Evidencia 3 Estructura de datos

Mauricio Estrada de la Garza
AL02976904
Igor Sung Min Kim Juliao
AL02829189
Juan Pablo Hernandez Parra
AL02887299

Reporte:
El algoritmo tiene solo una clase public class evidencia 3 y utiliza los imports:
import java.util.*;
import java.util.concurrent.*;

El código empieza por el main el que hace un ciclo booleano con un while para hacer un menu donde existe un scanner.next int para captar tu respuesta, si tu respuesta es 1 llama la funcion algoritmo, si tu respuesta es 2 hace un break y termina de correr el ciclo, si tu respuesta es otro número, repite el menú pidiendote seleccionar una opción válida.

Después tenemos la función algoritmo() donde:

- Solicita al usuario que ingrese el tiempo total de ejecución en segundos mediante la entrada estándar.
- Crea un ExecutorService con un pool fijo de 6 hilos para manejar la ejecución de las tareas de ordenamiento de manera concurrente.
- Inicializa dos mapas para almacenar el número de colecciones ordenadas por cada algoritmo y el tiempo promedio por colección para cada algoritmo, respectivamente. También inicializa una lista de objetos Future que contendrán los resultados de las tareas de ordenamiento.
- Define una lista de tamaños de arreglos que se utilizarán en las pruebas de ordenamiento.
- Itera sobre cada algoritmo de ordenamiento y tamaño de arreglo. Para cada combinación de algoritmo y tamaño de arreglo, crea una tarea y la envía al ExecutorService para su ejecución.
- Cada tarea genera una lista de números aleatorios, ordena la lista utilizando el algoritmo correspondiente y registra el tiempo que tarda en hacerlo. Continúa realizando esto hasta que haya transcurrido el tiempo total especificado por el usuario.
- Registra el número total de colecciones ordenadas por cada algoritmo y calcula el tiempo promedio por colección para cada algoritmo.
- Utiliza los objetos Future para esperar a que todas las tareas de ordenamiento se completen.
- Una vez que todas las tareas se han completado, se cierra el ExecutorService.
- Ordena los algoritmos por su eficiencia, medida como el tiempo promedio por colección.
- Imprime los resultados de las pruebas de ordenamiento, mostrando el número de colecciones ordenadas y el tiempo promedio por colección para cada algoritmo. El

tiempo se muestra en milisegundos si es menor a 1000, de lo contrario se muestra en segundos.

Luego tenemos la función ordenarLista(), la que crea un thread para cada uno de los metodos de ordenamiento.

Después de eso tenemos public static List<Integer> generarLista(int size) que crea un arraylist, que guarda numeros aleatorios del 0 al 999, tenemos public static List<Integer> generarListaV2(int size), que hace lo mismo pero con numeros del 1 al 5.

Ahora tenemos la funciones para ordenar las listas la primera es
public static void bubbleSortLista(List<Integer> lista) que usa dos ciclo for anidados y un if para ordenar la lista con el método bubble sort.

Después tenemos public static void selectionSortLista(List<Integer> lista) que usa 2 ciclos for anidados y un if para ordenar con el método selection sort.

Luego tenemos public static void insertionSortLista(List<Integer> lista) que ordena la lista usando el método insertion sort a partir de un while anidado a un ciclo for.

También tenemos el método public static void shellSortLista(List<Integer> lista) que ordena la lista con un shell sort hecho a partir de 3 ciclos for anidados.

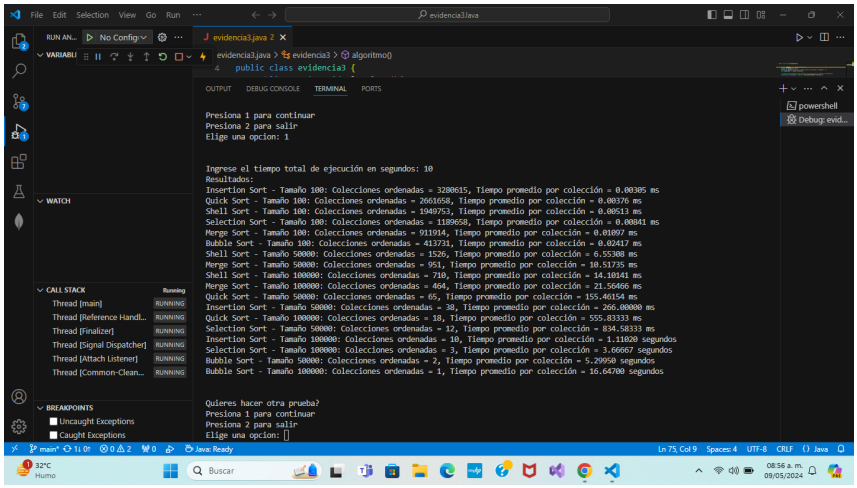
Finalmente tenemos el ordenamiento de mergesort que esta dividido en varias funciones la primera siendo private static void mergeSortLista(List<Integer> lista) que divide la lista en mitades.

