technische universität dortmund

SoSe 2018

M. Buchin
A. Krivošija, D. Mezlaf
L. Pfahler, A. Rey

DAP2 – Heimübung 4

Ausgabedatum: 27.04.2018 — Abgabedatum: Mo. 07.05.2018 bis 12 Uhr

Abgabe:

Schreiben Sie unbedingt immer Ihren vollständigen Namen, Ihre Matrikelnummer und Ihre Gruppennummer auf Ihre Abgaben! Beweise sind nur dort notwendig, wo explizit danach gefragt wird. Eine Begründung der Antwort wird allerdings *immer* verlangt.

Aufgabe 4.1 (6 Punkte): (Rekursionsgleichung)

Gegeben seien die Rekursionsgleichungen:

a)
$$T(n) = \begin{cases} 1 & \text{für } n = 1 \\ 8 \cdot T\left(\frac{n}{2}\right) + n^2\sqrt{n} & \text{für } n > 1 \end{cases}$$

b)
$$T(n) = \begin{cases} 1 & \text{für } n = 1 \\ 3 \cdot T\left(\frac{n}{9}\right) + 4n & \text{für } n > 1 \end{cases}$$

c)
$$T(n) = \begin{cases} 1 & \text{für } n = 1 \\ 16 \cdot T\left(\frac{n}{4}\right) + 2n^2 & \text{für } n > 1 \end{cases}$$

Bestimmen Sie jeweils eine asymptotische obere Schranke für $T(n)$ und beweisen Sie diese mittels vollständiger Induktion. Sie dürfen der Einfachheit halber annehmen, dass n von der Form 2^k, 9^k bzw. 4^k für ein $k \in \mathbb{N}$ ist.

Lösung:

a) Wir verwenden das Master-Theorem mit $f(n) = n^2\sqrt{n}$, setzen also $\gamma \cdot n^2\sqrt{n} = 8 \cdot \left(\frac{n}{2}\right)^2 \cdot \sqrt{\frac{n}{2}}$. Äquivalent dazu ist $\gamma = \sqrt{2}$ und dadurch $T(n) \in O(n^{\log_2 8}) = O(n^3)$. Die genauen Konstanten in dieser Teilaufgabe werden wir mittels **Teleskopsummenmethode** finden. Warum diese Methode funktioniert, wurde in der Lösung von Präsenzblatt 3 erklärt. Wir führen sie hier einmal an unserem Beispiel durch.

Wir nehmen an, dass n eine Zweierpotenz ist, und gucken uns den Wert der Funktion T jetzt für Zweierpotenzen an ($n = 2^k$). Die Rekursionsformel sieht dann so aus:

$$\begin{aligned} & T(2^k) = 8 \cdot T(2^{k-1}) + (2^k)^2\sqrt{2^k} \\ \Leftrightarrow \quad & T(2^k) = 8 \cdot T(2^{k-1}) + 2^{2k}\sqrt{2^k} \end{aligned}$$

Jetzt teilen wir die Gleichung auf beiden Seiten durch $8^k = 2^{3k}$ und erhalten als äquivalente Aussage:

$$\begin{aligned} & T(2^k) = 8 \cdot T(2^{k-1}) + 2^{2k}\sqrt{2^k} \\ \Leftrightarrow \quad & \frac{T(2^k)}{8^k} = \frac{8 \cdot T(2^{k-1})}{8^k} + \frac{2^{2k}\sqrt{2^k}}{2^{3k}} \\ \Leftrightarrow \quad & \frac{T(2^k)}{8^k} = \frac{8 \cdot T(2^{k-1})}{8^k} + \left(\frac{1}{\sqrt{2}}\right)^k \end{aligned}$$

