

Actividad 12 Estructura de datos

Mauricio Estrada de la Garza

AL02976904

Igor Sung Min Kim Juliao

AL02829189

Juan Pablo Hernandez Parra

AL02887299

Reporte:

El Algoritmo empieza con el main donde se crea un scanner para poder interactuar con la máquina, una variable booleana llamada ciclo que está inicializada como true, una variable int llamada conjunto que es donde se guarda el tamaño del arreglo a utilizar. Luego tiene un menú que utiliza un ciclo while y un switch para elegir la opción deseada, en el caso 1 ejecuta la función burbuja(), caso 2 ejecuta la función seleccion(), caso 3 ejecuta la función insercion(), caso 4 ejecuta la función shell(), y caso 5 cambia la variable booleana ciclo a false para así salir(terminar el programa).

El método conjuntos() utiliza un scanner.nextInt() para elegir el número de elementos del arreglo, si va a ser un test usando 10, 100 o 1000 elementos.

Las siguientes funciones tienen todas un switch para elegir si el caso es para el arreglo de 10, 100 o 1000 elementos, todas tienen un contador de tiempo y todas imprimen el resultado y el tiempo que les tardo en llegar a el:

shell()

Manda a llamar el método shellSort() con el arreglo como parámetro.

insercion()

Manda a llamar el método insertionSort() con el arreglo como parámetro.

seleccion()

Manda a llamar el método selectionSort() con el arreglo como parámetro.

burbuja()

Manda a llamar el método bubbleSort() con el arreglo como parámetro.

generarArreglo(int size) usa la cantidad seleccionada previamente en el menu y genera con un random numeros aleatorios del 0 al 999 para todos los elementos del arreglo.

public static void bubbleSort(int[] arr)

Este método implementa el algoritmo de ordenamiento conocido como "Bubble Sort" (ordenamiento burbuja) en Java. Aquí hay una descripción paso a paso de lo que hace:

1. Recibe como parámetro un array de enteros `arr` que se desea ordenar.
2. Declara una variable `n` que almacena la longitud del array.
3. Comienza un bucle externo que itera desde 0 hasta `n - 1`. Este bucle controla la cantidad de pasadas que se realizarán sobre el array.
4. Dentro del bucle externo, hay otro bucle interno que itera desde 0 hasta `n - i - 1`. Este bucle controla la cantidad de comparaciones que se realizarán en cada pasada. La razón por la que

se resta `i`` es porque después de cada pasada, el elemento más grande se coloca en su posición final, por lo que no es necesario compararlo nuevamente en las siguientes pasadas.

5. Dentro del bucle interno, se compara cada elemento `arr[j]` con el siguiente elemento `arr[j + 1]`.

6. Si `arr[j]` es mayor que `arr[j + 1]`, se intercambian los elementos. Esto significa que los elementos más grandes "burbujean" hacia el final del array.

7. Después de que el bucle interno completa todas las comparaciones para una pasada, el elemento más grande está en su posición final.

8. El bucle externo continúa, repitiendo este proceso hasta que no sea necesario realizar más intercambios.

9. Una vez que el bucle externo termina de iterar, el array estará ordenado de manera ascendente, con los elementos más pequeños al principio y los más grandes al final.

`selectionSort(int[] arr)`

Este método implementa el algoritmo de ordenamiento conocido como "Selection Sort" (ordenamiento por selección) en Java. Aquí tienes una descripción paso a paso de lo que hace:

1. Recibe como parámetro un array de enteros `arr` que se desea ordenar.

2. Declara una variable `n` que almacena la longitud del array.

3. Comienza un bucle externo que itera desde 0 hasta `n - 1`. Este bucle controla la cantidad de pasadas que se realizarán sobre el array.

4. En cada pasada del bucle externo, se asume que el elemento en la posición actual `i` es el mínimo.

5. Dentro del bucle externo, hay otro bucle interno que itera desde `i + 1` hasta el final del array (`n - 1`). Este bucle busca el índice del elemento mínimo en el subarray que comienza desde `i + 1`.

6. Dentro del bucle interno, si se encuentra un elemento que es menor que el elemento en `minIndex`, se actualiza `minIndex` con el índice de ese elemento.

