Prueba de Escritorio de Algoritmos de Ordenamiento

Estructura de datos

Rodrigo Legarreta Soto

09/03/2025

15068

Código Main

```java
/*
 * codigo main del segundo parcial
 * Autor: Rodrigo Legarreta Soto
 * matricula: 15068
 * dia: 06/03/2025
 */

public class SecondPartialMain {
    public static void main(String[] args) {
        //insertion sort
        System.out.println("insertion sort: ");

        int peorCaso[] = {5, 4, 3, 2, 1}; //se pone la lista en inverso para
ver el peor caso

        System.out.print("Peor Caso:");
        int operacionesPeor = InsertionSortExample.insertionSort(peorCaso);

        System.out.println("\nOperaciones realizadas: " + operacionesPeor);
//se manda a llamar la clase de insertion con sus variables para que compare
el arreglo

        System.out.println();

        //selection sort
        System.out.println("selection sort: ");
        int worstCase[] = {5, 4, 3, 2, 1};

        System.out.println("Peor caso:");
        System.out.println("Operaciones realizadas: " +
SelectionSortExample.selectionSort(worstCase));

        System.out.println();

        //bubble sort
        System.out.println("bubble sort: ");
        System.out.println("Peor caso:");
        int[] arr = {5, 4, 3, 2, 1}; // Peor caso (orden inverso)
        int comparisons = BubbleSortExample.bubbleSort(arr);
        System.out.println("operaciones realizadas: " + comparisons);

    }
}
```

Ordenamiento por inserción

El ordenamiento por inserción asume que el primer valor ya está ordenado, así que agarra el siguiente numero y los va comparando hacia la izquierda, de esta forma manda al que mayo valor tenga al lado derecho de la lista.

```
/*
 * Ejemplo de insertion sort
 * Autor: Rodrigo Legarreta Soto
 * matricula: 15068
 * dia: 06/03/2025
 * formula para el peor caso
 * T(n) = (n(n - 1)) / 2
 */

 // notacion del big o o(n^2)
public class InsertionSortExample {
    public static int insertionSort(int arr[]) { //aqui se crea la clase de
insertion sort para luego poder llamarla
        int n = arr.length;
        int operaciones = 0; //el contador sirve para ver el numero de
comparaciones que se hace
        for (int i = 1; i < n; i++) {// este bucle sirve para recorrer el
arreglo y compararlo para saber cual mover
            int key = arr[i];//se usa la variable key para guardar los
valores y poder compararlos
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {// este while sirve para
recorrer los valores mas grandes hacia la derecha
                arr[j + 1] = arr[j];
                j = j - 1;
                operaciones++;//el contador ve los movimientos y los va
guardando
            }
            arr[j + 1] = key;

        }
        return operaciones;
    }
}

//este programa agarra el primer valor y lo compara con todos los elementos
del arreglo y así saber cuál mover.
```

Prueba de escritorio

Iteración	i	key	Comparación	Movimiento	Resultado
1	1	4	5 > 4	arr[1] = arr[0] (5)	[4, 5, 3, 2, 1]
2	2	3	5 > 3	arr[2] = arr[1] (5)	[4, 3, 5, 2, 1]
			4 > 3	arr[1] = arr[0] (4)	[3, 4, 5, 2, 1]
3	3	2	5 > 2	arr[3] = arr[2] (5)	[3, 4, 2, 5, 1]
			4 > 2	arr[2] = arr[1] (4)	[3, 2, 4, 5, 1]
			3 > 2	arr[1] = arr[0] (3)	[2, 3, 4, 5, 1]
4	4	1	5 > 1	arr[4] = arr[3] (5)	[2, 3, 4, 1, 5]
			4 > 1	arr[3] = arr[2] (4)	[2, 3, 1, 4, 5]
			3 > 1	arr[2] = arr[1] (3)	[2, 1, 3, 4, 5]
			2 > 1	arr[1] = arr[0] (2)	[1, 2, 3, 4, 5]

Total, de comparaciones: 10

la i representa el 4 porque el numero 1 es 4 y el numero 0 es 5

i es igual a 4 porque es el primer numero que agarramos de la lista ya que el 5 asumimos que esta ordenado

al momento de agarrar 4 como nuestro primer numero se compara hacia la izquierda con el 5, vemos que es menor el 4 asi que se mueve hacia la izquierda

ahora tenemos que agarrar nuestro numero 2 en la lista para compararlo, el numero 2 de nuestra lista es el 3, ese se convierte en nuestro key porque sera el numero comparado, el numero 3 se compara con la nueva lista osea con el 5 y con el 4 y se manda hasta la derecha