Nun fassen wir die Terme $\frac{T(2^k)}{8^k}$ und $\frac{T(2^{k-1})}{8^{k-1}}$ als das k-te und $(k-1)$-te Element der gleichen Folge $(S_i)_{i\in\mathbb{N}}$ auf, die wir für $i \geq 0$ durch

$$S(i) = \frac{T(2^i)}{8^i},$$

definieren. Unsere Gleichung gibt uns nun die Differenz zweier aufeinanderfolgenden Folgeglieder:

$$\frac{T(2^k)}{8^k} - \frac{T(2^{k-1})}{8^{k-1}} = \left(\frac{1}{\sqrt{2}}\right)^k$$

Für Folgen mit dieser Eigenschaft kann man zeigen, dass man $S(k)$ errechnen kann, indem man die Abstände aufsummiert. Der Beweis dazu steht in der Lösung des 3. Präsenzblatts. Genauer gilt

$$S(i) = S(0) + \sum_{k=1}^{i} [S(k) - S(k-1)] \quad .$$

Es gilt $S(0) = T(2^0)/8^0 = 1/1 = 1$. Wir erhalten also

$$S(i) = 1 + \sum_{k=1}^{i} [S(k) - S(k-1)] = 1 + \sum_{k=1}^{i} \left(\frac{1}{\sqrt{2}}\right)^k = \sum_{k=0}^{i} \left(\frac{1}{\sqrt{2}}\right)^k = \frac{1 - \frac{1}{\sqrt{2}^{i+1}}}{1 - \frac{1}{\sqrt{2}}} \quad .$$

Wir haben $S(i)$ ausgerechnet, interessieren uns aber eigentlich für $T(2^i) = 2^{3i} \cdot S(i)$. Also rechnen wir jetzt:

$$T(2^i) = 2^{3i}S(i) = 2^{3i} \cdot \frac{1 - \frac{1}{\sqrt{2}^{i+1}}}{1 - \frac{1}{\sqrt{2}}} = \frac{2^{3i} - \frac{2^{2i}\sqrt{2}^i}{\sqrt{2}}}{\frac{\sqrt{2}-1}{\sqrt{2}}} = \frac{2^{3i}\sqrt{2} - 2^{2i}\sqrt{2}^i}{\sqrt{2}-1} \cdot \frac{\sqrt{2}+1}{\sqrt{2}+1}$$

Der Nenner ist gleich $(\sqrt{2}-1) \cdot (\sqrt{2}+1) = 2 - 1 = 1$. Für $T(n) = T(2^{\log_2 n})$ ergibt das also

$$\begin{aligned} T(n) &= \left(2^{3\log_2 n}\sqrt{2} - 2^{2\log_2 n}\sqrt{2}^{\log_2 n}\right) \cdot \left(\sqrt{2}+1\right) \\ &= \left(n^3\sqrt{2} - n^2\sqrt{n}\right) \cdot \left(\sqrt{2}+1\right) \\ &= (2+\sqrt{2}) \cdot n^3 - (1+\sqrt{2}) \cdot n^2\sqrt{n} \end{aligned}$$

Diese Behauptung beweisen wir nun noch mittels Induktion:

Behauptung: $T(n) = (2+\sqrt{2}) \cdot n^3 - (1+\sqrt{2}) \cdot n^2\sqrt{n} = O(n^3)$ (wir nehmen an, dass n eine 2er Potenz ist).

Beweis:

(I.A.) Für $n = 1$ ist $T(n) = 1 = (2+\sqrt{2}) \cdot 1^3 - (1+\sqrt{2}) \cdot 1^2\sqrt{1} = (2+\sqrt{2}) - (1+\sqrt{2}) = 1$.

(I.V.) Die Behauptung gelte für ein beliebiges, aber festes $n_0 = 2^{k_0}$ (n_0 ist eine Zweierpotenz).