7. Después de completar el bucle interno, `minIndex` contendrá el índice del elemento mínimo en el subarray que comienza desde `i`.

8. Fuera del bucle interno, se intercambian los elementos `arr[i]` y `arr[minIndex]`. Esto coloca el elemento mínimo en la posición `i` del array.

9. El bucle externo continúa, repitiendo este proceso hasta que todos los elementos hayan sido ordenados.

10. Una vez que el bucle externo termina de iterar, el array estará ordenado de manera ascendente, con los elementos más pequeños al principio y los más grandes al final.

`insertionSort(int[] arr)`

Este método implementa el algoritmo de ordenamiento conocido como "Insertion Sort" (ordenamiento por inserción) en Java. Aquí tienes una descripción paso a paso de lo que hace:

1. Recibe como parámetro un array de enteros `arr` que se desea ordenar.

2. Declara una variable `n` que almacena la longitud del array.

3. Comienza un bucle externo que itera desde el segundo elemento (índice 1) hasta el final del array.
4. En cada iteración del bucle externo, se selecciona el elemento en la posición `i` y se almacena en la variable `key`.
5. Se inicializa una variable `j` con el valor `i - 1`. Esta variable se utiliza para comparar y mover elementos hacia la derecha hasta encontrar la posición correcta para insertar el elemento `key`.
6. Se inicia un bucle `while` que se ejecuta mientras `j` es mayor o igual a 0 y el elemento en la posición `j` es mayor que `key`.
7. Dentro del bucle `while`, se mueve cada elemento a la derecha (`arr[j]` se mueve a `arr[j + 1]`) para dejar espacio para insertar `key` en la posición correcta.
8. Se decrementa `j` en cada iteración para continuar buscando la posición correcta para `key`.
9. Una vez que se encuentra la posición correcta para `key` (cuando `arr[j] <= key`), se inserta `key` en la posición `j + 1`.
10. El bucle externo continúa, repitiendo este proceso hasta que todos los elementos hayan sido insertados en su posición correcta.
11. Una vez que el bucle externo termina de iterar, el array estará ordenado de manera ascendente, con los elementos más pequeños al principio y los más grandes al final.

```
public static void shellSort(int[] arr)
```

Este método implementa el algoritmo de ordenamiento conocido como "Shell Sort" en Java. A continuación, te proporciono una descripción paso a paso de lo que hace:

1. Recibe como parámetro un array de enteros `arr` que se desea ordenar.
2. Declara una variable `n` que almacena la longitud del array.
3. Inicia un bucle externo que controla los intervalos de comparación. Comienza con el intervalo igual a la mitad del tamaño del array ($n / 2$), y en cada iteración del bucle, divide el intervalo a la mitad ($\text{intervalo} /= 2$) hasta que el intervalo sea 0.
4. Dentro del bucle externo, hay otro bucle que itera sobre los elementos del array comenzando desde el índice igual al valor del intervalo ($i = \text{intervalo}$) hasta el final del array.
5. En cada iteración del bucle interno, se guarda el valor del elemento actual en la variable `temp`.
6. Se inicializa una variable `j` con el valor `i`.
7. Se inicia un bucle interno dentro del bucle anterior, que se ejecuta mientras `j` sea mayor o igual al intervalo y el elemento en la posición `j - intervalo` sea mayor que `temp`.
8. Dentro del bucle interno, se mueven los elementos hacia la derecha ($\text{arr}[j] = \text{arr}[j - \text{intervalo}]$) para dejar espacio para insertar el valor `temp` en la posición correcta.
9. Se decrementa `j` en cada iteración del bucle interno para continuar buscando la posición correcta para `temp`.
10. Una vez que se encuentra la posición correcta para `temp`, se inserta `temp` en la posición `j`.
11. El bucle interno continúa, repitiendo este proceso hasta que se cumpla la condición de detención.
12. Después de completar el bucle interno, se mueve al siguiente intervalo en el bucle externo y se repite el proceso.

