Departamento de Ciencias de la Computación

Asignatura:

Estructura de datos

Tema:

Algoritmos de ordenación interna y simple, esquema bubble sort paso a paso, comparaciones y movimientos

Asesor:

Mgtr. Washington Eduardo Loza Herrera

Integrantes:

Andrade Julio

Bonilla David

Quiroz Maria

Tiupul Danny

27298

19/11/2025

Resumen

Esta investigación se enfoca en cómo las computadoras ponen en orden los datos que manejan internamente, un proceso clave llamado "ordenación interna". Analizamos qué tan rápidos y eficientes son estos métodos, cuánta memoria usan y si logran mantener el orden de los datos repetidos. En concreto, estudiaremos los algoritmos de ordenación más sencillos y básicos (como el "Bubble Sort", que ordena por intercambio, o el "Insertion Sort", que ordena por inserción), los cuales son fáciles de implementar pero se vuelven muy lentos si la cantidad de datos es grande. En esencia, este trabajo ofrece una mirada clara a los fundamentos y la utilidad de estas técnicas básicas para organizar la información en el mundo de la programación.

Introducción

Con esta investigación buscamos entender cómo se ordenan los datos es clave en la informática, ya que organizar la información de forma eficiente hace que los programas funcionen mucho mejor y que las búsquedas sean más rápidas. Esta investigación se centra en la ordenación interna, que son los métodos que trabajan directamente en la memoria principal de la computadora para ser más veloces. Estudiamos los algoritmos más sencillos (como los de intercambio, inserción y selección) porque, aunque son lentos para grandes cantidades de datos, son perfectos para entender las bases de la organización, como su velocidad y la memoria que utilizan. En resumen, este trabajo explica los conceptos esenciales de la ordenación interna y el papel fundamental de estos algoritmos básicos.

Desarrollo

Fundamentos de la Ordenación Interna

Definición de ordenación interna

La ordenación interna se refiere al proceso de organizar datos directamente en la memoria principal, es decir, todos los elementos a ordenar se encuentran cargados en la RAM del sistema, lo que permite su manipulación sin recurrir a almacenamiento secundario (Wikipedia, s.f.-a).

Cuando el volumen de datos excede la capacidad de la memoria principal, se requiere la ordenación externa, la cual utiliza dispositivos como discos o unidades de almacenamiento masivo (Admas University, s.f.).

Características de los algoritmos de ordenación interna

- Complejidad computacional

Los algoritmos de ordenación interna se analizan mediante su complejidad temporal, considerando los casos mejor, promedio y peor escenario, dependiendo del número de elementos nnn presentes en la lista (Admas University, s.f.).

Los algoritmos simples o “métodos directos” suelen presentar un comportamiento cuadrático, es decir, O(n2)O(n^2)O(n2), lo cual los hace poco eficientes para volúmenes grandes de datos (BUAP, s.f.).

Además, se analiza la complejidad espacial, que indica cuánta memoria adicional necesita el algoritmo aparte del arreglo original (Admas University, s.f.).

- Estabilidad

Un algoritmo de ordenamiento es estable cuando conserva el orden relativo entre elementos con claves duplicadas (GeeksforGeeks, s.f.-a).

Esta propiedad es importante en estructuras donde el orden previo entre elementos equivalentes tiene relevancia, tal como en registros con múltiples criterios de clasificación (UpGrad, s.f.).

Algoritmos como Bubble Sort e Insertion Sort son ejemplos de métodos estables (GeeksforGeeks, s.f.-a).

- Adaptabilidad

Un algoritmo es adaptativo si mejora su rendimiento cuando la lista ya se encuentra parcialmente ordenada (GeeksforGeeks, s.f.-a).

Esto permite reducir comparaciones y movimientos cuando la entrada no está completamente desordenada (Wikipedia, s.f.-b).

- Uso de memoria (in-place)

Muchos algoritmos internos se implementan como in-place, lo que significa que requieren una cantidad mínima de espacio adicional, generalmente constante (AVS College, s.f.).

Esto los vuelve adecuados para entornos con memoria limitada, ya que no necesitan estructuras auxiliares grandes (UpGrad, s.f.).

Clasificación de los algoritmos simples de ordenación interna

Los algoritmos simples se clasifican principalmente en tres familias: intercambio, inserción y selección (Studocu, s.f.).
Entre ellos:

- Intercambio: Bubble Sort.
- Inserción: Insertion Sort (BUAP, s.f.).
- Selección: Selection Sort (Wikipedia, s.f.-c).

Estos métodos comparten su complejidad típica de O(n2)O(n^2)O(n2) (BUAP, s.f.), por lo que resultan adecuados para conjuntos pequeños, pero no eficientes para colecciones de datos extensas.

En contraste, métodos más modernos como Quick Sort poseen una eficiencia más alta, alcanzando O(nlogn)O(n \log n)O(nlogn) (Studocu, s.f.).

