Actividad 13 Estructura de datos

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

Reporte:
El código empieza con el main el que manda llamar una función llamada algoritmo().
fuera del main esta la función algoritmo en donde se implementa un menú para seleccionar entre dos algoritmos de ordenamiento: Merge Sort y Quick Sort. Vamos a analizarla paso a paso:

1. Inicio del método algoritmo():
    - Se declara una variable ciclo de tipo booleano y se inicializa en true. Esta variable controlará el ciclo principal del programa.
2. Inicio del bucle while(ciclo):
    - Este bucle se ejecutará mientras ciclo sea verdadero, lo que significa que el programa continuará ejecutándose hasta que el usuario decida salir.
3. Declaración de variables dentro del bucle while:
    - Se crea un objeto Scanner para leer la entrada del usuario.
    - Se declaran variables para medir el tiempo de inicio, fin y duración de la ejecución de los algoritmos.
    - Se crea un arreglo calificaciones que contiene una serie de valores enteros.
4. Menú de selección de algoritmos:
    - Se muestra un mensaje pidiendo al usuario que elija entre Merge Sort (1) o Quick Sort (2).
    - El usuario introduce su opción mediante la entrada estándar.
5. Manejo de la opción seleccionada:
    - Si el usuario elige 1 (Merge Sort), se ejecuta el algoritmo de Merge Sort sobre el arreglo de calificaciones.
    - Si elige 2 (Quick Sort), se ejecuta el algoritmo de Quick Sort sobre el arreglo de calificaciones.
    - En ambos casos, se muestra el arreglo ordenado en la consola.
6. Medición del tiempo de ejecución:
    - Se toma el tiempo de inicio justo antes de ejecutar el algoritmo de ordenamiento.
    - Se toma el tiempo de finalización justo después de que el algoritmo haya terminado.
    - Se calcula la duración de la ejecución restando el tiempo de inicio del tiempo de finalización.
7. Mostrar tiempo de ejecución:
    - Se muestra en la consola el tiempo de ejecución del algoritmo seleccionado.

8. Preguntar al usuario si desea continuar o salir:
    - Se muestra un mensaje pidiendo al usuario que elija entre continuar (1) o salir (2).
    - El usuario introduce su opción mediante la entrada estándar.
9. Manejo de la opción seleccionada:
    - Si elige continuar (1), se vuelve al inicio del bucle while para permitir al usuario seleccionar otro algoritmo.
    - Si elige salir (2), se cambia el valor de ciclo a false, lo que termina el bucle while y, por lo tanto, el programa.

Luego está la función private static void mergeSort(int[] array) la cual implementa el algoritmo Merge Sort para ordenar un arreglo de enteros en orden ascendente. Aquí está una descripción paso a paso:

1. Comprobación del tamaño del arreglo:
    - Se obtiene la longitud del arreglo array.
    - Si la longitud del arreglo es igual o menor que 1, se devuelve inmediatamente, ya que un arreglo con 0 o 1 elemento ya está ordenado (este es el caso base de la recursión).
2. División del arreglo en mitades:
    - Se calcula el índice medio del arreglo y se almacena en la variable middle.
    - Se crean dos nuevos arreglos: leftArray para almacenar la mitad izquierda del arreglo original y rightArray para almacenar la mitad derecha del arreglo original.
3. División de elementos del arreglo original:
    - Se inicializan dos índices, i para la mitad izquierda y j para la mitad derecha.
    - Se itera sobre el arreglo original y se distribuyen sus elementos en los arreglos leftArray y rightArray según su posición en el arreglo original.
4. Llamadas recursivas a Merge Sort:
    - Se llama recursivamente a mergeSort para ordenar los arreglos leftArray y rightArray.
5. Unión de los arreglos ordenados:
    - Se llama al método merge para combinar los arreglos leftArray y rightArray en el arreglo original array.