asi va sucesivamente comparando el siguiente numero en la lista y convirtiendolo en nuestro key para que se pueda mover, al final se sustituye el j = -1 pero no lo puse para hacer mas simple la explicacion que, pero esta j sirve para sustituir el valor de nuestra key en nuestra lista, asi es como el programa mueve los datos

en la columna de movimientos puse las pocisiones que se iban a intercambiar

Justificación de la complejidad en Big O

Nuestra tabla de inserción muestra una complejidad de n^2, ya que en nuestro código key es comparado con todos los elementos de la izquierda, en el peor caso significa que key siempre será menor numero de la lista ya que esta misma esta en orden inverso, lo que quiere decir que siempre va a tener que recorrer todos los elementos de la lista y esto es muy tardado, esto usa la fórmula de n(n−1)2\frac{n(n-1)}{2}2n(n−1), al tener que recorrer todos los elementos ya que la lista esta en inverso no conviene y por eso tiene esa complejidad.

Conclusión del ordenamiento por inserción

Este código se me hizo medio complejo de entender gracias a la key, en el peor caso lo veo bien inútil ya que compara el key siendo el menor, con cada numero del arreglo y eso es muy tardado, si tuviéramos una lista casi ordenada no habría problema alguno pero esto lo veo confuso e inútil, digamos que mi key está en la iteración 3 de mi lista, al ser el peor caso ese numero seria el menor, entonces tengo que agarrarlo y compararlo con cada uno de los elementos a la izquierda y ponerlo hasta al principio lo cual es tardado.

Ordenamiento por selección

Este código se basa en buscar el elemento mas pequeño siempre, se agarra ese numero y se compara hasta que este quede ordenado, al momento de ya revisar y comparar un elemento no hace falta volver a hacer comparaciones.

```java
/*
 * Ejemplo de selection sort
 * Autor: Rodrigo Legarreta Soto
 * matricula: 15068
 * dia: 06/03/2025
 * formula para el peor caso
 * T(n) = (n(n - 1)) / 2
 */

  // notacion del big o o(n^2)
public class SelectionSortExample {
    public static int selectionSort(int arr[]) {
        int n = arr.length;
        int comparisons = 0;

        for (int i = 0; i < n - 1; i++) { //aqui se inicia el bucle for n representa el valor que sera cambiado de pocision, es -1 porque ahi quedara el dato ordenadp
            int minIndex = i;//miniindex guarda el elemento mas pequeno
            for (int j = i + 1; j < n; j++) {
                comparisons++;
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;//cuando se inicia el bucle se comparan todos los valores del arreglo y agarra el mas pequeno y lo mete en miniindex osea j para que pueda ser cambiado
                }
            }
            // Intercambio de elementos
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;//Si minIndex cambió, significa que se encontró un número menor y hay que intercambiarlo con arr[i].
        }
        return comparisons;
    }
}
```

Prueba de escritorio

Iteración	i (posición a ordenar)	Mínimo	Comparaciones	Intercambio	Resultado
1	0	1 (en arr[4])	1<2(3), 1<3(2), 1<4(1), 1<5(0)	arr[0] <-> arr[4]	[1, 4, 3, 2, 5]
2	1	2 (en arr[3])	2<5(4), 2<3(2), 2<4(1)	arr[1] <-> arr[3]	[1, 2, 3, 4, 5]
3	2	3 (en arr[2])	3<4(3), 3>2(1)	arr[2] <-> arr[2] (sin cambio)	[1, 2, 3, 4, 5]
4	3	4 (en arr[3])	4<5(4)	arr[3] <-> arr[3] (sin cambio)	[1, 2, 3, 4, 5]
5	4	5 (en arr[4])	No hay	arr[4] <-> arr[4] (sin cambio)	[1, 2, 3, 4, 5]

Total, de comparaciones: 10

aqui se recorre el arreglo y encuentra el 1 como menor numero, hace comparaciones con todos los numeros hasta que encuentra su posicion y ahi se deja

entre mas se avanza menos comparaciones el codigo tiene que hacer porque mucho ya estan siendo ordenados

cuando encuentra un lugar solo cambia el orden en el que estan los datos y se va ordenando

Justificación de la complejidad en Big O

Este código en el peor caso también tiene una complejidad de n^2 ya que al igual que el pasado, aunque ya este ordenado el número se deben de hacer algunas comparaciones, tiene que recorrer todo el arreglo para encontrar el menor y luego comparar todos los números para que este pueda ser intercambiado, este código en el peor caso y en el caso promedio tiene la misma complejidad porque, aunque el arreglo ya este ordenado este tiene que hacer sus comparaciones lo cual lo hace muy ineficiente.

