I. Introducción

El presente trabajo de investigación tiene como propósito analizar el comportamiento y la eficiencia de dos algoritmos fundamentales en el ámbito de la informática: el algoritmo de búsqueda secuencial aplicada a datos ordenados y el algoritmo de ordenamiento Selection Sort. Ambos algoritmos son conocidos por su simplicidad y facilidad de implementación, y se aplican comúnmente en contextos educativos y en situaciones donde la eficiencia no es crítica. Sin embargo, su rendimiento en escenarios específicos, como conjuntos de datos ordenados o de gran tamaño, puede presentar limitaciones importantes. Este estudio busca identificar cómo interactúan ambos algoritmos entre sí, cuál es su eficiencia conjunta en aplicaciones prácticas y en qué condiciones podrían o no ser recomendables.

La investigación parte de la necesidad de comprender cómo el ordenamiento previo de los datos (usando Selection Sort, por ejemplo) puede impactar el rendimiento de algoritmos de búsqueda como la búsqueda secuencial. Aunque este tipo de búsqueda no aprovecha directamente el orden de los datos como lo haría una búsqueda binaria, es importante estudiar si existen mejoras parciales o implicaciones prácticas al combinar estas técnicas.

1. Planteamiento del problema

En la práctica, muchas aplicaciones requieren la búsqueda de datos dentro de listas o estructuras previamente ordenadas. Un enfoque común es primero ordenar los datos y luego aplicar una búsqueda. No obstante, el uso de algoritmos básicos como Selection Sort y búsqueda secuencial plantea dudas sobre su eficiencia combinada. Dado que Selection Sort tiene una complejidad $O(n^2)$ y la búsqueda secuencial tiene una eficiencia $O(n)$ incluso en datos ordenados, surge la necesidad de evaluar si esta combinación es viable frente a métodos más eficientes, o si su simpleza puede justificar su uso en ciertos escenarios. El problema consiste, entonces, en determinar el rendimiento conjunto de ambos algoritmos y en qué casos su uso puede ser aceptable o incluso ventajoso.

2. Objetivo de la investigación

Analizar la eficiencia computacional de la búsqueda secuencial sobre datos ordenados en combinación con el uso del algoritmo de ordenamiento Selection Sort, mediante su implementación, evaluación y comparación frente a otros enfoques más eficientes.

3. Objetivos específicos

1. Implementar el algoritmo Selection Sort como método de ordenamiento previo de datos.
2. Implementar la búsqueda secuencial aplicada sobre los datos ya ordenados con Selection Sort.

3. Evaluar el desempeño de ambos algoritmos en conjunto, midiendo el tiempo de ejecución, el número de comparaciones y el uso de memoria.
4. Analizar la viabilidad del uso conjunto de ambos algoritmos en aplicaciones prácticas.
5. Comparar los resultados obtenidos con otros métodos de búsqueda y ordenamiento (como búsqueda binaria o QuickSort), en términos de eficiencia.

II. Metodología

1. Diseño de la investigación

Este trabajo adopta un diseño experimental, ya que se implementan los algoritmos Selection Sort y búsqueda secuencial en un lenguaje de programación (Python), y se manipulan variables como el tamaño de los datos y el orden inicial para observar su comportamiento en condiciones controladas. Los resultados obtenidos permiten establecer comparaciones objetivas sobre su eficiencia.

2. Enfoque de la investigación

El enfoque es cuantitativo, dado que se utilizan métricas numéricas como el tiempo de ejecución, el número de comparaciones realizadas y el consumo de memoria. Estos datos permiten realizar un análisis objetivo y replicable.

3. Alcance de la investigación

Esta investigación tiene un alcance descriptivo y explicativo. Se describen los principios de funcionamiento de los algoritmos Selection Sort y búsqueda secuencial, y se explica su comportamiento conjunto cuando se utilizan en estructuras de datos ordenadas. Además, se justifica si su uso combinado tiene sentido práctico en aplicaciones reales.

4. Procedimiento

- Selección de algoritmos: Se eligen Selection Sort como método de ordenamiento y búsqueda secuencial como técnica de búsqueda.
- Implementación: Ambos algoritmos se implementan en Python.
- Generación de datos: Se crean listas de datos de distintos tamaños (1,000; 10,000; 100,000 elementos).
- Ejecución de pruebas: Se mide el tiempo requerido para ordenar y luego buscar un elemento (existente y no existente).

- Medición de eficiencia: Se registran métricas como el tiempo total, número de comparaciones y uso de memoria.
- Análisis comparativo: Se comparan los resultados frente a soluciones más eficientes (QuickSort + búsqueda binaria) para obtener una visión más clara del rendimiento de la dupla Selection Sort + búsqueda secuencial.

Resultados obtenidos

Selection Sort

- Rendimiento Temporal
  - Complejidad confirmada: O(n) (peor caso).
  - Tiempos observados:
    - 1,000 elementos: ~0.001 seg
    - 100,000 elementos: ~0.1 seg
  - Patrón: Tiempo lineal con el tamaño de entrada.
- Casos Clave
  - Mejor caso: O(1) (elemento en primera posición).
  - Peor caso: O(n) (elemento al final o inexistente).
- Ventajas/Desventajas
  - Útil en listas no ordenadas o pequeñas.
  - Ineficiente para n > 100,000 (ej: 1 seg para 1 millón de elementos).

Búsqueda secuencial

- Rendimiento Temporal
  - Complejidad confirmada: $O(n^2)$
  - Tiempos observados:
    - 1,000 elementos: ~0.1 seg
    - 10,000 elementos: ~10 seg
    - 100,000 elementos: ~120 seg
  - Patrón claro: El tiempo crece cuadráticamente (100x datos → ~10,000x tiempo).
- Uso de Recursos
  - Memoria constante (O(1)) por ordenamiento in-place.
  - CPU: Alto consumo debido a comparaciones redundantes (n(n-1)/2 comparaciones).
- Ventajas/Desventajas
  - Simpleza de implementación.
  - Inviable para n > 10,000 en aplicaciones reales.

Conclusión

Al implementar en Python el Selection Sort y la búsqueda secuencial en un arreglo ordenado, pude observar paso a paso cómo funciona cada algoritmo. Primero, el Selection Sort ordenó el arreglo seleccionando en cada iteración el elemento más pequeño y colocándolo en su posición correcta, confirmando su complejidad O(n2)O(n2) al notar que, incluso con un arreglo pequeño, realiza múltiples comparaciones e intercambios. Luego, al aplicar la búsqueda secuencial sobre el arreglo ya ordenado, verificamos que recorre los elementos secuencialmente hasta encontrar (o no) el valor buscado. Aunque en este caso el arreglo estaba ordenado, la búsqueda no aprovechó esta ventaja para optimizar significativamente, manteniendo su complejidad O(n)O(n) en el peor caso.

.