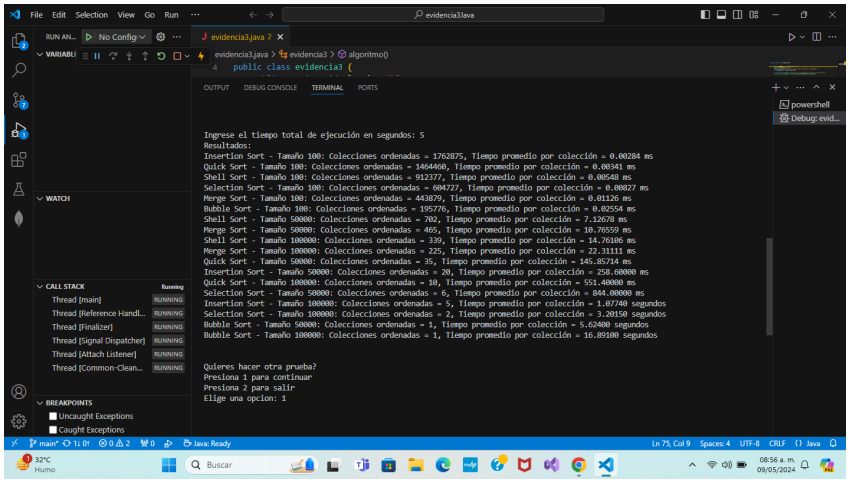
Y como ultima funcion de merge sort tenemos private static void mergeLista(List<Integer> leftList, List<Integer> rightList, List<Integer> lista) que ordena estas mitades dependiendo de su tamaño usando un while con un if y un else y otros dos ciclos while para que siga el ciclo hasta que la lista esté ordenada.

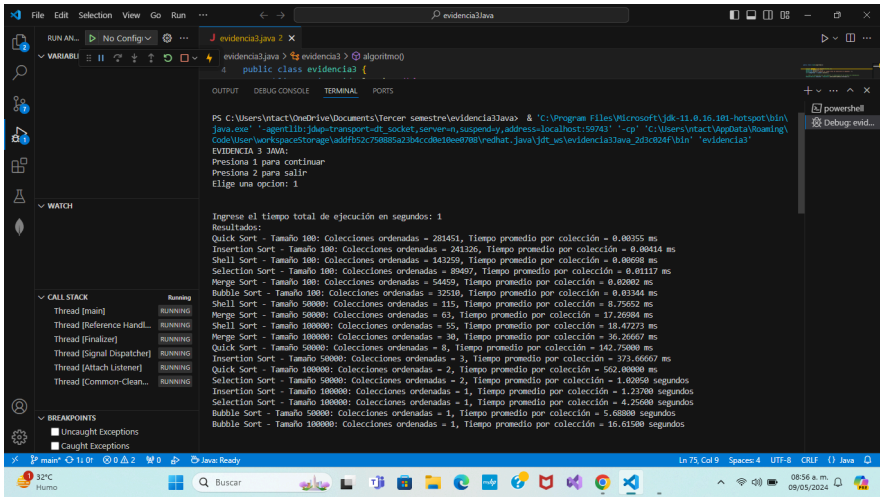
Ahora tenemos private static void quickSortLista(List<Integer> lista, int start, int end), que se llama a sí misma de manera recursiva y a la función private static int partitionLista(List<Integer> lista, int start, int end) para ordenarse en forma de quick sort.

Como última función de este algoritmo tenemos private static int partitionLista(List<Integer> lista, int start, int end) que usa un ciclo for con un if y usa la función swap para marcar el principio y el final de la lista para que el quick sort pueda ordenarla a partir de llamar esta función.