Conclusión del ordenamiento por selección

Este código es muy ineficiente, ya que, aunque le des una lista ordenada tiene que recorrer el arreglo para ver cual es el menor, y aun sabiendo cual es el mínimo los debe comparar con todos los números, si el arreglo ya esta en orden este compara el 1 con todos los demás aun sabiendo que es el mínimo, por eso este es mas ineficiente y tiene una complejidad de n^2 en todos los casos.

Ordenamiento por Burbuja

Este código se caracteriza porque compara en pares, los primeros dos números los compara y los intercambia, luego compara los siguientes dos números y así sucesivamente, este da varias pasadas del arreglo hasta que todos los números están ordenados.

```
 * Ejemplo de bubble sort
 * Autor: Rodrigo Legarreta Soto
 * matricula: 15068
 * dia: 05/03/2025
 * formula para el peor caso
 * T(n) = (n(n - 1)) / 2
// notacion del big o o(n^2)
public class BubbleSortExample {
    public static int bubbleSort(int[] arr) { //se crea la clase bubblesort
        int n = arr.length; //aqui se usa int para que el valor sea un
entero y se usa lenght para ver el tamano del arreglo
        int comparisons = 0; // este es un contador para poder ver el numero
de comparaciones del programa
        boolean swapped;

        for (int i = 0; i < n - 1; i++) {
            swapped = false;
            for (int j = 0; j < n - 1 - i; j++) { //aqui se usa un ciclo for
para el bubble sort en el que inicia en 0 en el que se comparan datos y se
intercambian elementos dentro de un arreglo
                comparisons++; // aqui se cuenta el numero de comparaciones
                if (arr[j] > arr[j + 1]) {

                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swapped = true;
                }
            }
            if (!swapped) break; // Si no hubo intercambios, ya está
ordenado
        }
        return comparisons;
    }

}
```

Prueba de escritorio

Iteración	Comparaciones	resultado
1	(5,4) , (5,3) , (5,2) , (5,1)	[4, 3, 2, 1, 5]
2	(4,3) , (4,2) , (4,1)	[3, 2, 1, 4, 5]
3	(3,2) , (3,1)	[2, 1, 3, 4, 5]
4	(2,1)	[1, 2, 3, 4, 5]
5	ya esta ordenado	[1, 2, 3, 4, 5]

Total, de comparaciones: 10

este codigo compara los numeros en pares, vemos que el 5 y 4 son los primeros y como el 5 es mayor se cambia de posicion, luego vuelve a comparar este 5 con el 3 porque ahora esos estan pares y asi sucesivamente

cada que mueve un numero al final o hasta donde sea el tope empieza la nueva iteracion con el nuevo numero que este al principio del arreglo

una vez ordenado ya no hace comparaciones

Justificación de la complejidad en Big O

Este código también en el peor caso tiene el o^2 ya que tiene que hacer muchas pasadas del arreglo para poder ordenarlos si está en orden inverso, aunque ya se hayan comparado algunos resultados aun así hará las pasadas necesarias para ordenar el arreglo, aunque en algunas partes ya este ordenado, este código en el mejor caso puede funcionar de una manera eficaz ya que no hace tanta comparación, pero en el peor caso si es complejo y medio lento.

Conclusión del ordenamiento por burbuja

Este código es más fácil de entender que los otros, pero no es tan eficaz como el de inserción, ya que hace más pasadas, aunque ya este medio ordenado el arreglo, puede ser bueno en otros casos, pero para esta lista no lo es tanto, al tener valores inversos hace muchas comparaciones innecesarias.

Conclusión general

Al entender los tres códigos me di cuenta del porque son los menos eficaces de todos, el de burbuja fue mi favorito ya que es el más fácil de entender, eso no significa que sea el mejor código ya que tiene muchas cosas innecesarias como el de pasar varias veces por el arreglo aunque ya este parcialmente ordenado, siento que el mas eficaz y a la vez complejo es el de inserción, a pesar de que tiene la misma complejidad n^2 este sabe cuando detenerse y para el mejor caso es el más eficaz, también es el que más me confunde ya que usa cosas como el key para almacenar el numero que se quiere intercambiar, el de selección lo pondría en ultimo lugar, ya que todo lo que hace lo hace lento, no importa que ya le des la lista ordenada este comparara cada elemento siempre, puede que sea mejor que el bubble sort en una lista desordenada pero el de selección siempre tendrá una complejidad n^2 porque siempre hará las mismas comparaciones, para mi el mas eficaz es el de inserción porque no necesariamente tiene que recorrer todo el arreglo, solo compara el key con lo demás y una vez ordenado algo no lo vuelve a utilizar, cuando un arreglo esta en el mejor caso el de inserción es la mejor opción y cuando no también, pero para mi es el más complejo de entender de todos.