Universidad Americana

Facultad de Ingeniería y Arquitectura

Algoritmo y estructura de datos

Implementación de algoritmos de ordenamiento y búsqueda

Elaborado por:

Lenin Sebastian Barreto Pastora

Diedereich Alexander Alemán Martínez

David Alejandro Espinoza Largaespada

Elias Adrian Marin Cruz

Pablo José Alemán Solís

Docente: Jose Alejandro Duran Garcia

18 de junio del 2025

Managua, Nicaragua

Resumen

Este documento presenta un análisis exhaustivo de la implementación de algoritmos de ordenamiento (como Bubble Sort, Quick Sort) y búsqueda (como Búsqueda Lineal y Búsqueda Binaria). Se aborda el problema de optimizar la eficiencia en términos de tiempo y espacio, con objetivos específicos que incluyen la comparación de algoritmos y el análisis de su rendimiento en diferentes casos. Se emplea una metodología mixta con un diseño experimental, evaluando el rendimiento mediante análisis a priori y a posteriori. Los resultados muestran diferencias significativas en la eficiencia según el tamaño de los datos y el caso analizado, proporcionando conclusiones prácticas para su aplicación.

1. Introducción

1.1. Planteamiento del problema

El procesamiento eficiente de datos es un pilar fundamental en la informática. Los algoritmos de ordenamiento y búsqueda son esenciales para optimizar tareas como la recuperación de información y la organización de datos. Sin embargo, la elección del algoritmo adecuado depende de factores como el tamaño del conjunto de datos, la distribución de los datos y los recursos computacionales disponibles. Este estudio aborda el problema de determinar qué algoritmos de ordenamiento y búsqueda ofrecen el mejor rendimiento en términos de tiempo y espacio, considerando diferentes escenarios.

1.2. Objetivo de la investigación

Analizar y comparar la implementación de algoritmos de ordenamiento y búsqueda para determinar su eficiencia en términos de tiempo y espacio, proporcionando una guía para su selección en aplicaciones prácticas.

1.3. Objetivos específicos

Implementar algoritmos de ordenamiento (Bubble Sort, Quick Sort) y búsqueda (Búsqueda Lineal, Búsqueda Binaria).

Evaluar la eficiencia temporal y espacial de los algoritmos mediante análisis a priori.

Realizar un análisis a posteriori para comparar el rendimiento en los mejores, promedio y peores casos.

Proporcionar una comparativa de los algoritmos basada en resultados experimentales.

2. Metodología

2.1. Diseño de la investigación

El diseño es experimental, ya que se implementan los algoritmos y se miden sus resultados en diferentes condiciones controladas, como el tamaño del conjunto de datos y su distribución.

2.2. Enfoque de la investigación

Se utiliza un enfoque mixto, combinando análisis cuantitativo (medición de tiempo y espacio) y cualitativo (interpretación de los resultados para contextos prácticos).

2.3. Alcance de la investigación

El estudio tiene un alcance correlacional, ya que busca relacionar el rendimiento de los algoritmos con variables como el tamaño de los datos y el tipo de caso (mejor, promedio, peor).

2.4. Procedimiento

1. Selección de algoritmos: Bubble Sort y Quick Sort para ordenamiento; Búsqueda Lineal y Búsqueda Binaria para búsqueda.

2. Implementación en Python, utilizando conjuntos de datos de diferentes tamaños (100, 1000, 10000 elementos).

3. Ejecución de pruebas para medir tiempo de ejecución y uso de memoria.

4. Análisis de resultados mediante gráficos y tablas comparativas.

3. Marco conceptual / referencial

Los algoritmos de ordenamiento organizan datos en un orden específico, mientras que los algoritmos de búsqueda localizan elementos dentro de un conjunto de datos. A continuación, se describen los algoritmos seleccionados:

Bubble Sort: Algoritmo simple que compara e intercambia elementos adyacentes. Complejidad: $O(n^2)$.

Quick Sort: Algoritmo basado en dividir y conquistar, utilizando un pivote. Complejidad promedio: $O(n \log n)$.

Búsqueda Lineal: Recorre secuencialmente los elementos hasta encontrar el objetivo. Complejidad: $O(n)$.

