

**UNIVERSIDAD DE SONORA**

**Unidad Regional Centro**

**División de Ingeniería**

# **Ingeniería en Sistemas de Información**

**Fundamentos de Computación 3**

**Maestro: IVAN DOSTOYEWSKI MEZA IBARRA**

**Grupo 1 – Hora 5 PM – 6 PM- Edif. 5G – L205**

**Nombre del Alumno:**

* **Vallejo Leyva Marcos**

**Actividad 1: Algoritmos de Ordenamiento**

# **Reporte de Código Java - Algoritmos de Ordenamiento**

**Descripción General:**

El código proporciona implementaciones de varios algoritmos de ordenamiento en Java. Los algoritmos incluidos son: selección ascendente y descendente, inserción ascendente y descendente, shell ascendente y descendente, y el algoritmo de ordenamiento rápido (quicksort). Además, el código incluye un método para imprimir un arreglo y un método principal que genera un arreglo desordenado, realiza copias del mismo y mide el tiempo de ejecución para cada algoritmo de ordenamiento.

**Algoritmos de Ordenamiento Implementados:**

**Clase Ordenamiento:**

La clase **Ordenamiento** contiene métodos estáticos para implementar varios algoritmos de ordenamiento y un método principal (**main**) que genera un arreglo aleatorio, lo ordena utilizando diferentes algoritmos y mide el tiempo de ejecución.

Texto

Descripción generada automáticamenteCaptura de pantalla de computadora

Descripción generada automáticamente

**Métodos de Ordenamiento:**

**Método seleccionAscendente:**

Este método implementa el algoritmo de ordenamiento por selección en orden ascendente. Selecciona repetidamente el elemento mínimo y lo intercambia con el elemento en la posición actual.

Texto

Descripción generada automáticamente

**Método seleccionDescendente:**

Similar al método anterior, pero ordena en forma descendente seleccionando repetidamente el elemento máximo.

Texto

Descripción generada automáticamente

**Método insercionAscendente:**

Implementa el algoritmo de ordenamiento por inserción en orden ascendente. Inserta cada elemento en su posición correcta dentro de la parte ya ordenada del arreglo.

Texto

Descripción generada automáticamente

**Método insercionDescendente:**

Similar al método anterior, pero ordena en forma descendente.

Pantalla de computadora con letras

Descripción generada automáticamente

**Método shellAscendente:**

Implementa el algoritmo de ordenamiento Shell en orden ascendente. Utiliza un intervalo de comparación variable para mejorar el rendimiento del algoritmo de inserción.

Texto

Descripción generada automáticamente

**Método shellDescendente:**

Similar al método anterior, pero ordena en forma descendente.

Texto

Descripción generada automáticamente

**Método Quicksort:**

Implementa el algoritmo de ordenamiento Quicksort. Divide el arreglo en subarreglos, ordena cada subarreglo y combina los resultados.

Texto

Descripción generada automáticamente

**Método partition:**

Método privado utilizado por Quicksort para realizar la partición del arreglo.

Texto

Descripción generada automáticamente

**Método imprimirArreglo:**

Este método simplemente imprime un arreglo usando **Arrays.toString()**.

Texto

Descripción generada automáticamente

**Método main:**

El método main organiza y ejecuta todo el proceso de generación, ordenación y medición del tiempo, proporcionando una visión completa del rendimiento de cada algoritmo en el contexto específico de este programa.

1. **Generación del Arreglo Desordenado:**

* Se crea un arreglo llamado **arregloDesordenado** de tamaño 1000.
* Se utiliza la clase **Random** para llenar este arreglo con números aleatorios entre 0 y 999.

Texto

Descripción generada automáticamente

1. **Copia del Arreglo Desordenado para Cada Método:**

* Se crea una copia del arreglo desordenado para cada algoritmo de ordenamiento que se probará. Esto se realiza mediante la función **‘Arrays.copyOf()’**.

Texto

Descripción generada automáticamente

1. **Impresión del Arreglo Desordenado:**

* Se imprime el arreglo original antes de ser ordenado.

Texto

Descripción generada automáticamente

1. **Ordenamiento y Medición del Tiempo:**
   * Se realiza la ordenación ascendente y descendente para cada algoritmo.
   * Se mide el tiempo de ejecución de cada algoritmo utilizando **System.nanoTime()**.
   * Esto se repite para cada algoritmo de ordenamiento, tanto en orden ascendente como en orden descendente.

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

1. **Impresión de Resultados:**
   * Se imprime el arreglo desordenado y los arreglos ordenados para cada algoritmo junto con el tiempo de ejecución.

Texto

Descripción generada automáticamente

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

**Análisis y Observaciones:**

1. El código utiliza la clase **Random** para generar números aleatorios y llenar un arreglo desordenado de tamaño 1000.
2. Se realiza una copia del arreglo desordenado para cada algoritmo de ordenamiento, lo que permite medir el tiempo de ejecución para cada algoritmo individualmente.
3. El tiempo de ejecución se mide en nanosegundos utilizando el método **System.nanoTime()**.

**Conclusiones**

* Los algoritmos de ordenamiento implementados producen resultados correctos en términos de ordenación ascendente y descendente.
* El tiempo de ejecución varía según el algoritmo, siendo Quicksort generalmente más eficiente en términos de tiempo en comparación con los otros algoritmos.
* La medición del tiempo de ejecución proporciona una comparación objetiva del rendimiento de cada algoritmo.

En general, el código proporciona una implementación clara y funcional de varios algoritmos de ordenamiento en Java, y los resultados obtenidos demuestran la eficacia de cada algoritmo en diferentes contextos.