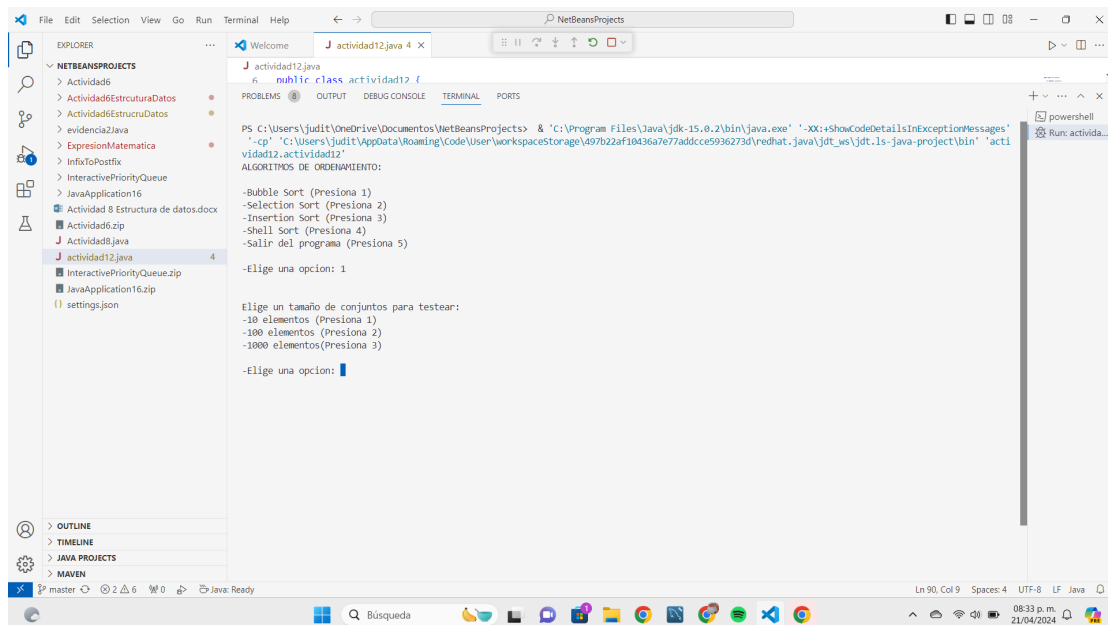
13. Una vez que se han procesado todos los intervalos y se ha completado el bucle externo, el array está ordenado de manera ascendente.

14. El algoritmo Shell Sort es una variante del método de inserción que mejora su rendimiento al hacer múltiples comparaciones y movimientos en intervalos más grandes, lo que ayuda a reducir la cantidad de movimientos necesarios para ordenar el array.

y finalmente

```
public static void imprimirArreglo(int[] arr) imprime el arreglo usando la función toString
```