Ventajas y desventajas de los algoritmos simples

Ventajas

- Son fáciles de programar e ideales para introducir conceptos fundamentales de ordenación (Scribd, s.f.).
- Requieren poca memoria adicional cuando son in-place (AVS College, s.f.).

- Algunos, como Bubble Sort, son adaptativos y permiten reducir operaciones si la lista ya tiene cierto grado de orden (GeeksforGeeks, s.f.-a).

Desventajas

- Su complejidad O(n2)O(n^2)O(n2) los vuelve ineficientes para grandes volúmenes de datos (BUAP, s.f.).
- No escalan adecuadamente en aplicaciones de alto rendimiento (Studocu, s.f.).
- Pueden presentar tiempos de ejecución altos comparados con algoritmos modernos y optimizados.

Bubble sort

Es un algoritmo de ordenamiento simple que se basa en comparar pares de elementos adyacentes y intercambiarlos si están en el orden incorrecto.

Este proceso se repite varias veces sobre la lista hasta que no se requieren más intercambios, lo que significa que los elementos ya están ordenados. (Cormen et al., 2009).

1. Pasadas

- Se realizan múltiples pasadas sobre el arreglo.
- En cada pasada, el algoritmo revisa los elementos uno por uno.
- Con cada pasada, el mayor de los elementos no ordenados queda ubicado en su posición final.

2. Comparaciones

- En cada pasada, se comparan pares de elementos consecutivos:
  A[i] con A[i+1].
- Si están desordenados, se procede al intercambio.

3. Swaps (intercambios)

- Si A[i] > A[i+1], entonces se realiza un swap.
- Los swaps son la operación más costosa del Bubble Sort.
- En el mejor caso no ocurren swaps; en el peor, ocurren muchos.

Complejidad temporal

El Bubble Sort no es un algoritmo eficiente para grandes volúmenes de datos

Mejor caso: O(n)

Ocurre cuando el arreglo ya está ordenado.
El algoritmo realiza una única pasada sin encontrar intercambios.

Caso promedio: $O(n^2)$

Es el comportamiento típico, donde se realizan muchas comparaciones y algunos swaps.

Peor caso: $O(n^2)$

Ocurre cuando el arreglo está completamente invertido. Requiere el máximo número de comparaciones y swaps.

Pasos del algoritmo Bubble sort

El funcionamiento de Bubble Sort se basa en los siguientes pasos (GeeksforGeeks, s.f.):

1. **Inicio:** parte desde el primer elemento del arreglo.
2. **Comparación:** Compara el elemento actual con el siguiente.
3. **Intercambio (Swap):** Si el elemento actual es mayor que el siguiente, los intercambia.
4. **Avance:** se mueve a la siguiente posición y repite los pasos de comparación e intercambio hasta llegar al final de la parte no ordenada del arreglo.
5. **Repetición:** Repite las pasadas inicio, comparación, intercambio y avance. Después de cada pasada completa, el elemento más grande de la parte no ordenada quedará en su posición correcta al final.
6. **Condición para terminar:** El algoritmo finaliza cuando en una pasada completa no se realiza ningún intercambio debido a que el arreglo ya está ordenado.

Ejemplo práctico método Bubble sort

Para ilustrar el funcionamiento de Bubble Sort, se utilizará el arreglo [5, 1, 4, 2, 8]. El algoritmo compara elementos adyacentes y los intercambia si están en el orden incorrecto. Este proceso se repite en múltiples pasadas hasta que no se requieren más intercambios (Cormen et al., 2009).

En la tabla 1, se muestra organizadamente los resultados obtenidos en la ejecución del método Bubble sort para el arreglo [5, 1, 4, 2, 8], donde se detalla los pasos de este algoritmo mediante las comparaciones e intercambios realizados.

Tabla 1. Comparaciones e intercambios realizados por el método Bubble sort

Número de pasada	Estado del arreglo
Inicial	[5, 1, 4, 2, 8]
Pasada 1	[1, 4, 2, 5, 8]
Pasada 2	[1, 2, 4, 5, 8]
Pasada 3	[1, 2, 4, 5, 8]

Bubble sort: Comparaciones y Movimientos

¿Qué es una comparación?

En el algoritmo de ordenamiento Bubble sort una comparación consiste en una operación que evalúa 2 elementos consecuentes los cuales deben cumplir con determinada condición para ser intercambiados.

¿Qué es un movimiento?

Un movimiento es un proceso que ocurre cuando 2 elementos cambian de lugar, ocurre un swap.

Un swap consta de 3 movimientos empleando una variable temporal.

```
temp = A[i];        // 1 movimiento
A[i] = A[i + 1];    // 1 movimiento
A[i + 1] = temp;    // 1 movimiento
```

Por lo que para cada swap se usan 3 movimientos.