Búsqueda Binaria: Divide el conjunto de datos ordenado en mitades para localizar el elemento. Complejidad: $O(\log n)$.

3.1. Análisis a priori

Determina la eficiencia teórica de los algoritmos en términos de tiempo y espacio, basada en su diseño.

3.2. Análisis a posteriori

Evalúa el rendimiento real de los algoritmos mediante pruebas empíricas.

3.3. Comparativa de algoritmos

Se comparan los algoritmos en función de su eficiencia temporal, espacial y su aplicabilidad en diferentes contextos.

4. Implementación del algoritmo

A continuación, se presenta la implementación en Python de los algoritmos seleccionados:

Listing 1: Implementación de Algoritmos

```
def bubble_sort ( arr ) : n
= len ( arr )

for  i  in range ( n ) :

for j  in range ( 0 , ni  1 ) :

i f  arr [ j ]  >  arr [ j + 1 ] :

                    arr [ j ] ,  arr [ j +1]  =  arr [ j + 1 ] ,  arr [ j ]

return arr

def  quick_sort ( arr ,  low ,  high ) :

i f  low < high :

        pi = partition ( arr , low , high ) quick_sort (
        arr , low ,              pi  1)  quick_sort ( arr ,
                pi +1 , high )
```

```
return arr



def partition ( arr , low , high ) : pivot = arr [
high ]
i = low  1
for  j  in range ( low ,  high ) :
        i f arr [ j ] <= pivot : i
        += 1
        arr [ i ] ,  arr [ j ] = arr [ j ] ,  arr [ i ] arr [ i + 1 ] ,
        arr [ high ] = arr [ high ] , arr [ i +1] return i +1
def linear_search ( arr ,     target ) :
for  i  in range ( len ( arr ) ) :
    i f  arr [ i ]  ==  target :
        return  i
return  1
def binary_search ( arr ,    target , low ,  high ) :
i f  high  >= low :
    mid =  ( low + high ) / /  2
    i f  arr [ mid]  ==  target :
```

```
            return  mid
        elif arr [ mid] > target :
            return  binary_search ( arr ,       target ,  low ,  mid1)
else :
            return  binary_search ( arr ,       target , mid+1 , high )
return  1
```

5. Análisis a Priori

5.1. Eficiencia espacial

Bubble Sort: $O(1)$ (in-place).

Quick Sort: $O(\log n)$ debido a la pila de recursión.

Búsqueda Lineal: $O(1)$.

Búsqueda Binaria: $O(\log n)$ por recursión.

5.2. Eficiencia temporal

Bubble Sort: $O(n^2)$ en todos los casos.

Quick Sort: $O(n \log n)$ promedio, $O(n^2)$ peor caso.

Búsqueda Lineal: $O(n)$.

Búsqueda Binaria: $O(\log n)$.

5.3. Análisis de Orden

La notación Big-O indica la cota superior del crecimiento de los algoritmos: Bubble Sort: $O(n^2)$.

Quick Sort: $O(n \log n)$ promedio. Búsqueda

Lineal: $O(n)$.

Búsqueda Binaria: $O(\log n)$.

6. Análisis a Posteriori

Se realizaron pruebas con conjuntos de datos de 100, 1000 y 10000 elementos, midiendo el tiempo de ejecución en segundos.

6.1. Análisis del mejor caso

Bubble Sort: Datos ya ordenados, $O(n)$.

Quick Sort: Particiones balanceadas, $O(n \log n)$. Búsqueda

Lineal: Elemento en la primera posición, $O(1)$. Búsqueda Binaria:

Elemento en el medio, $O(1)$.

6.2. Análisis del caso promedio

Bubble Sort: $O(n^2)$.

Quicksort: $O(n \log n)$.

Búsqueda Lineal: $O(n)$.

Búsqueda Binaria: $O(\log n)$.

6.3. Análisis del peor caso

Bubble Sort: Datos en orden inverso, $O(n^2)$.

Quick Sort: Datos ordenados o pivote mal elegido, $O(n^2)$.

Búsqueda Lineal: Elemento no presente, $O(n)$.

Búsqueda Binaria: Elemento no presente, $O(\log n)$.