Pruebas:



```
PS C:\Users\judit\OneDrive\Documents\NetBeansProjects> & 'C:\Program Files\Java\jdk-15.0.2\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages'
'-cp' 'C:\Users\judit\AppData\Roaming\Code\User\workspaceStorage\497b22af10436a7e77adcc5936273d\redhat-java\jdt_ws\jdt.ls-java-project\bin' 'acti
vidad12.actividad12'

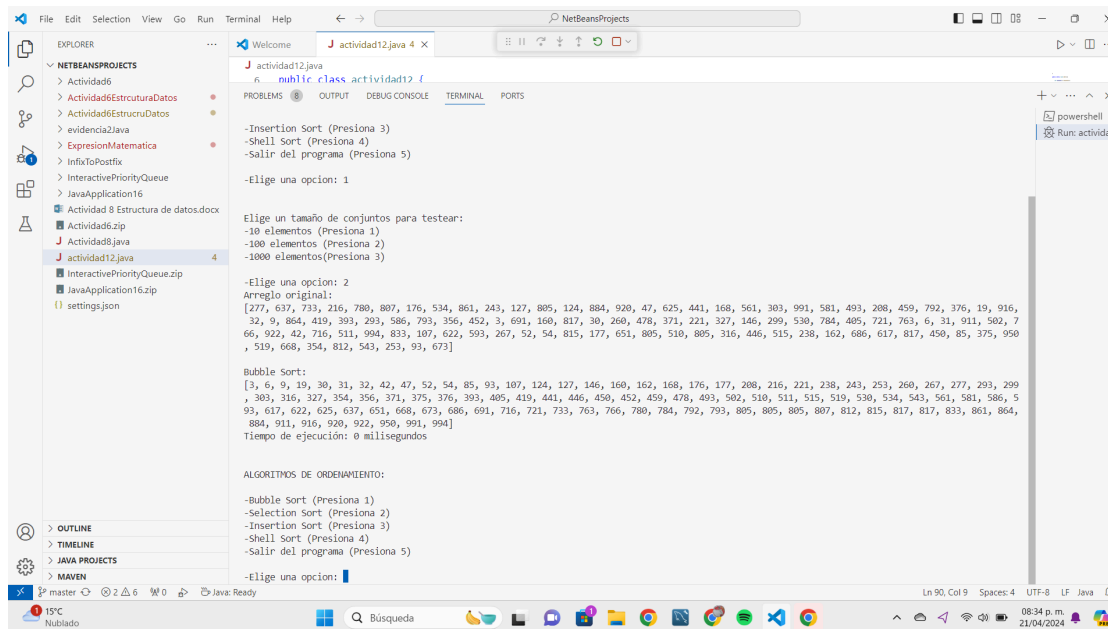
ALGORITMOS DE ORDENAMIENTO:

-Bubble Sort (Presiona 1)
-Selection Sort (Presiona 2)
-Insertion Sort (Presiona 3)
-Shell Sort (Presiona 4)
-Salir del programa (Presiona 5)

-Elige una opción: 1

Elige un tamaño de conjuntos para testear:
-10 elementos (Presiona 1)
-100 elementos (Presiona 2)
-1000 elementos (Presiona 3)

-Elige una opción: 1
```



```
-Insertion Sort (Presiona 3)
-Shell Sort (Presiona 4)
-Salir del programa (Presiona 5)

-Elige una opción: 1

Elige un tamaño de conjuntos para testear:
-10 elementos (Presiona 1)
-100 elementos (Presiona 2)
-1000 elementos (Presiona 3)

-Elige una opción: 2

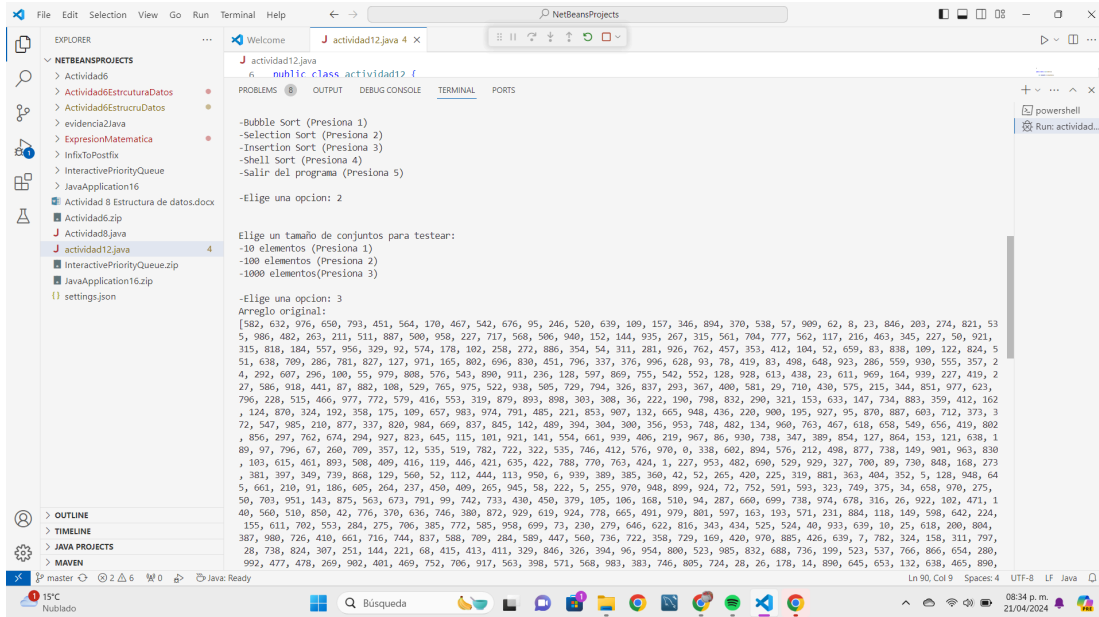
Arreglo original:
[277, 637, 733, 216, 780, 807, 176, 534, 861, 243, 127, 805, 124, 884, 920, 47, 625, 441, 168, 561, 303, 991, 581, 493, 208, 459, 792, 376, 19, 916,
