UNIVERSIDAD DEL VALLE DE GUATEMALA

CC 2008 - Programación Orientada a Objetos

Sección 31

Ing. MICHAELLE PEREZ RIZ

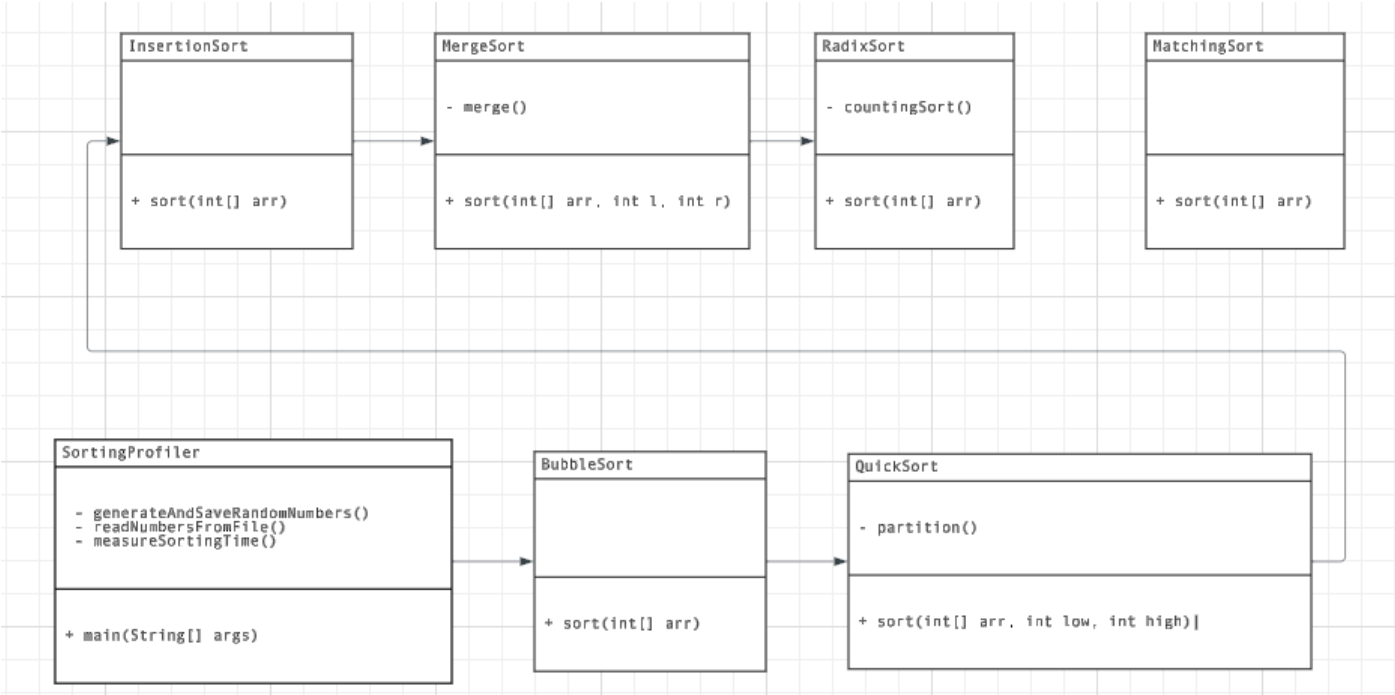
Laboratorio 4: Interfaces

Integrantes

Crista Dieguez
Manuel Ulin

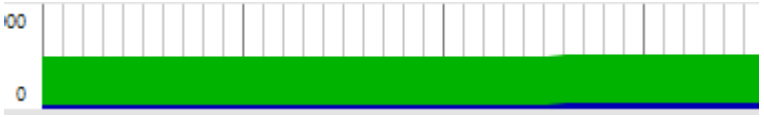
GUATEMALA, 8 de febrero de 2024

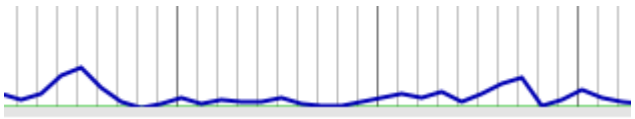
UML



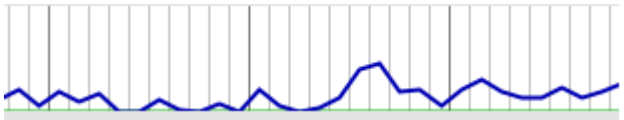
Informe de los resultados

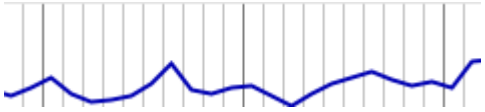
Insertion sort:



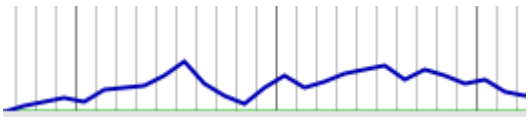


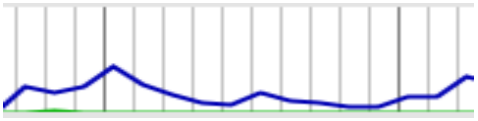
Matching sort:



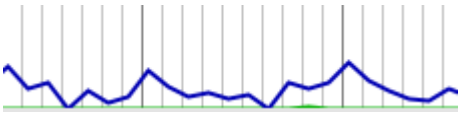


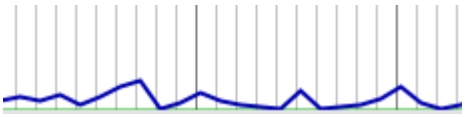
Merge Sort:



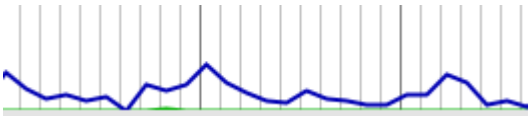


QuickSort:





RadixSort



Se puede observar que van cambiando el tiempo de las gráficas dependiendo de cada uno de los sort que se implementó y se midió con JProfiler para poder sacar esas comparaciones

Generación y Guardado de Números Aleatorios:

Se genera una serie de números aleatorios y se guardan en un archivo llamado random_numbers.txt. Estos números se utilizan como datos de entrada para los algoritmos de ordenación.

Iteración sobre Algoritmos de Ordenación:

Se itera sobre cada algoritmo de ordenación (BubbleSort, QuickSort, InsertionSort, MergeSort, RadixSort, MatchingSort).