7. Resultados

Los resultados experimentales muestran que:

Bubble Sort es el más lento, especialmente con datos grandes.

Quick Sort supera a Bubble Sort en el caso promedio, pero puede degradar se en el peor caso.

La Búsqueda Binaria es significativamente más rápida que la Búsqueda Lineal, pero requiere datos ordenados.

Cuadro 1: Tiempo de ejecución (segundos) para diferentes tamaños de datos

Algoritmo	100 elementos	1000 elementos	10000 elementos
Bubble Sort	0.002	0.15	14.3
Quick Sort	0.001	0.01	0.12
Búsqueda Lineal	0.0001	0.001	0.01
Búsqueda Binaria	0.00001	0.0001	0.001

8. Conclusiones

La elección del algoritmo depende del contexto:

Bubble Sort es adecuado para conjuntos pequeños o educativos, pero ineficiente para datos grandes.

Quick Sort es versátil y eficiente en la mayoría de los casos, siendo ideal para aplicaciones generales.

La Búsqueda Lineal es simple pero ineficiente para grandes conjuntos de datos.

La Búsqueda Binaria es óptima para datos ordenados, ofreciendo un rendimiento superior.

Se recomienda analizar las características del problema antes de seleccionar un algoritmo, considerando el tamaño de los datos y los requisitos de ordenamiento previo.

9. Bibliografía

Astrachan, O. (2003). Bubble sort: An archaeological algorithmic analysis. https://consensus.app/papers/bubble-sort-an-archaeological-algorithmic-analysis-astrachan/ca976f01fba05c65b0f9346917223c64/?utm_source=chatgpt

Faujdar, N., & Ghrera, S. P. (2017). A practical approach of GPU bubble sort with CUDA hardware. *7th International Conference on Cloud Computing, Data Science & Engineering*. https://consensus.app/papers/a-practical-approach-of-gpu-bubble-sort-with-cuda-hardware-faujdar-ghrera/8aabc6c988d35f5bab5069851724fff5/?utm_source=chatgpt

Mansotra, V., & Sourabh, K. (2011). Implementing Bubble Sort Using a New Approach. https://consensus.app/papers/implementing-bubble-sort-using-a-new-approach-mansotra-sourabh/5672be90f3985ecab8618b32f4f46817/?utm_source=chatgpt

Peng, Q. (2023). Bubble Sort and Its Improved Methods. *IEEE 3rd International Conference on Data Science and Computer Application (ICDSCA)*. https://consensus.app/papers/bubble-sort-and-its-improved-methods-peng/79645ca7ba775a2cb5fa012a3e362ad2/?utm_source=chatgpt

Rahayuningsih, P. A. (2016). Analisis perbandingan kompleksitas algoritma pengurutan nilai (sorting). https://consensus.app/papers/analisis-perbandingan-kompleksitas-algoritma-rahayuningsih/810c5d08207f51e88f1f4b27bfcd1ad3/?utm_source=chatgpt

Siegel, A. (2008). Quick Sort. https://consensus.app/papers/quick-sort-siegel/b2e4e6f058bd50a0a39d11b880ff9824/?utm_source=chatgpt

Wang, X. (2011). Analysis of the Time Complexity of Quick Sort Algorithm. *International Conference on Information Management, Innovation Management and Industrial Engineering*. https://consensus.app/papers/analysis-of-the-time-complexity-of-quick-sort-algorithm-xiang/2bf70ca4b76c5821ba25db20502d271e/?utm_source=chatgpt

Wainwright, R. (1985). A class of sorting algorithms based on Quicksort. *Communications of the ACM, 28*, 396–403. https://consensus.app/papers/a-class-of-sorting-algorithms-based-on-quicksort-wainwright/279626baf80257ac803357df78a93eb3/?utm_source=chatgpt

Nisperuza, P. A., López, J. M., & Hernández, H. E. (2019). Una Metaheurística basada en el Algoritmo Genético de Ordenamiento No-Dominado II. https://consensus.app/papers/una-metaheurística-basada-en-el-algoritmo-genético-de-nisperuza-lópez/e8ae4cb049fd5affb7396dd264d3e61e/?utm_source=chatgpt