32, 9, 864, 419, 393, 293, 586, 793, 356, 452, 3, 691, 160, 817, 30, 260, 478, 371, 221, 327, 146, 299, 530, 784, 405, 721, 763, 6, 31, 911, 502, 7
66, 922, 42, 716, 511, 994, 833, 107, 622, 593, 267, 52, 54, 815, 177, 651, 805, 510, 805, 316, 446, 515, 238, 162, 686, 617, 817, 450, 85, 375, 950
, 519, 668, 354, 812, 543, 253, 93, 673]

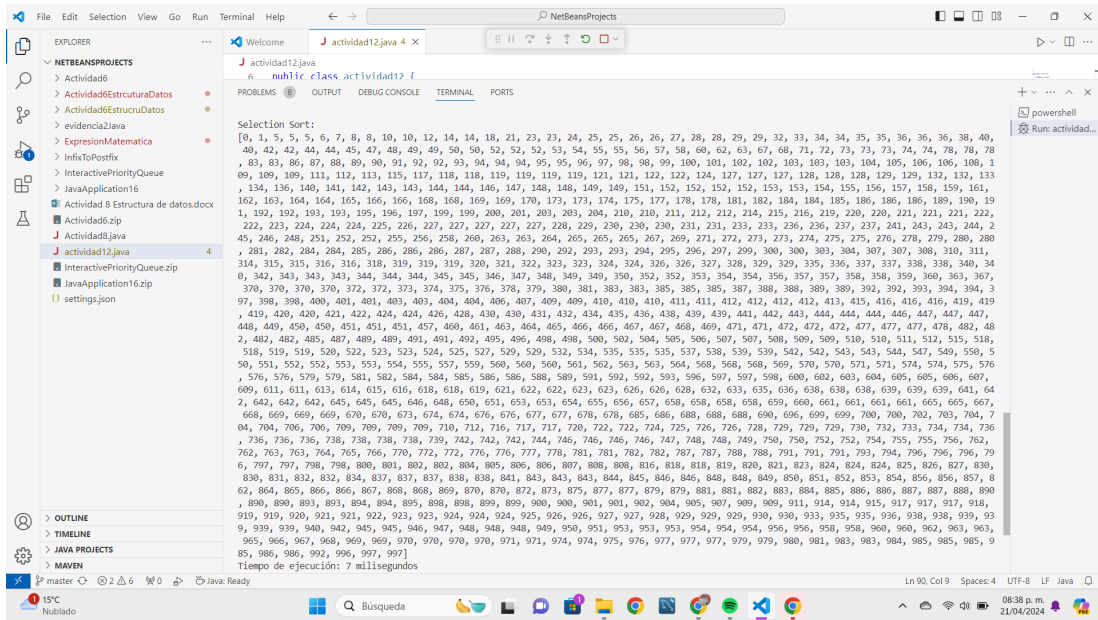
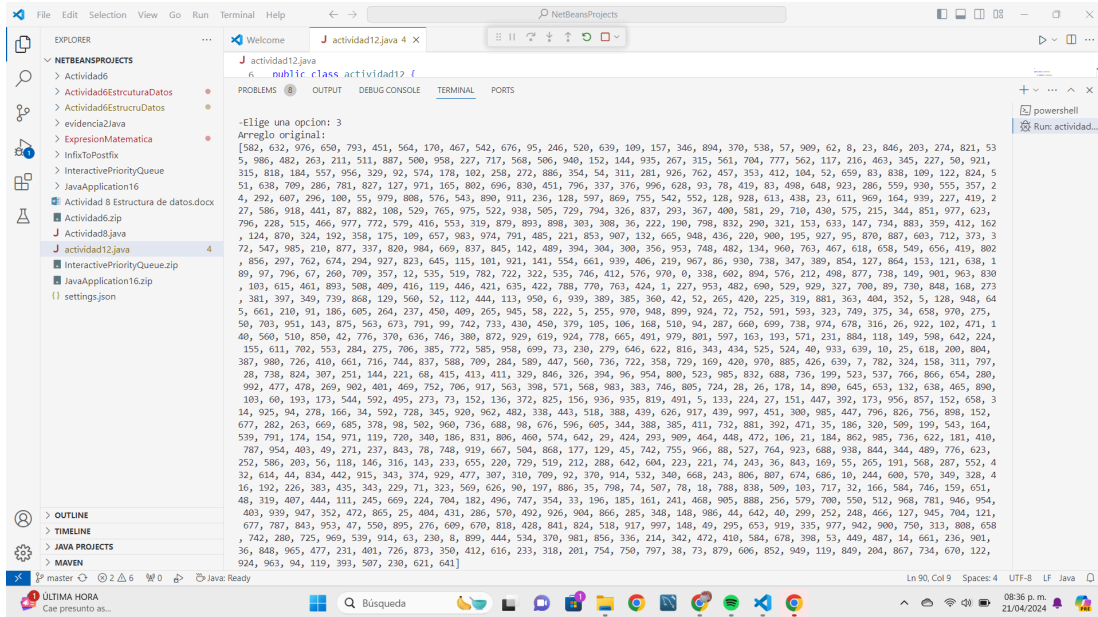
Bubble Sort:
[3, 6, 9, 19, 30, 31, 32, 42, 47, 52, 54, 85, 93, 107, 124, 127, 146, 160, 162, 160, 176, 177, 208, 216, 221, 238, 243, 253, 260, 267, 277, 293, 299
, 303, 316, 327, 354, 356, 371, 375, 376, 393, 405, 419, 441, 446, 450, 452, 459, 478, 493, 502, 510, 511, 515, 519, 530, 534, 543, 561, 581, 586, 5
93, 617, 622, 625, 637, 651, 668, 673, 686, 691, 716, 721, 733, 763, 766, 780, 784, 792, 793, 805, 805, 805, 807, 812, 815, 817, 817, 833, 861, 864,
884, 911, 916, 920, 922, 950, 991, 994]
Tiempo de ejecución: 0 milisegundos

ALGORITMOS DE ORDENAMIENTO:

-Bubble Sort (Presiona 1)
-Selection Sort (Presiona 2)
-Insertion Sort (Presiona 3)
-Shell Sort (Presiona 4)
-Salir del programa (Presiona 5)

-Elige una opción: 1
```