(I.S.) Betrachte $n = 2n_0$ (die nächste 2er Potenz). Dann gilt

$$\begin{aligned} T(n) &= 8 \cdot T\left(\frac{2n_0}{2}\right) + n^2\sqrt{n} = 8 \cdot T(n_0) + n^2\sqrt{n} \\ &\overset{\text{(IV)}}{=} 8 \cdot \left((2+\sqrt{2}) \cdot n_0^3 - (1+\sqrt{2}) \cdot n_0^2\sqrt{n_0}\right) + n^2\sqrt{n} \\ &= (2+\sqrt{2}) \cdot (2n_0)^3 - \sqrt{2} \cdot (1+\sqrt{2}) \cdot (2n_0)^2\sqrt{2n_0} + n^2\sqrt{n} \\ &= (2+\sqrt{2}) \cdot n^3 - (\sqrt{2}+2) \cdot n^2\sqrt{n} + n^2\sqrt{n} \\ &= (2+\sqrt{2}) \cdot n^3 - (1+\sqrt{2}) \cdot n^2\sqrt{n}. \end{aligned}$$

Damit folgt die Behauptung für alle natürliche Zahlen (der Form $n = 2^k$, $k \in \mathbb{N}_0$).

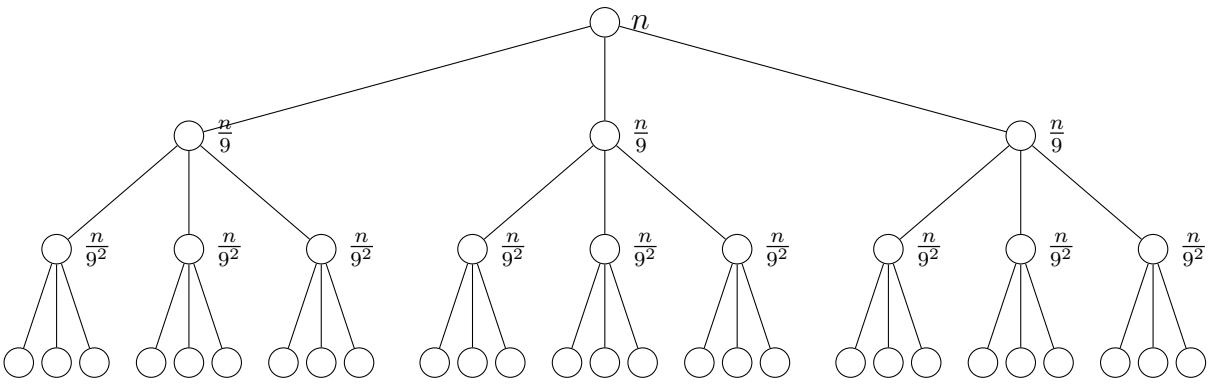
b) Wir verwenden das Master-Theorem mit $f(n) = 4n$, setzen also $\gamma \cdot 4n = 4 \cdot \frac{n}{4}$. Äquivalent dazu ist $\gamma = \frac{1}{4}$ und dadurch $T(n) \in O(f(n)) = O(4n) = O(n)$.

Die genauen Konstanten in dieser Teilaufgabe werden wir mittels **Baummethode** finden. Laufzeiten der Form $T(n) = a \cdot T(\frac{n}{b}) + f(n)$ entstehen, wenn ein rekursiver Algorithmus

- a rekursive Aufrufe macht,
- die rekursiven Aufrufe eine Eingabe der Länge n/b erhalten,
- zudem eine zusätzliche Laufzeit von $f(n)$ benötigt, und
- einen Rekursionsabbruch hat, der lediglich eine konstante Laufzeit benötigt.

Hier ist $a = 3$, $b = 9$ und $f(n) = 4n$.

Die 'zusätzliche Laufzeit' sowie die konstante Laufzeit im Rekursionsabbruch sind eigentlich die einzige 'tatsächliche Arbeit', die anfällt. Wir müssen sie nun geschickt für die rekursiven Aufrufe aufsummieren. Dazu gucken wir uns den Rekursionsbaum anhand von unserem Beispiel an. Hier gibt es immer drei rekursive Aufrufe, da $a = 3$ ist. Außerdem reduziert sich die Eingabelänge auf ein Neuntel, da $b = 9$ ist. Der Baum sieht also so aus:



Neben den Knoten steht, wie viel von den Eingabedaten in diesem Knoten ankommt. Wenn in einem Knoten $n/9^2$ Daten sind, dauert die 'zusätzliche' Laufzeit natürlich $f(n/9^2)$ statt $f(n)$. In unserem Beispiel wäre das also $f(n) = 4n$ im Wurzelknoten, $f(n/9) = 4 \cdot \frac{n}{9}$ auf der zweiten Ebene, $f(n/9^2) = 4 \cdot \frac{n}{9^2}$ auf der dritten Ebene und so weiter.

Jetzt müssen wir noch zählen, wie viele Knoten auf einer Ebene eigentlich sind. Auf der ersten Ebene ist nur ein Knoten (die Wurzel) mit Kosten von $4n$. Dies sind also auch die Kosten der ersten Ebene. Auf der zweiten Ebene sind drei Knoten, jeder davon kostet $4 \cdot (n/9)$. Diese Ebene ergibt also Kosten von $3 \cdot 4 \cdot (n/9) = 4 \cdot (n/3)$. Auf der dritten Ebene sind neun Knoten, die jeweils $4 \cdot (n/9^2)$ kosten, zusammen macht das $4 \cdot (n/3^2)$.

Unser Baum hat (wenn n eine Neunerpotenz ist) $\log_9 n + 1$ Ebenen, da man n genau $\log_9 n$ mal durch neun teilen muss, bis man 1 erhält, d.h. bis in jedem Knoten nur noch ein Element verarbeitet wird (und weil man zusätzlich die erste Ebene mit dem Wurzelknoten hat).

Wir sollen jetzt das Muster erkennen. Auf Ebene i sind nämlich 3^{i-1} Knoten, und jeder davon kostet $4 \cdot (n/9^{i-1})$, zusammen kosten alle Knoten auf Ebene i also $4 \cdot (n/3^{i-1})$. Das stimmt hier nicht auf der untersten Ebene $(1+\log_9 n)$. Dort haben wir Laufzeit 1 in jedem Knoten, da $T(1) = 1$ ist, und wir haben $3^{\log_9 n+1-1} = \sqrt{n}$ viele Knoten auf der untersten Ebene. Die Kosten auf der Ebene der Blätter beträgen also $\sqrt{n}$.

Jetzt tragen wir die Kosten zusammen. Dafür müssen wir $4 \cdot (n/3^{i-1})$, angefangen von $i = 1$ bis zu $i = \log_9 n$, aufsummieren, plus $\sqrt{n}$ für die Blätter. Wir erhalten:

$$\sqrt{n} + \sum_{i=1}^{\log_9 n} 4n \cdot \frac{1}{3^{i-1}} = \sqrt{n} + 4n \cdot \sum_{i=0}^{\log_9 n - 1} \frac{1}{3^i} = \sqrt{n} + 4n \cdot \frac{1 - \frac{1}{3^{\log_9 n}}}{1 - \frac{1}{3}}$$

$$= \sqrt{n} + 4n \cdot \frac{1 - \frac{1}{\sqrt{n}}}{\frac{2}{3}} = \sqrt{n} + 6n \cdot \left(1 - \frac{1}{\sqrt{n}}\right) = 6n - 5\sqrt{n}.$$

Die bisherige Analyse ist eher ein intuitives, schlaues Erraten der Lösung. Die Behauptung ist mittels Induktion zu beweisen:

Behauptung: $T(n) = 6n - 5\sqrt{n} = O(n)$ (wir nehmen an, dass n eine 9er-Potenz ist).

Beweis:

(I.A.) Für $n = 1$ ist $T(n) = 1 = 6 \cdot 1 - 5\sqrt{1} = 6 - 5 = 1$.

(I.V.) Die Behauptung gelte für ein beliebiges, aber festes $n_0 = 9^{k_0}$ (n_0 ist eine 9er-Potenz).

(I.S.) Betrachte $n = 9 \cdot n_0$ (die nächste 9er Potenz). Dann gilt

$$\begin{aligned} T(n) &= 3 \cdot T\left(\frac{9 \cdot n_0}{9}\right) + 4n = 3 \cdot T(n_0) + 4n \\ &\overset{\text{(IV)}}{=} 3 \cdot (6n_0 - 5\sqrt{n_0}) + 4n = 2 \cdot 3 \cdot 3 \cdot n_0 - 3 \cdot 5\sqrt{n_0} + 4n \\ &= 2 \cdot 9n_0 - 5\sqrt{9n_0} + 4n = 2n - 5\sqrt{n} + 4n \\ &= 6n - 5\sqrt{n} \end{aligned}$$

Damit folgt die Behauptung für alle natürliche Zahlen n (der Form $n = 9^k$, $k \in \mathbb{N}_0$).

c) Wir verwenden das Master-Theorem mit $f(n) = 2n^2$, setzen also $\gamma \cdot 2n^2 = 16 \cdot 2 \cdot \left(\frac{n}{4}\right)^2$. Äquivalent dazu ist $\gamma = 1$ und dadurch $T(n) \in O(n^2 \log n)$. Da wir aber keine Konstante für diese Behauptung kennen, sollten wir genaueren Ausdrück herleiten. Diese Teilaufgabe werden wir mittels **Einsetzmethode** lösen.

$$\begin{aligned}
T(n) &= 16 \cdot T\left(\frac{n}{4}\right) + 2n^2 \\
&= 16 \cdot \left(16 \cdot T\left(\frac{n}{4^2}\right) + 2 \cdot \frac{n^2}{4^2}\right) + 2n^2 = 16^2 \cdot T\left(\frac{n}{4^2}\right) + 2n^2 + 2n^2 \\
&= 16^2 \cdot \left(16 \cdot T\left(\frac{n}{4^3}\right) + 2 \cdot \left(\frac{n}{4^2}\right)^2\right) + 2n^2 + 2n^2 \\
&= 16^3 \cdot T\left(\frac{n}{4^3}\right) + 2n^2 + 2n^2 + 2n^2 \\
&= \ldots = 16^k \cdot T\left(\frac{n}{4^k}\right) + 2k \cdot n^2
\end{aligned}$$

für alle k, sodass $n > 4^k$. Wenn $k = \log_4 n$, gilt weiter:

$$\begin{aligned}
T(n) &= 16^{\log_4 n} \cdot T(1) + 2n^2 \cdot \log_4 n = n^2 \cdot 1 + 2n^2 \cdot \log_4 n \\
&= 2n^2 \cdot \log_4 n + n^2 = O(n^2 \cdot \log n)
\end{aligned}$$

Auch hier ist die bisherige Analyse eher ein intuitives, schlaues Erraten der Lösung. Die Behauptung ist mittels Induktion zu beweisen:

Behauptung: $T(n) = 2n^2 \log_4 n + n^2 = O(n^2 \log n)$ (wir nehmen an, dass n eine 4er Potenz ist).

Beweis:

(I.A.) Für $n = 1$ ist $T(n) = 1 = 2 \cdot 1^2 \cdot \log_4 1 + 1^2 = 2 \cdot 0 + 1 = 1$.

(I.V.) Die Behauptung gelte für ein beliebiges, aber festes $n_0 = 4^{k_0}$ (n_0 ist eine Vierer-potenz).

(I.S.) Betrachte $n = 4 \cdot n_0$ (die nächste 4er Potenz). Dann gilt

$$\begin{aligned}
T(n) &= 16 \cdot T\left(\frac{4 \cdot n_0}{4}\right) + 2n^2 = 16 \cdot T(n_0) + 2n^2 \\
&\overset{\text{(IV)}}{=} 16 \cdot \left(2n_0^2 \cdot \log_4 n_0 + n_0^2\right) + 2n^2 = 2(4n_0)^2 \cdot \log_4 n_0 + (4n_0)^2 + 2n^2 \cdot \log_4 4 \\
&= 2n^2 \cdot (\log_4 n_0 + \log_4 4) + n^2 = 2n^2 \cdot \log_4 4n_0 + n^2 \\
&= 2n^2 \cdot \log_4 n + n^2
\end{aligned}$$

Damit folgt die Behauptung für alle natürliche Zahlen n (der Form $n = 4^k$, $k \in \mathbb{N}_0$).

Hinweise: Für Fälle, in denen die Aufteilung genau gleich viele Kosten verursacht wie die Kosten beim Zusammenfügen (Teilaufgabe 1), lässt sich der Induktionsbeweis meist ohne größeren Aufwand durchführen. In den Fällen der Teilaufgaben 2 und 3, in denen die Kosten des Zusammenfügens echt geringer sind als die Kosten der Aufteilung, muss man auf die Wahl der Konstanten achten. Hier ist es allerdings in der Regel möglich, eine gleiche Umformung der rekursiven Form anzugeben, wie das hier auch bei allen Teilaufgaben geschehen ist. Wenn die Umformung keine Abschätzung enthält, so lässt sich mit dem Ergebnis direkt der Induktions-beweis führen.

□

Aufgabe 4.2 (4 Punkte): (Merge-Operation und Schleifeninvariante)

Im Folgenden ist als Pseudocode eine mögliche Realisierung der in der Vorlesung verwendeten Funktion Merge(A, p, q, r) gegeben. Diese Funktion fügt die aufsteigend sortierten Teilarrays $A[p \dots q]$ und $A[(q+1) \dots r]$ $(1 \leq p \leq q < r \leq \text{length}[A])$ von A zum sortierten Teilarray $A[p \dots r]$ zusammen.

Merge(A, p, q, r):

```
1  B ← new Array [1 ... (r − p + 1)]
2  i ← p
3  j ← q + 1
4  k ← 1
5  while k ≤ r − p + 1 do
6      if i ≤ q and (j > r or A[i] ≤ A[j]) then
7          B[k] ← A[i]
8          i ← i + 1
9      else
10         B[k] ← A[j]
11         j ← j + 1
12     k ← k + 1
13 A[p ... r] ← B[1 ... (r − p + 1)]
```

Zeigen Sie die Korrektheit des gegebenen Programms Merge. Formulieren Sie dazu zunächst eine Invariante für die While-Schleife in Zeile 5, und beweisen Sie ihre Korrektheit mittels vollständiger Induktion. Verwenden Sie dann Ihre Invariante, um die Korrektheit des Programms zu folgern. Sie dürfen annehmen, dass die Anweisung in Zeile 13 den Inhalt des Arrays $B[1 \dots (r-p+1)]$ elementweise korrekt in das Teilarray $A[p \dots r]$ kopiert.

Lösung:

Es ist vorausgesetzt, dass $A[p \dots q]$ und $A[q+1 \dots r]$ sortiert sind (wir setzen hier immer aufsteigende Sortierung voraus) und ferner $p \leq q$ sowie $q+1 \leq r$ gilt.

Wir behaupten, dass folgende Invariante $Inv(k)$ für $k \in \{1, \dots, r-p+2\}$ vor dem k-ten Schleifeneintritt gilt:

i) $p \leq i \leq q+1$;

ii) $q+1 \leq j \leq r+1$;

iii) $(i-p)+(j-q-1) = k-1$;

iv) $(k > 1 \text{ und } i \leq q) \Rightarrow A[i] \geq B[k-1]$;

v) $(k > 1 \text{ und } j \leq r) \Rightarrow A[j] \geq B[k-1]$;

vi) das Teilarray $B[1..k-1]$ ist korrekt aufsteigend sortiert.

Wir beweisen unsere Invariante mittels vollständiger Induktion über k. Jede der sechs Teilaussagen ist dabei im Induktionsanfang sowie im Induktionsschritt zu beachten.

IA. Für $k = 1$ wurde die While-Schleife noch nicht betreten. In den Zeilen 2, 3 und 4 sind $i \leftarrow p$, $j \leftarrow q + 1$ und $k \leftarrow 1$ initialisiert. Damit sind die Aussagen i), ii) und iii) der Invariante erfüllt. Da $k = 1$ ist, sind die Voraussetzungen der Bedingungen in Aussagen iv) und v) falsch, daher sind die Implikationen wahr, d. h. iv) und v) sind auch erfüllt. Das Teilarray $B[1..0] = \emptyset$ ist ein leeres Array und als solches korrekt sortiert (vi)). Damit ist die Invariante für $k = 1$ korrekt.

