

Suchen und Sortieren

Reiner Hüchting

9. März 2023

Themenüberblick

Suchverfahren

Sortierverfahren

Themenüberblick

Suchverfahren

Sortiervverfahren

Themenüberblick

Suchverfahren

Lineare Suche

Binäre Suche

Sortierverfahren

Suchverfahren – Lineare Suche

Ziel: Finde die Position eines Elements in einer Liste

- ▶ Naiver Ansatz: Durchsuche die Liste Element für Element von Anfang bis Ende.

Vorteil

- ▶ Funktioniert für jede Liste.

Komplexität

- ▶ Linear in der Länge der Liste (Schreibe: $O(n)$).
- ▶ Bei Länge n müssen im Worst Case alle n Elemente mit dem gesuchten verglichen werden.

Themenüberblick

Suchverfahren

Lineare Suche

Binäre Suche

Sortierverfahren

Suchverfahren – Binäre Suche

Ziel: Finde die Position eines Elements in einer Liste

- ▶ Ansatz: Vergleiche das mittlere Element mit dem gesuchten.
- ▶ Fahre entweder nur links oder nur rechts der Mitte fort.

Vor- und Nachteile

- ▶ Funktioniert nur für sortierte Listen.
- ▶ Ist erheblich schneller als die lineare Suche.

Komplexität

- ▶ Logarithmisch in der Länge der Liste (Schreibe: $O(\log n)$).
- ▶ In jedem Schritt wird der Suchraum halbiert („**Divide and Conquer**“).
- ▶ Bei Länge n müssen im Worst Case nur $\log_2 n$ Elemente mit dem gesuchten verglichen werden.

Themenüberblick

Suchverfahren

Sortierverfahren

Themenüberblick

Suchverfahren

Sortierverfahren

Insertion Sort

Selection Sort

Bubble Sort

Quick Sort

Merge Sort

Sortiervverfahren – Insertion Sort

Sortieren durch Einfügen

- ▶ Ansatz: Nimm das nächstbeste Element und füge es an der passenden Stelle ein.

Vorteil

- ▶ schnell für kurze Listen
- ▶ einfach zu verstehen und zu implementieren.
- ▶ typischer Aufräum-Ansatz

Komplexität

- ▶ Quadratisch in der Länge der Liste (Schreibe: $O(n^2)$).
- ▶ Bei Länge n müssen n Elemente einsortiert werden.
- ▶ Jedes Einsortieren dauert bis zu n Schritte.

Sortierverfahren – Insertion Sort

Sortieren durch Einfügen

- ▶ Ansatz: Nimm das nächstbeste Element und füge es an der passenden Stelle ein.

typische Implementierung für das Einsortieren eines Elements

- ▶ Füge das nächste Element am Ende der Liste an.
- ▶ Tausche es solange nach links, bis es größer als sein linker Nachbar ist.

Beobachtung

- ▶ kann sehr effizient **in place** umgesetzt werden.
- ▶ d.h. ohne eine separate Hilfsliste

Themenüberblick

Suchverfahren

Sortierverfahren

Insertion Sort

Selection Sort

Bubble Sort

Quick Sort

Merge Sort

Sortierv Verfahren – Selection Sort

Sortieren durch Auswählen

- ▶ Ansatz: Suche das kleinste Element aus dem noch unsortierten Teil und hänge es ans Ende der sortierten Liste.

Vorteil

- ▶ schnell für kurze Listen
- ▶ einfach zu verstehen und zu implementieren.
- ▶ typischer Aufräum-Ansatz

Komplexität

- ▶ Quadratisch in der Länge der Liste (Schreibe: $O(n^2)$).
- ▶ Bei Länge n müssen n Elemente gesucht werden.
- ▶ Jede Suche dauert bis zu n Schritte.

Sortierv Verfahren – Selection Sort

Sortieren durch Auswählen

- ▶ Ansatz: Suche das kleinste Element aus dem noch unsortierten Teil und hänge es ans Ende der sortierten Liste.

typische Implementierung für das Einsortieren eines Elements

- ▶ Suche das kleinste Element im unsortierten Teil der Liste.
- ▶ Vertausche das Element mit dem ersten noch nicht einsortierten.

Beobachtung

- ▶ kann sehr effizient **in place** umgesetzt werden.
- ▶ d.h. ohne eine separate Hilfsliste

Themenüberblick

Suchverfahren

Sortierverfahren

Insertion Sort

Selection Sort

Bubble Sort

Quick Sort

Merge Sort

Sortierv Verfahren – Bubble Sort

Sortieren durch Aufsteigen

- ▶ Ansatz: Tausche nach und nach Elemente nach rechts, wenn sie größer als ihre Nachbarn sind.

