Resurse examen SD

Seria 15

June 25, 2025

Contents

1 Sortări

Algoritmii de sortare pot fi clasificați după complexitate, spațiu, stabilitate și dacă se bazează pe comparații:

1.1 Clasificare

- **Elementari:** Insertion, Selection, Bubble
- **Divide et Impera:** Merge Sort, Quick Sort
- **Heap-based:** Heap Sort
- **Non-comparative:** Counting Sort, Radix Sort, Bucket Sort

1.2 Stabilitate

Un algoritm de sortare este *stabil* dacă menține ordinea relativă a elementelor egale. Acest lucru este important când sortăm obiecte cu multiple chei.

1.3 Tabel de complexități

Algorithm	**Time Complexity**			**Space (worst)**
	Best	**Average**	**Worst**	
Quicksort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n^2)$	$O(\log n)$
Mergesort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$	$O(n)$
Timsort	$\Omega(n)$	$\Theta(n \log n)$	$O(n \log n)$	$O(n)$
Heapsort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$	$O(1)$
Bubble Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Tree Sort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n^2)$	$O(n)$
Shell Sort	$\Omega(n \log n)$	$\Theta\big(n(\log n)^2\big)$	$O\big((n \log n)^2\big)$	$O(1)$
Bucket Sort	$\Omega(n + k)$	$\Theta(n + k)$	$O(n^2)$	$O(n + k)$
Radix Sort	$\Omega(n\,k)$	$\Theta(n\,k)$	$O(n\,k)$	$O(n + k)$
Counting Sort	$\Omega(n + k)$	$\Theta(n + k)$	$O(n + k)$	$O(n + k)$
Cubesort	$\Omega(n)$	$\Theta(n \log n)$	$O(n \log n)$	$O(n)$

Table 1: Array Sorting Algorithms: time and space complexities

1.4 Informații generale

- Sortările prin comparație au limită inferioară $\Omega(n \log n)$.
- Algoritmii non-comparativi (Counting, Radix) pot ajunge la $O(n)$ în condiții favorabile.
- Alegerea pivotului în Quick Sort influențează performanța:
  - pivot ales random
  - mediana din 3/5/7
  - mediana medianelor
- Pentru vectori mici, sortarea cu algoritmi $O(n^2)$ poate fi mai eficientă decât în $O(n \log n)$ (constanta poate fi mare).

1.5 Counting sort

Se creează un vector de frecvență cu valorile pe care trebuie să le sortăm. Valorile din vector sunt numărate, iar vectorul este reconstruit în ordine sortată prin parcurgerea vectorului de frecvență.

Se aplică când elementele sunt numere întregi într-un interval $[0, k]$, (până la 10^6)

1.6 Bucket sort

Se creează k buckets și fiecare element x din vectorul de intrare este distribuit în găleata indexată de o funcție de mapare (de ex. $\lfloor k \cdot \frac{x-a}{b-a} \rfloor$ pentru valori în $[a, b]$). Fiecare găleată este sortată intern (adesea prin Insertion Sort), apoi se concatenează conținutul găleților pentru a obține vectorul sortat.

1.7 Radix sort

Sortarea se face pe mai multe trepte de cifre: pentru fiecare poziție de cifră (de la LSD la MSD sau invers) se aplică Counting Sort pentru a grupa elementele după valoarea cifrei curente. După procesarea tuturor cifrelor, vectorul devine complet sortat.

2 Complexități

În studiul algoritmilor, folosim trei noțiuni fundamentale pentru a măsura creşterea funcţiei de timp $T(n)$ în raport cu dimensiunea de intrare n:

- **O** (Big-O): reprezintă o limită superioară asimptotică.

$$T(n) = O(f(n)) \iff \exists\, c > 0,\ n_0 :\ \forall n \geq n_0,\ T(n) \leq c\, f(n).$$

- **Ω** (Big-Omega): reprezintă o limită inferioară asimptotică.

$$T(n) = \Omega(f(n)) \iff \exists\, c > 0,\ n_0 :\ \forall n \geq n_0,\ T(n) \geq c\, f(n).$$

- **Θ** (Big-Theta): când există simultan limite superioară și inferioară de același ordin.

$$T(n) = \Theta(f(n)) \iff T(n) = O(f(n)) \ \wedge\ T(n) = \Omega(f(n)).$$

Astfel, $\Theta(f(n))$ indică creşterea „exactă” (în sens asimptotic), iar O şi Ω doar conturul superior, respectiv inferior.