**Escuela de Ingeniería**

**Departamento de Computación**

**Estructuras de datos y algoritmos fundamentales (TC1031)**

**Profesor: Dr. Leonardo Chang**

### Actividad práctica. Algoritmos de ordenamiento

**ATENCIÓN: Subir este document con las respuestas en formato PDF!**

|  |  |
| --- | --- |
| **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 bajo determinadas condiciones, así como analizar e interpretar los resultados obtenidos, comparando los diferentes algoritmos. |
| **Instrucciones** | Utilizando programación genérica (templates) y sobrecarga de operadores en C++, programa una clase Sorter que incluya los siguientes métodos y atributos:  Métodos:  Selection sort  Bubble sort  Insertion sort  Merge sort  Quick sort  Cada método debe recibir por referencia el vector a ordenar.  Genere un arreglo de 100 000 números enteros de manera aleatoria.  Realice el ordenamiento de una copia del arreglo inicial (debemos hacer copiar sino se intentará ordenar un arreglo ya ordenado)  Mida el tiempo de ejecución de cada caso y complete las tablas que aparecen más adelante en este documento.  Genere algunas gráficas (en Google Sheets) comparando los resultados de todos los algoritmos y sus tiempos de ejecución.  Analice e interprete los resultados alcanzados.  Realice una copia de este documento en Google Docs y complete las secciones indicadas más adelante.  Suba a la plataforma Canvas el archivo con sus resultados.  Suba a Github todos los códigos programados.  No se aceptan trabajos fuera de fecha ni por correo electrónico. |
| **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  Computadora |
| **Tiempo estimado** | 5 horas |

# Respuestas

Repositorio de GitHub:

*https://github.com/DemiurgeApeiron/PDEDD/tree/master/sorting*

Tablas con los resultados de las mediciones:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algoritmos de ordenamiento | | | | | | | | | | | | |
| **Tabla de resultados a completar (***tiempo en s***)** | | | | | | | | | | | | |
| **Alg.** | **C1** | **C2** | **C3** | **C4** | **C5** | **C6** | **C7** | **C8** | **C9** | **C10** | **Media** | **Stdev** |
| Selection sort | 101.236 | 101.349 | 100.194 | 98.5769 | 106.093 | 103.055 | 110.62 | 102.29 | 116.058 | 106.083 | 104.55549 | 5.53387899 |
| Bubble sort | 151.434 | 147.615 | 146.764 | 149.116 | 148.667 | 159.14 | 159.744 | 158.466 | 185.212 | 158.113 | 156.4271 | 11.9156479 |
| Insertion sort | 28.5509 | 27.5857 | 27.7335 | 27.804 | 27.8276 | 27.8392 | 27.9969 | 28.0731 | 51.6981 | 29.3029 | 30.44119 | 7.48590461 |
| Merge Sort | 0.30946 | 0.293418 | 0.402404 | 0.30763 | 0.299032 | 0.311853 | 0.297879 | 0.405277 | 0.383917 | 0.319798 | 0.3330668 | 0.04522374 |
| Quick sort |  |  |  |  |  |  |  |  |  |  |  |  |

Interpretación de los resultados:

*Al observar los resultados uno puede percatarse de cómo hay una gran diferencia entre los algoritmos, Por ejemplo es bastante visible que el algoritmo de bubbleSort fue el más lento de todos con un promedio de 104.55549s esto es debido a qué tiene una complejidad de O(n^2), ya que este algoritmo en el peor de los casos recorre toda la lista al cuadrado, ya que es un for loop anidado. En cuarto lugar queda el algoritmo selectionSort con un promedio de 156.4271s, ya que este compara cada numero contra uno espesifico, sin embargo este algoritmo también tiene una complejidad de O(n^2) debido a que en el peor de los casos también va a recorrer toda la lista al cuadrado. En tercer lugar queda el algoritmo insertion Sort con un promedio de 30.44119s, ya que este genra una lista de elementos ordenados y una de desordenados, por lo cual cuando encentra la posicion adecada de un numero en la lista ordenada se detiene ya que el resto de la lista esta ordenada, en el peor de los casos va a tener una complejidad temporal O(n^2) porque si los numeros estan invertidos en la lista ordenada el nuevo numero va a ser menor a todos en la lista ordenada. En segundo lugar queda el mergeSort el segundo más rápido con un promedio de 0.3330668s esto es debido que este algoritmo corta la lista a la mitad por cada secuencia y se bifurca por ambos lados y luego los junta ordenadamente, pero como los va ordenando por bifuracion solo recorre el imput una ves, cómo corta el input a la mitad cada vez que se ejecuta y luego los junta recorriendo el imput una ves tiene una complejidad temporal de O(nlog(n)) la cual si se busca en una tabla de complejidades temporales es una de las menores complejidades que hay para la organisacion de numeros.*

Gráficas comparativas