficas que la tendencia de los algoritmos es la misma en la ejecución por una potente Macbook Pro contra un simple Raspberry Pi 3. Por esto es esencial elegir los algoritmos más apropiados.

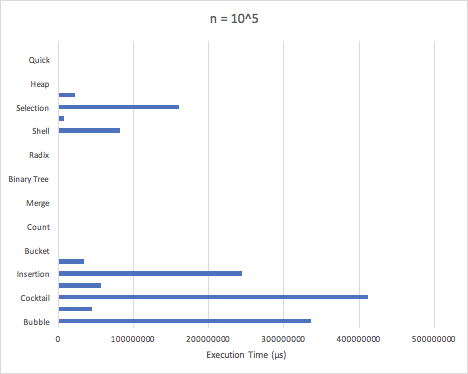
**Gráficas comparativas**

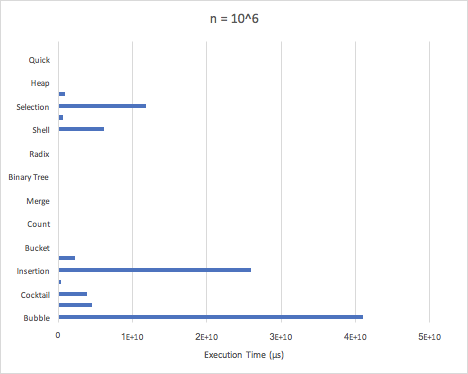
****

****

****

****

****

****