
AuD - Zusammenfassung

Moritz Gerhardt

Contents

Was ist ein Algorithmus?	2
Sektion 2	2
2.1 Sortierproblem	2
2.2 Insertion Sort	3
2.3	4

Sektion 1 Was ist ein Algorithmus?

Ein Algorithmus beschreibt eine Handlungsvorschrift zur Umwandlung von Eingaben in eine Ausgabe. Dabei sollte ein Algorithmus im allgemeinen folgende Voraussetzungen erfüllen:

1. Bestimmt:
 - Determiniert: Bei gleicher Eingabe liefert der Algorithmus gleiche Ausgabe.
 \implies Ausgabe nur von Eingabe abhängig, keine äußeren Faktoren.
 - Determinismus: Bei gleicher Eingabe läuft der Algorithmus immer gleich durch die Eingabe.
 \implies Gleiche Schritte, Gleiche Zwischenstände.
2. Berechenbar:
 - Finit: Der Algorithmus ist als endlich definiert. (Theoretisch)
 - Terminierbar: Der Algorithmus stoppt in endlicher Zeit. (Praktisch)
 - Effektiv: Der Algorithmus ist auf Maschine ausführbar.
3. Anwendbar:
 - Allgemein: Der Algorithmus ist für alle Eingaben einer Klasse anwendbar, nicht nur für speziellen Fall.
 - Korrekt: Wenn der Algorithmus ohne Fehler terminiert, ist die Ausgabe korrekt.

Sektion 2 Sortieren

2.1 Sortierproblem

Sortieralgorithmen sind die wohl am häufigsten verwendeten Algorithmen. Hierbei wird als Eingabe eine Folge von Objekten gegeben, die nach einer bestimmten Eigenschaft sortiert werden. Der Algorithmus soll die Eingabe in der richtigen Reihenfolge (nach einer bestimmten Eigenschaft) zur Ausgabe umwandeln. Es wird hierbei meist von einer total geordneten Menge ausgegangen. (Alle Elemente sind miteinander vergleichbar).

Eine Totale Ordnung wie folgt definiert:

Eine Relation \leq auf M ist eine totale Ordnung, wenn:

- Reflexiv: $\forall x \in M : x \leq x$
(x steht in Relation zu x)
- Transitiv: $\forall x, y, z \in M : x \leq y \wedge y \leq z \implies x \leq z$
(Wenn x in Relation zu y steht und y in Relation zu z steht, so folgt, dass x in Relation zu z steht)
- Antisymmetrisch: $\forall x, y \in M : x \leq y \wedge y \leq x \implies x = y$
(Wenn x in Relation zu y steht und y in Relation zu x steht, so folgt, dass $x = y$)
- Totalität: $\forall x, y \in M : x \leq y \vee y \leq x$
(Alle Elemente müssen in einer Relation zueinander stehen)

2.2 Insertion Sort

```
1 Function insertion_sort(A):
2   for i = 1 to A.length - 1 do
3     key = A[i]                                ▷ Element zum Sortieren
4     j = i - 1                                    ▷ Einfügapunkt wird von hinten gesucht
5     while j >= 0 and A[j] > key do
6       A[j + 1] = A[j]                          ▷ Elemente nach Rechts verschieben
7       j = j - 1
8     end
9     A[j + 1] = key                                ▷ Element wird in den Einfügapunkt geschoben
10  end
11 end
```

Prinzip: Die Eingabe wird von links nach rechts durchlaufen. Dafür wird für jedes Element

2.3 Merge Sort

```
1 Function mergeSort(A, left, right):
2   if left < right then
3      $\triangleright$  Wenn Bereich ist nicht leer  $\text{mid} = \text{floor}((\text{left} + \text{right}) / 2)$   $\triangleright$  Nach unten gerundet
4     mergeSort(A, left, mid)  $\triangleright$  Sortiert von left zu mid
5     mergeSort(A, mid + 1, right)  $\triangleright$  Sortiert von mid + 1 zu right
6     mergeA, left, mid, right  $\triangleright$  Fügt Hälften zusammen
7   end
8 end
```

```
1 Function merge(A, left, mid, right):
2   B = new Array[right - left + 1]  $\triangleright$  Temp array
3   p = left
4   q = mid + 1
5    $\triangleright$  Array A wird mithilfe von zwei Pointern durchgegangen, von links und von mid + 1
6   for i = 0 to right - left do
7      $\triangleright$  Lläuft für jedes Element im Zielbereich
8     if q > right or (p =< mid and A[p] =< A[q]) then
9        $\triangleright$  Wenn q > right, dann ist der rechte Teil durchlaufen. Wenn p =< mid ist, so ist der linke Teil
          noch nicht durchlaufen. Wenn das linke Element =< dem rechten ist, dann wird das Element
          dem temp array B hinzugefügt.
10      B[i] = A[p]
11      p = p + 1
12    end
13    else
14       $\triangleright$  Wenn oben nicht zutrifft:
15      B[i] = A[q]
16      q = q + 1
17    end
18  end
19  for i = 0 to right - left do
20     $\triangleright$  Lläuft wieder für jedes Element im Zielbereich
21     $\triangleright$  Kopiert den jetzt sortierten temp array B zurück in den eigentlichen Array an der richtigen Stelle
22    A[i + left] = B[i]  $\triangleright$  i + left, damit nicht am Anfang sondern an dem richtigen Teilbereich eingefügt
        wird.
23  end
24 end
```