Vorteile

- ▶ schnell für kurze Listen
- ▶ sehr intuitiv
- ▶ Lokales Verhalten: Vergleiche nur benachbarte Elemente.

Komplexität

- ▶ Quadratisch in der Länge der Liste (Schreibe: $O(n^2)$).
- ▶ Bei Länge n müssen n Elemente aufsteigen.
- ▶ Jeder Durchlauf dauert bis zu n Schritte.

Sortiervverfahren – Bubble Sort

Sortieren durch Aufsteigen

- ▶ Ansatz: Tausche nach und nach Elemente nach rechts, wenn sie größer als ihre Nachbarn sind.

Beobachtung

- ▶ kann sehr effizient **in place** umgesetzt werden.
- ▶ d.h. ohne eine separate Hilfsliste

Analyse

- ▶ Große Elemente am Anfang steigen schnell auf.
- ▶ Kleine Elemente am Ende sinken nur langsam ab.
- ▶ \Rightarrow langsam bei (fast) umgekehrt sortierten Listen

Sortierv Verfahren – Bubble Sort

Beobachtung bei BubbleSort

- ▶ Große Elemente steigen schnell auf, kleine sinken langsam ab.

Weiterentwicklung: CombSort/GapSort

- ▶ Ansatz: Vergleiche und vertausche am Anfang Elemente mit größerem Abstand
- ▶ Komplexität im Best Case: $O(n \log n)$.
- ▶ Komplexität im Worst Case: $O(n^2)$.

Weiterentwicklung: CocktailSort

- ▶ Ansatz: Wie bei BubbleSort, aber wechsele die Richtungen ab.
- ▶ Vorteil: Alle Elemente bewegen sich ungefähr gleich schnell.
- ▶ Komplexität im Best und Worst Case: $O(n^2)$.

Themenüberblick

Suchverfahren

Sortierverfahren

Insertion Sort

Selection Sort

Bubble Sort

Quick Sort

Merge Sort

Sortiervverfahren – Quick Sort

Schnelles Divide and Conquer-Verfahren

- ▶ Ansatz: Sortiere Elemente bzgl eines Referenzelements vor. Sortiere anschließend rekursiv die Teillisten.

Vorteile

- ▶ schnell für lange Listen
- ▶ Gilt (mit Modifikationen) als das schnellste verfügbare Sortiervverfahren.

Komplexität

- ▶ Worst Case: $O(n^2)$.
- ▶ Average- und Best-Case: $O(n \log n)$.
- ▶ Vorsortieren braucht n Vergleiche.
- ▶ Im Idealfall wird mit jedem Schritt die Liste halbiert.

Sortierv Verfahren – Quick Sort

Schnelles Divide and Conquer-Verfahren

- ▶ Ansatz: Sortiere Elemente bzgl eines Referenzelements vor. Sortiere anschließend rekursiv die Teillisten.

Modifikationen

- ▶ Für kurze Listen auf *InsertionSort* ausweichen.
- ▶ Rekursionstiefe begrenzen: Auf *MergeSort* wechseln, um $O(n \log n)$ im Worst-Case zu garantieren.

Beobachtung

- ▶ kann sehr effizient **in place** umgesetzt werden.
- ▶ Der Worst-Case ist gerade die umgekehrt sortierte Liste.
- ▶ gut parallelisierbar

Themenüberblick

Suchverfahren

Sortierverfahren

Insertion Sort

Selection Sort

Bubble Sort

Quick Sort

Merge Sort

Sortiervverfahren – Merge Sort

Schnelles Divide and Conquer-Verfahren

- ▶ Ansatz: Halbiere die Liste rekursiv, bis nur noch einzelne Elemente übrig sind. Setze dann sortierte Listen zu längeren sortierten Listen zusammen.

Vorteile

- ▶ schnell für lange Listen

Komplexität

- ▶ Worst Case: $O(n \log n)$.
- ▶ Mit jedem Schritt wird die Liste halbiert.
- ▶ Zusammensetzen dauert n Schritte.

Sortierverfahren – Merge Sort

Schnelles Divide and Conquer-Verfahren

- ▶ Ansatz: Halbiere die Liste rekursiv, bis nur noch einzelne Elemente übrig sind. Setze dann sortierte Listen zu längeren sortierten Listen zusammen.

Modifikationen

- ▶ *TimSort*: Identifiziere bereits sortierte Teillisten und spare diese bei der Rekursion aus.
- ▶ Standard-Sortierverfahren in Python

Beobachtung

- ▶ i.d.R. nicht in place umgesetzt (braucht Hilfsarrays der Länge $n/2$)
- ▶ gut für verkettete Listen
- ▶ gut parallelisierbar