Reflexión:

Mau: La verdad no estubo tan complicado de hacer los algoritmos, al fin de cuentas sirven para lo mismo pero hacen los procesos de diferentes maneras y unos algoritmos procesan la información más rápido que otros, lo único que pudimos mejorar fue hacer un poquito de código más limpio para ahorrar más tiempo pero de ahí en fuera funciona bien el programa. Además últimamente desarrollo este tipo de algoritmos y meterle lógica que ha ayudado a programar de una mejor forma mis proyectos personales para mi portafolio como desarrollador web y también me ha ayudado analizar bugs y problemas de una manera más sencilla.

Juan Pablo: el proceso no fue tan complejo sin embargo tuvimos que realizar varias pruebas y correcciones ya que en varias ocasiones se mostraban problemas al momento de poner a correr el código pero con los cambios menores realizados al algoritmo se logró funcionar sin ninguna complicación.

Igor: A mi me pareció algo nuevo el tener que tomar el tiempo de cada una de las ejecuciones pero fuera de eso no le vi mucha diferencia a este trabajo y los que hicimos anteriormente, si algo se me hizo un poco curioso era la forma de organizar los arreglos pero al fin y al cabo no tenía mucha ciencia lo único que tenías que hacer era usar bien los ciclos for y las condicionales if. De cierta forma siento que estos ciclos y la forma de usarlos son parecidos a cómo manejamos los bucles y condiciones de salida de la recursión.