Después está la función private static void merge(int[] leftArray, int[] rightArray, int[] array) que toma dos arreglos ordenados (leftArray y rightArray) y los combina en un único arreglo ordenado (array). Aquí está la descripción paso a paso:

1. Inicialización de variables:
    - Se calculan las longitudes de los subarreglos izquierdo (leftSize) y derecho (rightSize) del arreglo original.
    - Se inicializan tres índices: i para recorrer el arreglo resultante, l para recorrer leftArray y r para recorrer rightArray.
2. Merge de los subarreglos:

- Se utiliza un bucle while para recorrer ambos subarreglos mientras haya elementos en ambos.
- En cada iteración, se compara el elemento actual en leftArray[l] con el elemento actual en rightArray[r].
    - Si leftArray[l] es menor que rightArray[r], se coloca leftArray[l] en array[i] y se incrementa tanto i como l.
    - Si rightArray[r] es menor o igual a leftArray[l], se coloca rightArray[r] en array[i] y se incrementa tanto i como r.

3. Completar elementos restantes:
    - Después de que uno de los subarreglos se haya recorrido completamente, se pueden quedar elementos restantes en el otro subarreglo.
    - Se utilizan dos bucles while adicionales para copiar los elementos restantes de leftArray y rightArray en array.
        - Si aún quedan elementos en leftArray, se copian al arreglo resultante array.
        - Si aún quedan elementos en rightArray, también se copian al arreglo resultante array.

Estas son todas las funciones para mergesort.

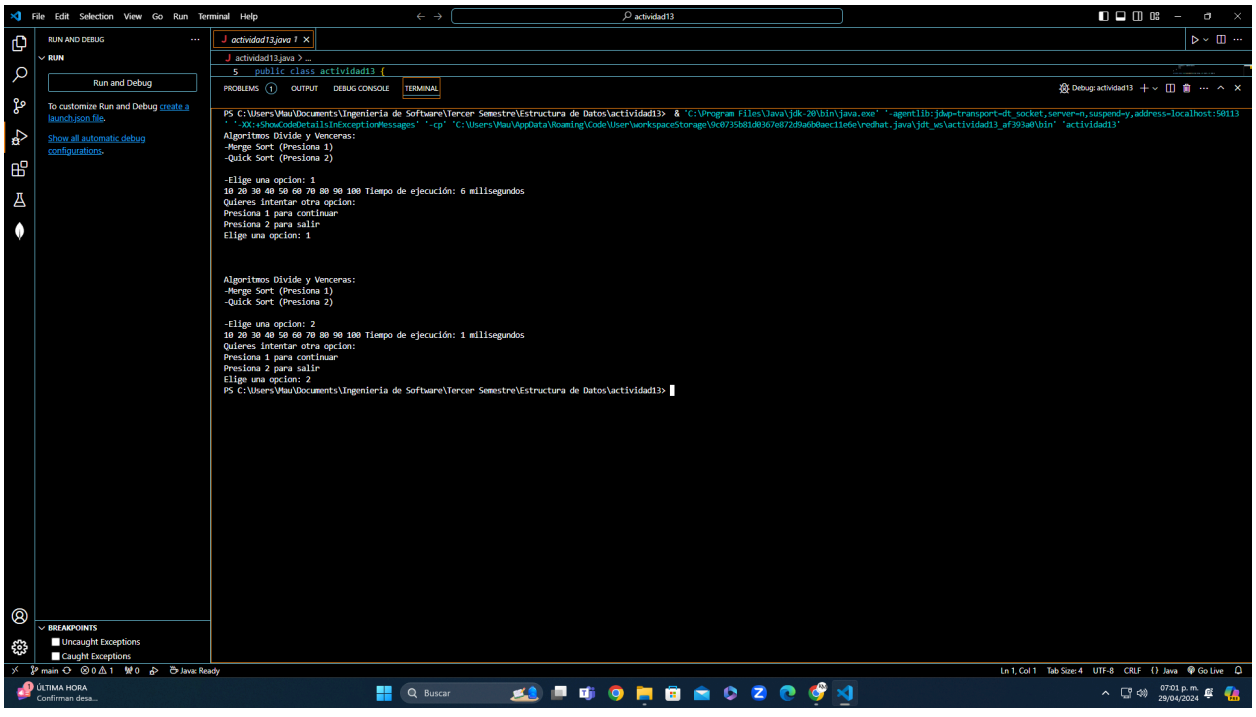
Ahora esta private static void quickSort(int[] array, int start, int end) el cual implementa el algoritmo de ordenamiento Quick Sort para ordenar un arreglo de enteros en orden ascendente. Aquí tienes una descripción paso a paso:

