PRÁCTICA 05 – Análisis comparativo y aplicación de algoritmos de ordenamiento en estructuras lineales

Curso: Algoritmos y Estructuras de Datos
Código: SIS210
Ciclo: IV
Docente: Mg. Aldo Hernán Zanabria Gálvez
Duración estimada: 3 horas
Semana: 5 – 2025-I

I. Objetivo general

Aplicar, analizar y comparar algoritmos de ordenamiento clásicos (Bubble Sort, Selection Sort, Insertion Sort) y eficientes (Merge Sort, Quick Sort), implementados sobre estructuras de datos lineales, evaluando su rendimiento, complejidad algorítmica y adecuación según el tipo de entrada y lenguaje (C++ y Python).

II. Objetivos específicos

- Implementar algoritmos de ordenamiento sobre vectores y listas.
- Comparar el rendimiento de algoritmos cuadráticos vs. algoritmos eficientes.
- Visualizar y analizar el comportamiento de los algoritmos bajo entradas ordenadas, inversas y aleatorias.
- Evaluar los tiempos de ejecución y el número de operaciones realizadas.
- Reflexionar sobre el impacto de la estructura de datos utilizada en la eficiencia del algoritmo.

III. Marco teórico

1. Definición Formal de Ordenamiento

El ordenamiento es una operación fundamental en ciencias de la computación, que consiste en reorganizar los elementos de un conjunto $A = \{a_1, a_2, ..., a_n\}$ en una secuencia ordenada creciente o decreciente respecto de una relación de orden total (por ejemplo, ≤).

Formalmente, un algoritmo de ordenamiento transforma una secuencia de entrada no ordenada:

$A = \{a_1, a_2, ..., a_n\}$

en una secuencia ordenada:

$A' = \{a_1', a_2', ..., a_n'\}$ tal que $a_1' \leq a_2' \leq ... \leq a_n'$

donde A' es una permutación de los elementos originales de A.

2. Clasificación de Algoritmos de Ordenamiento

Los algoritmos de ordenamiento se pueden clasificar en dos grandes grupos:

- **Algoritmos de Comparación**: Determinan el orden de los elementos comparándolos entre sí. Ejemplos incluyen Bubble Sort, Selection Sort, Insertion Sort, Merge Sort y Quick Sort.

- **Algoritmos No Basados en Comparación**: Utilizan propiedades específicas de los datos para ordenar sin comparaciones directas, como Counting Sort y Radix Sort.

3. Complejidad Computacional

La eficiencia de un algoritmo de ordenamiento se evalúa principalmente por su complejidad temporal y espacial, expresadas en notación asintótica. Las más comunes son:

- Crecimiento lineal
- Crecimiento logarítmico lineal
- Crecimiento cuadrático
- Constante

En el caso de ordenamiento, el objetivo es minimizar el número de comparaciones e intercambios realizados.

4. Propiedades Analíticas de los Algoritmos de Ordenamiento

- **Estabilidad**: Un algoritmo es estable si preserva el orden relativo de los elementos con claves iguales. Merge Sort es estable, mientras que Quick Sort no lo es en su implementación básica.
- **In-Place**: Un algoritmo es in-place si requiere una cantidad constante de espacio adicional. Quick Sort es in-place, mientras que Merge Sort no lo es en su forma estándar.
- **Adaptabilidad**: Algunos algoritmos, como Insertion Sort, son eficientes para listas que ya están parcialmente ordenadas.

5. Aplicación en Estructuras de Datos

Los algoritmos de ordenamiento pueden aplicarse sobre diferentes estructuras de datos, como:

- **Arreglos Estáticos**: Permiten acceso aleatorio, ideales para algoritmos como Quick Sort y Merge Sort.
- **Listas Enlazadas**: Requieren ordenamiento mediante algoritmos adaptados, como Merge Sort no recursivo, ya que el acceso secuencial limita el rendimiento de métodos basados en índices.

6. Evaluación Empírica

Para evaluar el rendimiento real, se implementan los algoritmos en distintos lenguajes y se miden los siguientes indicadores:

- Tiempo de ejecución (ttt): medido en milisegundos.
- Número de comparaciones: operaciones de comparación lógica.
- Número de intercambios: reordenamientos efectivos.

Las pruebas se realizan sobre tres tipos de entrada:

- **Aleatoria**: sin patrón previo
- **Ordenada**: en orden creciente
- **Inversa**: en orden decreciente

Estos escenarios permiten observar la sensibilidad del algoritmo a diferentes configuraciones.

IV. Actividades

Parte A – Implementación en C++

1. Implementar Bubble Sort, Insertion Sort, Merge Sort y Quick Sort.
2. Aplicarlos a un arreglo de 1000 elementos aleatorios, ordenados e inversos.
3. Medir el tiempo de ejecución utilizando chrono.

Parte B – Implementación en Python

1. Implementar los mismos algoritmos utilizando listas.
2. Usar time para medir rendimiento.
3. Aplicar a entradas de las mismas características y comparar resultados.

V. Comparación técnica esperada

Algoritmo	**Tipo de entrada**	**Tiempo C++ (ms)**	**Tiempo Python (ms)**	**Comparaciones**	**Intercambios**
Bubble Sort	Aleatoria				
Merge Sort	Ordenada				
Quick Sort	Inversa				

VI. Informe a entregar

1. Portada institucional
2. Objetivos de la práctica
3. Implementación en ambos lenguajes
    - Códigos funcionales y comentados
    - Capturas de ejecución
    - Medición de tiempos (con tablas y gráficos si es posible)
4. Comparación entre algoritmos
5. Análisis de eficiencia y adecuación de uso
6. Conclusiones personales (mínimo tres)
7. Enlace a OnlineGDB / Google Colab o GitHub (si aplica)

VII. Rúbrica de evaluación (Total: 20 puntos)

Criterio de evaluación	**Puntaje**
Implementación completa y funcional de algoritmos	6 pts
Medición y análisis de tiempos de ejecución	2 pts
Comparación crítica y técnica entre algoritmos	6 pts
Presentación formal y estructura del informe	4 pt
Conclusiones claras y reflexivas	2 pt

Referencias:

1. Brassard, G., & Bratley, P. (1997). *Fundamentos de algoritmia*. Prentice Hall.
2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms* (4th ed.). MIT Press.
3. Goodrich, M. T., & Tamassia, R. (2014). *Data structures and algorithms in Python*. Wiley.
4. Weiss, M. A. (2017). *Data structures and algorithm analysis in C++* (4th ed.). Pearson.
5. Knuth, D. E. (1998). *The art of computer programming, Volume 3: Sorting and searching* (2nd ed.). Addison-Wesley.
6. Skiena, S. S. (2008). *The algorithm design manual* (2nd ed.). Springer.
7. Karumanchi, N. (2011). *Data structures and algorithms made easy*. CareerMonk Publications.
8. Hernández Yáñez, L. (2013). *Fundamentos de la programación: Algoritmos de ordenación*. Universidad Complutense de Madrid. Recuperado de https://www.fdi.ucm.es/profesor/luis/fp/FP07.pdf
9. Universidad Don Bosco. (2019). *Métodos de ordenamiento. Parte 1*. Recuperado de https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-iv/2019/ii/guia-3.pdf
10. Wikipedia. (2023). *Algoritmo de ordenamiento*. Recuperado de https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento