Datenstrukturen und Algorithmen

Abgabe: 17.05.2017

Georg C. Dorndorf Matr.Nr. 366511
Adrian C. Hinrichs Matr.Nr. 367129

Aufgabe 5

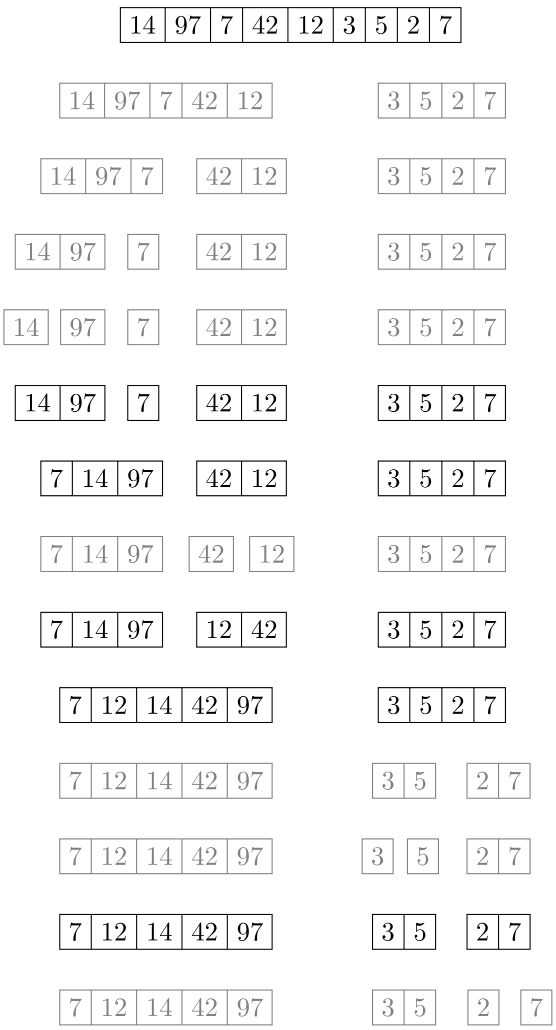


Abbildung 1: Visualisierung des Merge-Algorithmus (1/2)

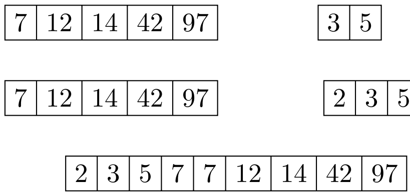
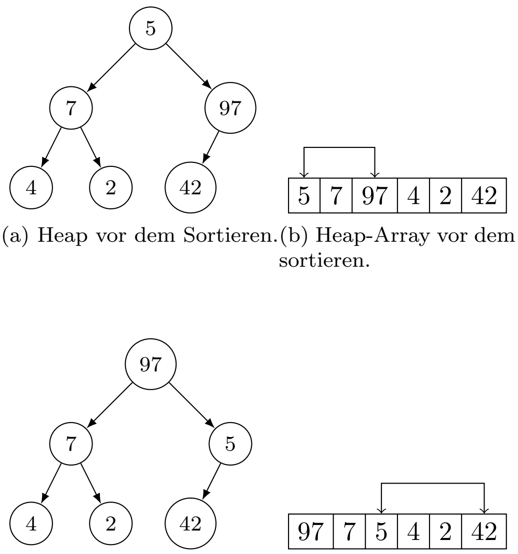


Abbildung 2: Visualisierung des Merge-Algorithmus (1/2)

Die ausführung des Mergesort-Algorithmus ist in den Abbildungen 1 und 2 dargestellt. Die grauen Zeilen stellen das Produkt einer Split-Operation dar und dienen daher nur der veranschaulichung.

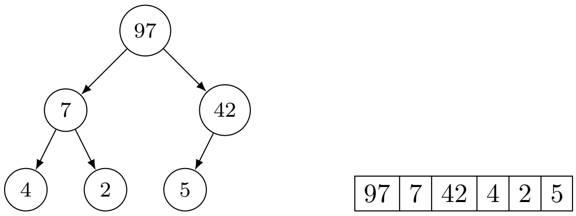
Aufgabe 6



(a) Heap vor dem Sortieren. (b) Heap-Array vor dem sortieren.

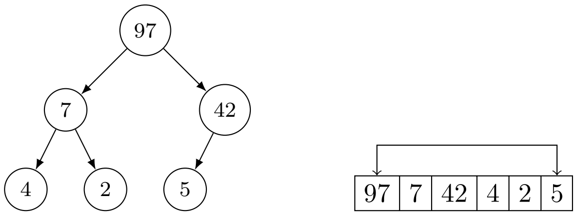
(c) Heap vor dem Sortieren. (d) Heap-Array vor dem sortieren.

Abbildung 3: Aufbau des Heaps



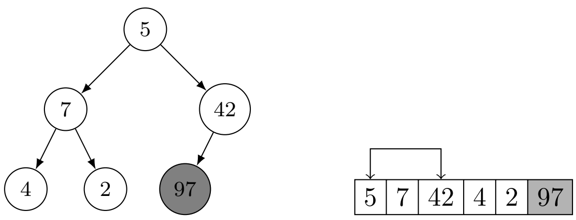
(a) Heap vor dem Sortieren. (b) Heap-Array vor dem sortieren.

Abbildung 4: Fertig aufgebauter Heap



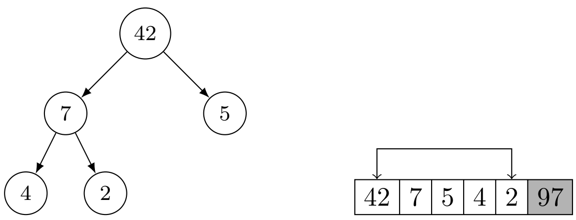
(a) Erste Iteration (b) Erste Iteration

Abbildung 5: Sortiervorgang



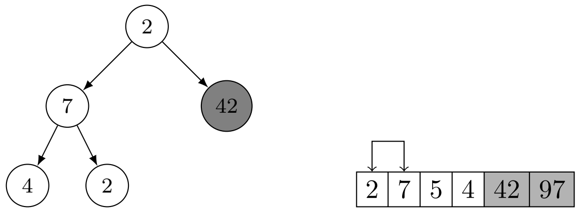
(a) Zweite Iteration (b) Zweite Iteration

Abbildung 6: Sortiervorgang



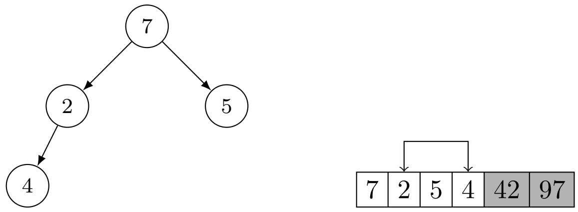
(a) Dritte Iteration (b) Dritte Iteration

Abbildung 7: Sortiervorgang



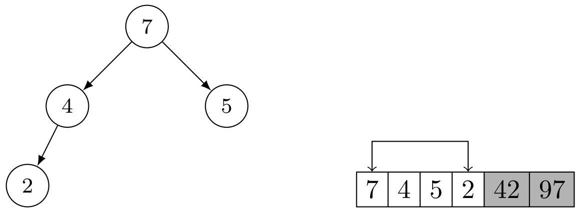
(a) Vierte Iteration (b) Vierte Iteration

Abbildung 8: Sortiervorgang



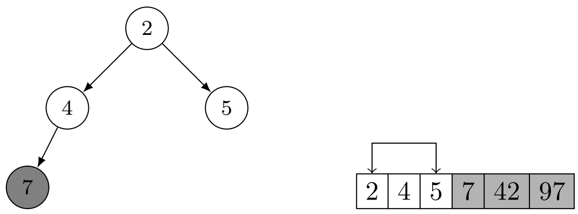
(a) Fünfte Iteration (b) Fünfte Iteration

Abbildung 9: Sortiervorgang



(a) Sechste Iteration (b) Sechste Iteration

Abbildung 10: Sortiervorgang



(a) Siebte Iteration (b) Siebte Iteration

Abbildung 11: Sortiervorgang

(a) Achte Iteration (b) Achte Iteration

Abbildung 12: Sortiervorgang

(a) Neunte Iteration (b) Neunte Iteration

Abbildung 13: Sortiervorgang

(a) Zehnte Iteration (b) Zehnte Iteration

Abbildung 14: Sortiervorgang

(a) Elfte Iteration (b) Elfte Iteration

Abbildung 15: Sortiervorgang

(a) Fertig

Abbildung 16: Sortiervorgang

QEF

Aufgabe 7

a)

Es ergibt sich für Timmis MERGE-SORT Algorithmus sowohl im Best- als auch im Worst- und Average-Case eine Komplexitätsrekursionsgleichung $T(n)$ von (I).

$$T(n) = \begin{cases} 1 & \text{, falls } n = 1 \\ 3 \cdot T(\frac{n}{3}) + \Theta(n), & \text{, falls } n \neq 1 \end{cases} \tag{I}$$

Diese Rekursionslgeichung ergibt sich, da MERGE-SORT ein Divide and Conquer Algorithmus ist. Timmis Variante von MERGE-SORT teilt dabei jedes Problem in 3 Unterteile wobei wir vereinfachend annehmen, dass für n gilt $n = 3^m$ mit $m \in \mathbb{N}$. Diese Annahme beeinflusst nicht die Größenordnung des Wachstums der Komplexität.

Es fallen also beim teilen des Problems 3 gleich große Teilpobleme an, die jeweils lediglich $\frac{n}{3}$ groß sind. Daraus ergibt sich für diese Operation eine Laufzeitkomplexität von $T(n) = 3 \cdot T(\frac{n}{3})$. Die Merge funktion wird jede Rekursionstiefe auf n Elemente angewandt dies ergibt sich aus der Annahme, dass `merge(int[] A, int l, int mid, int r)` nach Aufgabe $\Theta(r - l + 1)$ viele kostet also eine Komplexität von $T(n) = \Theta(n)$. Aus der Abbruchbedingung der Rekursion erhält man offensichtlich $T(1) = 1$. Insgesamt ehalten wir als Rekusionsgleichung für alle drei Fälle (I).

Es fällt auf, dass die Rekursionsgleichung leicht mittels des MASTER-THEOREMS einer Komplexitätsklasse zugeordnet werden kann. Es folgt also mithilfe des MASTER-THEOREMS:

$$T(n) = 3 \cdot T(\frac{n}{3}) + \Theta(n) \tag{II}$$

Wir definieren:

$$b := 3; \; c := 3; \; f(n) := n; \text{ und}$$
$$E := \frac{\log 3}{\log 3} = 1 \tag{III}$$

Es fällt auf, dass $f(n) \in \Theta(n^E)$ gilt. Also gilt mit dem MASTER-THEOREM (V);

$$T(n) \in \Theta(E \cdot \log n) \tag{IV}$$
$$\overset{E=n}{\Rightarrow} T(n) \in \Theta(n \cdot \log n) \tag{V}$$

QED

b)

Ja der gegebene Algorithmus ist stabil, da lediglich Zahlen getauscht werden, die echt kleiner beziehungsweise echt größer sind.

Georg C. Dorndorf Matr.Nr. 366511
Adrian C. Hinrichs Matr.Nr. 367129

c)

Sei das zu sortierende Array modelliert als n-Tupel a mit $a_i, n, i \in \mathbb{N}$ Der triviale Fall wäre $n = 1$, für welchen offensichtlich gilt dass das Array sortiert ist. Nun beweisen wir Induktiv, dass der Algorithmus für alle $n \in \mathbb{N}$ korrekt sortiert. Vorraussetzung ist die Korrektheit der `merge()`-Funktion.

$$(IA) : n = 1 : \tag{VI}$$

ist trivialerweise sortiert. Der Algorithmus terminiert direkt. ■

$$n = 2 : \tag{VII}$$

wird in der else-clause offensichtlich richtig sortiert, da nach ihrer Ausführung gilt:

$$a_1 \leq a_2 \tag{VIII}$$
$$\Leftrightarrow \forall i \in \mathbb{N} \text{ mit } i < n : a_i \leq a_{i+1} \tag{IX}$$

■

(IV) : Die Behauptung gelte nun für ein festes aber beliebiges $n \in \mathbb{N}$.

$$(IS) : \; n \mapsto n + 1 \tag{X}$$
$$\tag{XI}$$

Timmis Merge-Sort teilt Arrays, die länger als 2 sind in zwei Teile. Daraus folgt folgender Induktionsschritt:

$$\forall i \in \mathbb{N} \text{ mit } i < n : a_i \leq a_{i+1} \tag{XII}$$

Seien l, r die Indizes auf, die Mergesort in der aktuellen Rekursionstiefe angewendet wird. Seien drei $\lceil \frac{n}{3} \rceil$-Tupel b, c, d und k wie folgt definiert:

$$k \coloneqq \lceil \frac{n}{3} \rceil \tag{XIII}$$
$$b_i \coloneqq a_i \qquad \forall i \in \mathbb{N} \text{ mit } i < k \tag{XIV}$$
$$c_i \coloneqq a_i \qquad \forall i \in \mathbb{N} \text{ mit } l + k < i < 2k \tag{XV}$$
$$d_i \coloneqq a_i \qquad \forall i \in \mathbb{N} \text{ mit } 2k < i < r \tag{XVI}$$

Dann gilt für das $n+1$-Tupel a:

$$a = \text{merge}(\text{merge}(\text{sort}(b), \text{sort}(c)), \text{sort(d)}) \tag{XVII}$$
$$\text{mit } a'_i = a_i \forall i \leq n \text{ und } a''_1 = a_{n+1} \tag{XVIII}$$

nach Induktionsvorraussetzung werde b, c, d richtig sortiert. Nach Vorraussetzung verhält sich die merge-Funktion korrekt. Somit ist die Korrektheit Timmis Merge-Sort Algorithmus mittels vollständiger Induktion für alle $n \in \mathbb{N}$ bewiesen.

