Informatică – Algoritmi de sortare

Introducere: ce este sortarea și de ce contează

În informatică, sortarea reprezintă procesul de organizare a elementelor unui set de date într-o anumită ordine – fie crescătoare, fie descrescătoare. Poate părea un proces banal, dar sortarea este una dintre cele mai frecvente și esențiale operații în programare.

Fie că e vorba despre:

- afișarea rezultatelor unei căutări pe Google,
- organizarea numelor într-o agendă telefonică,
- indexarea în baze de date sau
- procesarea rapidă a datelor în algoritmi complecși,

sortarea eficientă este crucială. Performanța algoritmului de sortare poate influența în mod semnificativ eficiența întregii aplicații.

Clasificarea algoritmilor de sortare

1. După complexitate

- **Algoritmi simpli ($O(n^2)$)**:
    - *Bubble Sort*
    - *Insertion Sort*
    - *Selection Sort*
- **Algoritmi eficienți ($O(n \log n)$)**:
    - *Merge Sort*
    - *Quick Sort*
    - *Heap Sort*

- **Sortări speciale (O(n))**:
    - *Counting Sort*
    - *Radix Sort*
    - *Bucket Sort*

2. După tipul de sortare

- **Sortare internă** – datele sunt sortate în memorie.
- **Sortare externă** – folosită când datele sunt prea mari pentru a încăpea în RAM (ex: sortarea pe disc).

3. După stabilitate

Un algoritm este **stabil** dacă menține ordinea originală a elementelor egale. Stabilitatea este importantă în cazuri precum sortarea unei liste de utilizatori după nume, dar păstrând ordinea inițială a vârstei.

Exemple de algoritmi

🔁 Bubble Sort

Este cel mai simplu algoritm. Compară elemente adiacente și le interschimbă dacă sunt în ordine greșită. Se repetă acest proces până când lista este sortată.

- Complexitate: $O(n^2)$
- Avantaj: ușor de înțeles
- Dezavantaj: ineficient pentru seturi mari

🪜 Insertion Sort

Inserează fiecare element în poziția corectă față de elementele deja sortate din stânga.

- Complexitate: $O(n^2)$

- Eficient pentru liste mici sau aproape sortate

⚔️ Quick Sort

Alege un pivot și separă lista în două subliste:

- cele mai mici decât pivotul
- cele mai mari

Apoi aplică recursiv algoritmul. Este rapid și eficient în practică.

- Complexitate: O(n log n) în medie, $O(n^2)$ în cel mai rău caz
- Nu este stabil

🧬 Merge Sort

Împarte lista în jumătăți, sortează fiecare jumătate recursiv, apoi le îmbină.

- Complexitate: O(n log n)
- Stabil, dar necesită spațiu suplimentar

⚙️ Heap Sort

Transformă lista într-un heap (arbore binar complet), extrage elementul maxim și reconstruiește heapul.

- Complexitate: O(n log n)
- Nu este stabil

Sortări necomparative

Acestea nu folosesc comparații directe, ci se bazează pe proprietăți specifice ale datelor.

🎯 Counting Sort

Funcționează doar pe numere întregi dintr-un interval restrâns. Numără de câte ori apare fiecare element și reconstruiește lista sortată.

- Complexitate: O(n + k), unde k este gama valorilor
- Extrem de eficient pentru date uniforme

🧮 Radix Sort

Sortează elementele cifră cu cifră, folosind o sortare stabilă la fiecare pas.

- Ideal pentru numere mari sau șiruri
- Utilizat în aplicații industriale

Aplicații reale

- Motoare de căutare: sortarea paginilor după relevanță
- E-commerce: sortarea produselor după preț sau popularitate
- Gaming: leaderboard-uri și statistici
- Baze de date: optimizarea interogărilor (ex: ORDER BY)
- Inteligență artificială: preprocesarea datelor de intrare

Compararea performanței

Algoritm	Timp mediu	Spațiu	Stabil
Bubble Sort	$O(n^2)$	O(1)	Da
Quick Sort	O(n log n)	O(log n)	Nu
Merge Sort	O(n log n)	O(n)	Da

Heap Sort	O(n log n)	O(1)	Nu
Radix Sort	O(nk)	O(n+k)	Da

Concluzie

Înțelegerea algoritmilor de sortare nu înseamnă doar cunoașterea unor proceduri tehnice – înseamnă stăpânirea uneia dintre fundamentele eficienței în programare. Alegerea celui mai potrivit algoritm depinde de tipul datelor, dimensiunea datasetului și cerințele de viteză și memorie. Sortarea este o artă la intersecția dintre teorie și practică.