IV. Es gelte die Invariante für beliebiges aber festes k mit $1 \leq k \leq r - p + 1$.

IS. Sei $k > 1$. Wir wollen zeigen, dass die Invariante auch für $k+1$, also nach der k-ten Iteration der Schleife, erfüllt ist. Da $k \leq r - p + 1$ ist, ist die Eintrittsbedingung der While-Schleife wahr und die Schleife wird betreten. Es sind nach I.V. Teil vi) bisher $k - 1$ Elemente im Array B korrekt sortiert.

Aus I.V. (Teile i), ii) und iii)) folgt, dass diese drei Aussagen auch nach der k-ten Iteration (bzw. vor $k + 1$-ter Iteration) weiterhin gelten werden, da i nur bis $q + 1$ (If-Bedingung in der Zeile 6) bzw. j nur bis $r + 1$ erhöht werden können. Beide Seiten der Gleichung in iii) werden in einer Iteration genau um 1 erhöht (Zeile 8/11 und 12).

Es gilt $k > 1$ (es gibt also mindestens ein Element in B) Wir unterscheiden zwei Fälle: Der Eintrag $B[k]$ wird entweder auf $A[i]$ (Zeile 7) oder auf $A[j]$ (Zeile 10) gesetzt. Der Programmfluss gibt vor, dass genau einer dieser beiden Fälle in der k-ten Iteration eintritt.

Fall 1: $B[k] \leftarrow A[i]$. Dann gelten $i \leq q$ und entweder $j > r$ oder $A[i] \leq A[j]$. Der Invariantenteil iv) wird wie folgt gezeigt:

- Falls $i = q$ vor der Iteration, gilt nun $i = q+1$ und damit Aussage iv) $((\bot \Rightarrow p) \Leftrightarrow \top$, für alle p).
- Falls $i < q$ vor der Iteration, gelten anschließend noch $i \leq q$ und $B[k] = A[i] \leq A[i + 1]$, weil $A[p..q]$ aufsteigend sortiert ist. Somit ist iv) korrekt nach der k-ten, bzw. vor der $k + 1$-ten Iteration.

Den Teil v) zeigen wir ähnlich:

- Falls $j > r$ vor der Iteration, folgt die Korrektheit von v) $((\bot \Rightarrow p) \Leftrightarrow \top)$.
- Falls $j \leq r$, gilt $B[k] = A[i] \leq A[j]$ ($A[i]$ wurde anstelle von $A[j]$ gewählt). Auch dann ist v) korrekt nach der k-ten, bzw. vor der $k + 1$-ten Iteration.

Nach der I.V.-Teil iv) ist $B[k - 1] \leq A[i]$ und durch die Zuweisung gilt $B[k] = A[i]$, zusammen also $B[k - 1] \leq A[i] = B[k]$. Zusammen mit I.V.-Teil vi) erhalten wir, dass $B[1..k]$ aufsteigend sortiert ist, die letzte zu zeigende Aussage unserer Invariante.

Der zweite Fall ($B[k] \leftarrow A[j]$) wird analog analysiert. Somit gilt die Invariante auch nach der k-ten, bzw. vor der $k + 1$-ten Schleifeniteration.

Unsere Invariante gilt dementsprechend für alle $1 \leq k \leq r - p + 2$. Wenn $k = r - p + 2$ ist, wird die Schleife verlassen. Nach dem Invariantenteil vi) ist das Array $B[1..r - p + 1]$ korrekt sortiert. Dieses Array enthält alle Elemente von $A[p..r]$ und das sortierte Array wird in der Zeile 13 in A überschrieben. Somit arbeitet die Funktion *Merge(A,p,q,r)* korrekt und das Array $A[p..r]$ ist korrekt aufsteigend sortiert.