Informatik
Grundlagen

Abitur Vorbereitung

Digital Handbook · Druckversion

Lerntracker

Behalte deinen Fortschritt im Blick – trage Datum und Notizen ein

Thema	Status	Datum	Notizen
Laufzeit	☐		
Komplexität	☐		
Bubble Sort	☐		
Insertion Sort	☐		
Selection Sort	☐		
Merge Sort	☐		
Quick Sort	☐		
Dynamische Reihung	☐		
Stapel (Stack)	☐		
Schlange (Queue)	☐		
Binärbaum	☐		
UML-Klassendiagramme	☐		
DEA	☐		
NEA	☐		
Kellerautomat	☐		
Mealey	☐		
Moore	☐		
Grammatiken	☐		
ER-Modell	☐		

Thema	Status	Datum	Notizen
Normalformen	⬛		
Anomalien	⬛		
SQL	⬛		
ASCII	⬛		
Binärcodierung	⬛		
Huffman	⬛		
Caesar	⬛		
Vigenère	⬛		

1 · Algorithmik

1.1 Laufzeit

Die Laufzeitanalyse hilft, Algorithmen miteinander zu vergleichen und vorherzusagen, wie sie sich bei wachsender Eingabegröße verhalten. Im Abitur: O-Notation (obere Schranke). n = Anzahl der Elemente, Laufzeit = Funktion von n.

Notation	Name	Beispiel	Qualität
O(1)	konstant	Zugriff auf ein Element, einfache Berechnung, logische Abfrage	sehr gut
O(log n)	logarithmisch	Binäre Suche (sortiertes Array) – Suchbereich halbiert sich jedem Schritt	sehr gut
O(n)	linear	Lineare Suche – im Worst Case wird jedes Element einmal betrachtet	gut
O(n log n)	effizient	Merge Sort: Aufteilen O(log n) + Zusammenführen O(n) = O(n log n)	gut
O(n²)	quadratisch	Bubble Sort – zwei verschachtelte Schleifen ⇒ n×n Schritte	mittel
O(2^n)	exponentiell	Rekursive Fibonacci-Berechnung – jeder Aufruf erzeugt zwei weitere	schlecht

Je langsamer die Laufzeit wächst, desto besser ist der Algorithmus für große Eingaben.

Fallunterscheidung:

Fall	Bedeutung	Beispiel (lineare Suche)
Best Case	günstigster Fall	Element an Position 0 → O(1)
Average Case	durchschnittlicher Fall	—
Worst Case	ungünstigster Fall	Element am Ende / nicht vorhanden → O(n)

1.2 Komplexität

Beschreibt den Ressourcenverbrauch eines Algorithmus.

Art	Beschreibt
Zeitkomplexität	Laufzeit des Algorithmus
Speicherkomplexität	Zusatzspeicher, den der Algorithmus benötigt

1.3 Sortierverfahren

Im Abi: Wie funktioniert der Algorithmus, welche Laufzeit hat er, wann ist er sinnvoll. Fokus liegt auf Vergleich der Verfahren, nicht auf komplexem Code.

Algorithmus	Best Case	Worst Case	Stabil?	Beschreibung
Bubble Sort	O(n)	O(n²)	✓	Benachbarte Elemente vergleichen und tauschen. Große Elemente 'wandern' nach rechts. Einfach, aber langsam.
Insertion Sort	O(n)	O(n²)	✓	Nächstes Element nehmen und an der richtigen Stelle einfügen. Gut für kleine Daten.
Selection Sort	O(n²)	O(n²)	✗	Kleinstes Element suchen und an die richtige Position tauschen. Immer gleiche Laufzeit.

Algorithmus	Best Case	Worst Case	Stabil?	Beschreibung
Merge Sort	O(n log n)	O(n log n)	✓	Rekursiv teilen, dann sortierte Teillisten zusammenführen. Schnell und stabil, braucht Extraspeicher.
Quick Sort	O(n log n)	O(n^2)	✗	Pivot-Element wählen und Liste in kleinere/größere Elemente teilen. Durchschnittlich sehr schnell.

Stabile Algorithmen (Reihenfolge gleicher Elemente bleibt erhalten): Bubble Sort, Insertion Sort, Merge Sort. Nicht stabil: Selection Sort, Quick Sort.