PRUEBAS:


Presiona 1 para continuar
Presiona 2 para salir
Elige una opcion: 1
Ingrese el tiempo total de ejecución en segundos: 10
Resultados:
Insertion Sort - Tamaño 100: Colecciones ordenadas = 3280615, Tiempo promedio por colección = 0.00305 ms
Quick Sort - Tamaño 100: Colecciones ordenadas = 2661658, Tiempo promedio por colección = 0.00376 ms
Shell Sort - Tamaño 100: Colecciones ordenadas = 1949753, Tiempo promedio por colección = 0.00513 ms
Selection Sort - Tamaño 100: Colecciones ordenadas = 1189658, Tiempo promedio por colección = 0.00841 ms
Merge Sort - Tamaño 100: Colecciones ordenadas = 911914, Tiempo promedio por colección = 0.01097 ms
Bubble Sort - Tamaño 100: Colecciones ordenadas = 413731, Tiempo promedio por colección = 0.02417 ms
Shell Sort - Tamaño 50000: Colecciones ordenadas = 1526, Tiempo promedio por colección = 6.55308 ms
Merge Sort - Tamaño 50000: Colecciones ordenadas = 951, Tiempo promedio por colección = 10.51735 ms
Shell Sort - Tamaño 100000: Colecciones ordenadas = 710, Tiempo promedio por colección = 14.10141 ms
Merge Sort - Tamaño 100000: Colecciones ordenadas = 464, Tiempo promedio por colección = 21.56986 ms
Quick Sort - Tamaño 50000: Colecciones ordenadas = 65, Tiempo promedio por colección = 155.46154 ms
Insertion Sort - Tamaño 50000: Colecciones ordenadas = 38, Tiempo promedio por colección = 266.00000 ms
Quick Sort - Tamaño 100000: Colecciones ordenadas = 18, Tiempo promedio por colección = 555.83333 ms
Selection Sort - Tamaño 50000: Colecciones ordenadas = 12, Tiempo promedio por colección = 834.58333 ms
Insertion Sort - Tamaño 100000: Colecciones ordenadas = 10, Tiempo promedio por colección = 1.11020 segundos
Selection Sort - Tamaño 100000: Colecciones ordenadas = 3, Tiempo promedio por colección = 3.66667 segundos
Bubble Sort - Tamaño 50000: Colecciones ordenadas = 2, Tiempo promedio por colección = 5.29950 segundos
Bubble Sort - Tamaño 100000: Colecciones ordenadas = 1, Tiempo promedio por colección = 16.64700 segundos
Quieres hacer otra prueba?
Presiona 1 para continuar
Presiona 2 para salir
Elige una opcion:


Ingrese el tiempo total de ejecución en segundos: 5
Resultados:
Insertion Sort - Tamaño 100: Colecciones ordenadas = 1762875, Tiempo promedio por colección = 0.00284 ms
Quick Sort - Tamaño 100: Colecciones ordenadas = 1464400, Tiempo promedio por colección = 0.00341 ms
Shell Sort - Tamaño 100: Colecciones ordenadas = 912377, Tiempo promedio por colección = 0.00548 ms
Selection Sort - Tamaño 100: Colecciones ordenadas = 604727, Tiempo promedio por colección = 0.00827 ms
Merge Sort - Tamaño 100: Colecciones ordenadas = 443879, Tiempo promedio por colección = 0.01126 ms
Bubble Sort - Tamaño 100: Colecciones ordenadas = 195776, Tiempo promedio por colección = 0.02554 ms
Shell Sort - Tamaño 50000: Colecciones ordenadas = 702, Tiempo promedio por colección = 7.12678 ms
Merge Sort - Tamaño 50000: Colecciones ordenadas = 465, Tiempo promedio por colección = 10.76559 ms
Shell Sort - Tamaño 100000: Colecciones ordenadas = 339, Tiempo promedio por colección = 14.76106 ms
Merge Sort - Tamaño 100000: Colecciones ordenadas = 225, Tiempo promedio por colección = 22.31111 ms
Quick Sort - Tamaño 50000: Colecciones ordenadas = 35, Tiempo promedio por colección = 145.85714 ms
Insertion Sort - Tamaño 50000: Colecciones ordenadas = 20, Tiempo promedio por colección = 258.60000 ms
Quick Sort - Tamaño 100000: Colecciones ordenadas = 10, Tiempo promedio por colección = 551.40000 ms
Selection Sort - Tamaño 50000: Colecciones ordenadas = 6, Tiempo promedio por colección = 844.00000 ms
Insertion Sort - Tamaño 100000: Colecciones ordenadas = 5, Tiempo promedio por colección = 1.07740 segundos
Selection Sort - Tamaño 100000: Colecciones ordenadas = 2, Tiempo promedio por colección = 3.20150 segundos
Bubble Sort - Tamaño 50000: Colecciones ordenadas = 1, Tiempo promedio por colección = 5.62400 segundos
Bubble Sort - Tamaño 100000: Colecciones ordenadas = 1, Tiempo promedio por colección = 16.89100 segundos
Quieres hacer otra prueba?
Presiona 1 para continuar
Presiona 2 para salir
Elige una opcion: 1


