

Lernzettel

Suchen und Sortieren in Feldern: Algorithmen und Komplexität

Universität: Technische Universität Berlin
Kurs/Modul: Einführung in die Informatik - Vertiefung
Erstellungsdatum: September 20, 2025



Zielorientierte Lerninhalte, kostenlos!
Entdecke zugeschnittene Materialien für deine Kurse:

<https://study.AllWeCanLearn.com>

Einführung in die Informatik - Vertiefung

Lernzettel: Suchen und Sortieren in Feldern: Algorithmen und Komplexität**(1) Suchen in Feldern**

Lineare Suche. Durchsuche das Feld von links nach rechts, bis der gesuchte Wert gefunden ist oder das Feld zu Ende ist. Im schlimmsten Fall werden alle Elemente geprüft.

$$T_{\text{lin}}(n) = n \quad (\text{Worst-case, Anzahl Vergleiche})$$

$$T_{\text{lin, best}} = 1$$

$$T_{\text{lin, avg}} = \frac{n+1}{2}$$

Binäre Suche. Voraussetzung: das Feld ist sortiert. Es wird wiederholt das mittlere Element mit dem gesuchten Wert verglichen und das Suchfeld entsprechend halbiert.

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1 \quad T(1) = 1$$

$$T(n) = \Theta(\log n)$$

$$T_{\text{Worst}}(n) = \left\lceil \log_2 n \right\rceil + 1$$

$$S_{\text{iter}}(n) = O(1) \quad (\text{zusätzlicher Speicher})$$

$$S_{\text{rec}}(n) = O(\log n) \quad (\text{rekursiver Ansatz})$$

Beobachtungen. - Binäre Suche ist nur sinnvoll, wenn das Feld sortiert ist. - Zeitkomplexität wächst logarithmisch, daher schnell für große Felder. - Iterative Implementierung benötigt weniger Stapelraum als rekursive.

(2) Sortieren in Feldern

Bubble Sort. Ein einfacher sortieralgorithmus, der benachbarte Elemente vertauscht, bis das Feld sortiert ist.

$$T_{\text{Bub}}(n) \in \{O(n^2), \Theta(n^2)\} \quad (\text{Durchschnitt/Worst})$$

$$T_{\text{Bub, best}} = O(n) \quad (\text{mit Optimierung: Abbruch, falls kein Tausch})$$

$$S_{\text{Bub}}(n) = O(1)$$

Stabil: Ja, In-Place: Ja.

Insertion Sort. Zieht jedes Element an die richtige Position in dem bereits sortierten Teil.

$$T_{\text{Ins}}(n) \in \{O(n^2), \Theta(n^2)\} \quad (\text{Durchschnitt/Worst})$$

$$T_{\text{Ins, best}} = O(n)$$

$$S_{\text{Ins}}(n) = O(1)$$

Stabil: Ja, In-Place: Ja.

Selection Sort. Wählt wiederholt das kleinste Element aus dem unsortierten Teil aus und platziert es am Anfang.

$$T_{\text{Sel}}(n) = O(n^2) \quad (\text{Best/Worst/Average})$$
$$S_{\text{Sel}}(n) = O(1)$$

Stabil: Nein, In-Place: Ja.

Merge Sort. Teilt das Feld in Hälften, sortiert Teillfelder rekursiv und führt sie zusammen.

$$T_{\text{Merge}}(n) = O(n \log n) \quad (\text{allgemein})$$
$$S_{\text{Merge}}(n) = O(n) \quad (\text{zusätzlicher Speicher})$$

Stabil: Ja, In-Place: Nein (in Standardvariante).

Quick Sort. Wählt einen Pivot, teilt das Feld in zwei Teile und sortiert rekursiv.

$$T_{\text{Quick}}(n) \approx O(n \log n) \quad (\text{Durchschnitt})$$
$$T_{\text{Quick}}(n) = O(n^2) \quad (\text{Schlechtester Fall, z. B. Pivot am Rand})$$
$$S_{\text{Quick}}(n) = O(\log n) \quad (\text{Stapelraum})$$

Stabil: Nein, In-Place: Ja.

Wichtige Beobachtungen zur Auswahl. - Für große Datenmengen mit begrenztem Speicher ist Quick Sort oft bevorzugt, wenn keine Stabilität verlangt wird. - Merge Sort ist stabil und zuverlässig in ' $O(n \log n)$ ', benötigt aber zusätzlichen Speicher. - Insertion Sort ist bei fast sortierten Feldern sehr gut (linear), ansonsten quadratic. - Bubble und Selection Sort sind einfach, aber meist ineffizient für große Felder.

(3) Komplexität, Begriffe und Kriterien der Analyse

Grundbegriffe. - Groß-O-Notation: $f(n) = O(g(n))$ bedeutet, dass $f(n)$ sich höchstens wie $g(n)$ verhält für hinreichend großes n .

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

- Theta-Notation: $f(n) = \Theta(g(n))$ bedeutet $f(n)$ ist sowohl $O(g(n))$ als auch $\Omega(g(n))$.

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

Beste, durchschnittliche und schlechteste Laufzeit. - Best Case: minimale Anzahl Vergleiche/Zugriffe, oft linear oder konstant. - Average Case: typische mittlere Kosten; häufig komplizierter zu bestimmen, oft durch Modelle angenähert. - Worst Case: maximale Kosten, oft durch adversarische Eingaben erreicht.

Platzbedarf. - In-Place bedeutet, dass nur eine kleine feste Menge zusätzlicher Speicher benötigt wird (oft $O(1)$). - Nicht in-place bedeutet zusätzlicher Speicher $O(n)$ (z. B. Merge Sort).

Stabilität. - Ein Sortieralgorithmus ist stabil, wenn gleiche Schlüssel ihre relative Reihenfolge behalten. - Stabilität ist wichtig, wenn zusätzliche Strukturen (z. B. Partnerdaten) vorhanden sind.

(4) Praktische Hinweise zur Lehre und Anwendung

- Wähle Sortieralgorithmus je nach Feldgröße, benötigtem Speicher und Stabilitätsanforderung.
- Für grobe Schätzungen genügt oft $O(n \log n)$ als Obergrenze bei Sortierungen; bei sehr kleinen Feldern ist Insertion Sort oft praktischer. - Beim Suchen gilt: sortierte Felder bevorzugen binäre Suche; unsortierte Felder nutzen lineare Suche.

Zusammenfassung. - Suchen: Lineare Suche – einfach, lineare Zeit; Binäre Suche – schnell, sortiertes Feld, logarithmische Zeit. - Sortieren: Bubble, Insertion, Selection – einfach, oft quadratic; Merge und Quick Sort – effizient, je nach Stabilität und Speicherbedarf unterschiedlich.