Para cada algoritmo, se ejecuta el algoritmo para diferentes tamaños de datos (10, 110, 210, ..., 3000) y se mide el tiempo de ejecución.
Los tiempos de ejecución se guardan en archivos separados, uno para cada algoritmo, con el formato "nombre_del_algoritmo_times.txt".
Medición del Tiempo de Ejecución:

-Para medir el tiempo de ejecución de cada algoritmo, se utiliza System.nanoTime() antes y después de ejecutar el algoritmo.
La diferencia entre estos dos tiempos se calcula para obtener el tiempo de ejecución del algoritmo en nanosegundos.

Este tiempo se imprime en la consola junto con el nombre del algoritmo y el tamaño de los datos.

Ordenación de Datos Ordenados:

-Después de medir los tiempos de ejecución para datos aleatorios, se ordenan los números generados previamente para tener datos ordenados.
-Se repite el proceso de medición del tiempo de ejecución para cada algoritmo utilizando estos datos ordenados.
-Los tiempos de ejecución para datos ordenados también se guardan en archivos separados.

En resumen, el código SortingProfiler.java genera datos aleatorios, ejecuta varios algoritmos de ordenación para diferentes tamaños de datos, mide los tiempos de ejecución de cada algoritmo y guarda estos tiempos en archivos separados tanto para datos aleatorios como ordenados. Esto proporciona una comparación de rendimiento entre los diferentes algoritmos de ordenación bajo diferentes condiciones de entrada.