EVIDENCIA 3 JAVA:
Presiona 1 para continuar
Presiona 2 para salir
Elige una opcion: 1
Ingrese el tiempo total de ejecución en segundos: 1
Resultados:
Quick Sort - Tamaño 100: Colecciones ordenadas = 281451, Tiempo promedio por colección = 0.00355 ms
Insertion Sort - Tamaño 100: Colecciones ordenadas = 241326, Tiempo promedio por colección = 0.00414 ms
Shell Sort - Tamaño 100: Colecciones ordenadas = 143259, Tiempo promedio por colección = 0.00698 ms
Selection Sort - Tamaño 100: Colecciones ordenadas = 89497, Tiempo promedio por colección = 0.01117 ms
Merge Sort - Tamaño 100: Colecciones ordenadas = 54459, Tiempo promedio por colección = 0.02082 ms
Bubble Sort - Tamaño 100: Colecciones ordenadas = 32510, Tiempo promedio por colección = 0.03344 ms
Shell Sort - Tamaño 50000: Colecciones ordenadas = 115, Tiempo promedio por colección = 8.75652 ms
Merge Sort - Tamaño 50000: Colecciones ordenadas = 63, Tiempo promedio por colección = 17.26984 ms
Shell Sort - Tamaño 100000: Colecciones ordenadas = 55, Tiempo promedio por colección = 18.47273 ms
Merge Sort - Tamaño 100000: Colecciones ordenadas = 30, Tiempo promedio por colección = 36.26667 ms
Quick Sort - Tamaño 50000: Colecciones ordenadas = 8, Tiempo promedio por colección = 142.75000 ms
Insertion Sort - Tamaño 50000: Colecciones ordenadas = 3, Tiempo promedio por colección = 373.66667 ms
Quick Sort - Tamaño 100000: Colecciones ordenadas = 2, Tiempo promedio por colección = 562.00000 ms
Selection Sort - Tamaño 50000: Colecciones ordenadas = 2, Tiempo promedio por colección = 1.02050 segundos
Insertion Sort - Tamaño 100000: Colecciones ordenadas = 1, Tiempo promedio por colección = 1.23700 segundos
Selection Sort - Tamaño 100000: Colecciones ordenadas = 1, Tiempo promedio por colección = 4.25600 segundos
Bubble Sort - Tamaño 50000: Colecciones ordenadas = 1, Tiempo promedio por colección = 5.68800 segundos
Bubble Sort - Tamaño 100000: Colecciones ordenadas = 1, Tiempo promedio por colección = 16.61500 segundos

CONCLUSIONES:

Mau: Lo bueno que ya teníamos hechos los algoritmos de ordenamiento, de lo contrario este reto hubiera sido más complicado de hacer con eso que se ocupaban diccionarios y arraylists.

Juan Pablo: Que bueno que durante todo el semestre estos retos se hayan hecho en equipo porque cuando uno lo hace solo es más complicado, el trabajo en equipo en la informática es clave para el éxito.

Igor: Este proyecto se me hizo el más complicado de todos los proyectos hasta ahora, pero al mismo tiempo fue uno de los que mejor entendí, lo que más me confundía era como ordenar las listas, pero una vez ya lo hubiéramos hecho el resto era fácil y me gusto mucho de este proyecto trabajar con threads y concurrencia, me hizo pensar en cosas que no eran posibles antes de, y puede que esta vez hayamos usado los threads para hacer comparaciones entre los tiempos de cada método de ordenamiento, pero ellos me hicieron pensar en que se puede hacer un algoritmo para algo mucho más rápido si tan solo dividimos en threads, al igual que muchas otras posibilidades y por lo tanto creo que fue el proyecto del que más aprendí en todo el semestre.