1. Verificación del caso base:
    - Se verifica si el índice end es menor o igual al índice start. Si es así, significa que el subarreglo a ordenar tiene cero o un elemento, por lo que ya está ordenado y se devuelve inmediatamente sin hacer más operaciones (esto es el caso base de la recursión).
2. Selección del pivote y partición del arreglo:
    - Se llama al método partition para seleccionar un pivote y particionar el arreglo en dos subarreglos alrededor del pivote. El pivote se coloca en su posición correcta en el arreglo, de manera que todos los elementos menores que el pivote estén a su izquierda y todos los elementos mayores que el pivote estén a su derecha.
3. Recursión en los subarreglos:
    - Se llama recursivamente a quickSort para ordenar el subarreglo izquierdo (desde el inicio hasta la posición del pivote menos uno).
    - Se llama recursivamente a quickSort para ordenar el subarreglo derecho (desde la posición del pivote más uno hasta el final).
4. Proceso de división y conquista:
    - El algoritmo se divide en subproblemas más pequeños al seleccionar un pivote y particionar el arreglo en dos subarreglos. Luego, se resuelven estos subproblemas ordenando recursivamente los subarreglos izquierdo y derecho.

y por ultimo el método private static int partition(int[] array, int start, int end) se encarga de seleccionar un pivote dentro del arreglo y particionar el arreglo alrededor de ese pivote. Aquí tienes una descripción paso a paso:

1. Selección del pivote:
    - El pivote se elige como el último elemento del arreglo (array[end]).
2. Inicialización de variables:
    - Se inicializa un índice i en start - 1, que será utilizado para realizar el intercambio de elementos menores que el pivote.
3. Recorrido del arreglo:
    - Se itera a través del arreglo desde start hasta end utilizando un bucle for.
    - En cada iteración, se compara el elemento actual del arreglo (array[j]) con el pivote.
    - Si el elemento es menor que el pivote, se incrementa el índice i y se intercambian los elementos array[i] y array[j]. Esto coloca los elementos menores que el pivote en la parte izquierda del arreglo.
4. Intercambio final del pivote:
    - Después de que se haya recorrido todo el arreglo, se intercambia el elemento en la posición i + 1 (donde i apunta al último elemento menor que el pivote) con el pivote (array[end]). Esto coloca el pivote en su posición final en el arreglo, de manera que todos los elementos a su izquierda son menores que él y todos los elementos a su derecha son mayores que él.
5. Retorno del índice del pivote:
    - Se retorna el índice i + 1, que ahora indica la posición del pivote en el arreglo después de la partición.

Pruebas:



Reflexiones:

Igor: Este código, se me hizo demasiado fácil, pues al fin y al cabo son cosas que ya hicimos de una manera más difícil en la actividad anterior por lo tanto no lo veo como algo innovador ni nada que requiera mucha atención, sino que pudimos hacer el código más corto y compacto porque no tenía tanto que ejecutar, sino que solo fue una prueba de una pequeña comparación entre el quick sort y el mergesort, por lo que no tenía mucha ciencia.

Juan Pablo:está actividad fue más sencilla comparada con la otra actividad anterior ya que vimos lo mismo pero con menos complejidad o al menos eso sentimos todos, fue entretenido y hasta y me es raro decirlo muy divertido.

Mau: Por suerte que esta actividad fue más sencilla que las tareas anteriores, y ahora que estamos a nada de terminar el semestre, me pondré a repasar todos los trabajos y programare los algoritmos en JavaScript ya que es el lenguaje que ahorita mas domino y además de ayudarme a mejorar mi logica de programacion y resolver problemas mas facil, me pondran retos de estos temas en entrevistas de trabajo.