Conteo de operaciones en distintos escenarios

En el algoritmo Bubble Sort clásico, el número de comparaciones NO cambia, sin importar si la lista está ordenada o no.

$$\sum_{i=0}^{n-2}(n-1-i) = \frac{(n-1)n}{2} \in \Theta(n^2).$$

Por lo para todos los casos: El peor, promedio y el mejor se tendrá el mismo orden de complejidad.

Impacto en el rendimiento

Como siempre se conserva el orden de complejidad O(n^2), el rendimiento empeora a medida que aumenta el tamaño del arreglo. Los swaps son mucho más costosos porque involucran demasiadas operaciones de memoria.

Posibles optimizaciones

- **Detener el algoritmo si no se realizó ningún swap**
  Después de cada pasada, al no existir ningún swap significa que el arreglo está ordenado. En el mejor de los casos reduce el orden de complejidad a O(n).
- **Evitar swaps innecesarios**
  Comparar primero y solo mover si es estrictamente necesario. Este proceso aunque ya lo realiza el algoritmo existen variantes que buscan minimizar movimientos por medio de técnicas de “value caching”.
- **Usar un mejor algoritmo**
  Para la mayoría de casos prácticos por ejemplo; insertion sort en mejor para listas casi ordenadas, mientras que Marge Sort/Quick Sort son mejores para listas con tamaños gigantes.

Conclusiones

En conclusión, el análisis de la ordenación interna confirma que los algoritmos de organización más simples son la base esencial en el software. Estos métodos nos permitieron entender de manera clara cómo la velocidad y el uso de memoria impactan el rendimiento de un programa.

Bubble Sort es un algoritmo de ordenamiento sencillo y comprensible que se fundamenta en la comparación e intercambio de elementos adyacentes. Por su complejidad cuadrática en la mayor parte de los casos, su eficacia está limitada, pero es útil para fines educativos y para conjuntos de datos pequeños. Por ende, no se recomienda para aplicaciones que requieren un rendimiento elevado. No obstante, sirve para exponer ideas fundamentales de algoritmos y estructuras de datos.

El método de ordenamiento Bubble sort es de gran utilidad para arreglos pequeños sin embargo es ineficiente para arreglos muy grandes, en donde los algoritmos como Quick Sort o Merge Sort resultan más convenientes y eficientes al momento de ordenar los elementos del arreglo.

Para concluir, el método de Bubble Sort se emplea principalmente para retratar de forma básica cómo operan los algoritmos de ordenamiento, su orden de complejidad con respecto a la cantidad de datos con la que se estén trabajando y cuál sería la opción más eficiente para trabajar con estos datos.

Recomendaciones

Se recomienda que los estudiantes programen y prueben los algoritmos simples en la práctica para entenderlos mejor. Es importante comparar su velocidad con métodos más rápidos para aprender a elegir el algoritmo correcto según el tamaño de los datos.

Es útil comparar Bubble Sort con otros algoritmos simples como Insertion Sort y Selection Sort, ya que estos suelen ser más eficientes en varios escenarios; esto permite entender mejor las limitaciones del método burbuja y por qué su uso se reserva principalmente para fines educativos.

Se recomienda emplear el método de Bubble sort para introducir conceptos básicos para una mayor comprensión del algoritmo y su funcionamiento sin embargo no se recomienda en proyectos donde los arreglos de datos son muy extensos.

Se recomienda usar de forma didáctica el método Bubble Sort con la finalidad de explicar de qué forma interactúan los algoritmos de ordenamiento con los datos alojados en una base de datos y archivo plano.

Referencias

AURS University. (s. f.). Data Structures [PDF]. Recuperado de https://aursbackend.admasuniversity.edu.et/uploads/documents/4f91b8db-dae0-4737-8194-94ad5a70b1f8.pdf

AVS College. (s. f.). DSA Notes: Sorting [PDF]. Recuperado de https://avscollege.ac.in/notes/computer_application/DSA.pdf

Cormen, T. H., et al. (2009). Introduction to Algorithms (3ra ed.). The MIT Press.

EstructuraDatos.tripod.com. (s. f.). Estructuras de Datos – Ordenación. Recuperado de http://estructuradatos.tripod.com/ordenacion.html

GeeksforGeeks. (s. f.). Classification of Sorting Algorithms. Recuperado de https://www.geeksforgeeks.org/dsa/classification-of-sorting-algorithms/

GeeksforGeeks. (s.f.). Bubble Sort. Recuperado de https://www.geeksforgeeks.org/bubble-sort/

Guía de Programación IV, Universidad Don Bosco (UDB). (2019). Guía No. 3: Algoritmos de ordenamiento. Recuperado de https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-iv/2019/ii/guia-3.pdf

Studocu. (s. f.). Ordenación interna y métodos directos [Apuntes]. Recuperado de https://www.studocu.com/latam/document/universidad-americana-paraguay/programacion-avanzada/u4-prog-avanzada-apuntes-de-la-materia/92785915