QED

Aufgabe 8

a)

Zu zeigen: Ein heap mit n Knoten hat die Höhe $\lfloor \log_2 n \rfloor$.

Beweis: Es ist bekannt, dass ein Heap ein Binärbaum ist. Für Binärbäume mit n Knoten gilt, dass die Höhe *mindestens* $\lceil \log_2(n+1) \rceil - 1$ ist. Die minimale Form eines Binärbaums ist erreicht, wenn alle Ebenen, bis auf die letzte komplett gefüllt sind, (was nach VL-07, Folie 5 eine Bedingung für einen Heap ist) ein Heap ist also ein minimaler binärer Baum. Zu zeigen: Die höhe eines minimalen binären Baums ist $\lfloor \log_2(n) \rfloor$

Fall 1: $\log_2(n+1) \in \mathbb{R} \backslash \mathbb{N}$, so gilt:

$$\lceil \log_2(n+1) \rceil - 1 = \lfloor \log_2(n+1) \rfloor \tag{XIX}$$
$$\stackrel{\star}{=} \lfloor \log_2(n) \rfloor \tag{XX}$$

□

$\star$: $\log_2(n+1) \in \mathbb{R} \backslash \mathbb{N}$ und $\log_2(n+1) - \log_2 n \in]0, 1[\; \forall n \in \mathbb{N}$

Fall 2: $\log_2(n+1) \in \mathbb{N}$, so gilt:

$$\log_2(n+1) \in \mathbb{N} \text{ und } \log_2(n) \in \mathcal{O}(n) \tag{XXI}$$
$$\Rightarrow \exists \epsilon \in]0, 1[: \log_2(n) + \epsilon = \log_2(n+1) \tag{XXII}$$
$$\Rightarrow \lceil \log_2(n+1) \rceil - 1 \overset{\substack{\log_2(n+1) \in \mathbb{N} \\ 0 < \epsilon < 1}}{=} \lceil \log_2(n+1) - \epsilon \rceil - 1 \tag{XXIII}$$
$$\overset{(XXII)}{=} \lceil (\log_2(n) + \epsilon) - \epsilon \rceil - 1 \tag{XXIV}$$
$$= \lceil \log_2(n) \rceil - 1 \tag{XXV}$$
$$\overset{\log_2 n \notin \mathbb{N}}{=} \lfloor \log_2 n \rfloor \tag{XXVI}$$

□

Es wurde also gezeigt, das die höhe eines minimalen binären Baums, und somit auch die eines Heaps genau $\lfloor \log_2(n) \rfloor$ ist.

QED

b)

Zu zeigen: Ein Heap hat maximal $\lceil \frac{n}{2^{h+1}} \rceil$ Knoten mit der Höhe h.

Beweis: Betrachten wir einen beliebigen Heap. Sei die Höhe des Heaps (aslo die Anzahl der Ebenen) als H gegeben, die Anzahl der Knoten als n.
Die maximale Anzahl an Knoten, die e Kanten von der Wurzel entfernt sind, beläuft sich nun auf 2^e (nach Vorlesung)

Fall 1: $H = 1 \Leftrightarrow n = 1$
Die behauptung gilt Trivialerweise.

□

Fall 2: $H > 1 \Leftrightarrow n > 1$
Es gilt für die höhe jedes Knotens: $h = (H-1) - e$ (die Wurzel befindet sich auf Ebene 1).
Sei N die maximale anzahl an Knoten eines Heaps der höhe H.
Die maximale Anzahl der Knoten der höhe h ist also:

$$2^{(H-1)-h} = 2^{H-(h+1)} \tag{XXVII}$$
$$= \frac{2^H}{2^{h+1}} \tag{XXVIII}$$
$$= \frac{2^H}{2^{h+1}} \tag{XXIX}$$
$$\overset{*}{=} \frac{N+1}{2^{h+1}} \tag{XXX}$$
$$\overset{**}{=} \left\lceil \frac{n}{2^{h+1}} \right\rceil \tag{XXXI}$$
$$\tag{XXXII}$$

*:

$$8\text{a} \Rightarrow H = \lfloor \log_2(N+1) \rfloor$$

da $h \in \mathbb{N}$ folgt $\log_2(N+1) \in \mathbb{N}$, also:

$$\begin{aligned} &= \log_2(N+1) \\ \Leftrightarrow 2^H &= 2^{\log_2(N+1)} \\ &= N+1 \end{aligned}$$

■

** : Das tatsächliche n muss offensichtlich im Intervall $2^{H-1} < n \geq 2^H - 1 = N$ liegen, da ansonsten nicht genügend Knoten im Heap enthalten wären, um H ebene zu besitzen. Des weiteren ist n somit immer größer als 2^{h+1} ■

Die behauptung ist also Bewiesen

QED