

Kapitel 1

Divide-and-Conquer Verfahren und das Prinzip der Rekursion

Der Literaturtip. Ein gutes Buch unter vielen zum Gesamtthema der Vorlesung ist [CLR90]. Es enthält auch einführende Kapitel zum Algorithmusbegriff sowie zur Laufzeitanalyse von Algorithmen und zu Wachstumsraten von Funktionen (O , Ω , Θ). Ein weiteres empfehlenswertes Buch ist [OW93]. Für das Kapitel „Divide-and-Conquer Verfahren und das Prinzip der Rekursion“ speziell empfehlen wir [CLR90].

Ein grundlegendes algorithmisches Prinzip besteht darin, ein Problem zu lösen, indem man es in Probleme (meist desselben Typs) kleinerer Größe oder mit kleineren Werten aufteilt, diese löst und aus den Lösungen eine Lösung für das Ausgangsproblem konstruiert.

1.1 Ein Beispiel aus der Informatik-Grundvorlesung

MERGE SORT ist ein Verfahren zum Sortieren einer Folge von n Werten, auf denen es eine Ordnung \leq gibt. MERGE SORT sortiert eine Folge der Länge n , indem es zunächst halb so lange Teilfolgen sortiert und aus diesen die sortierte Folge der Länge n „zusammenmischt“. Dies kann, wie im Folgenden beschrieben, auf iterative oder auf rekursive Weise durchgeführt werden.

1.1.1 Iterativ bzw. bottom-up

Iteratives Merge-Sort

1. sortiere zunächst Teilfolgen der Länge zwei und mische sie zu sortierten Folgen der Länge vier zusammen;
2. mische je zwei sortierte Folgen der Länge vier zu sortierten Folgen der Länge acht zusammen;
3. u.s.w.

1.1.2 Rekursiv

Sehr oft werden Divide-and-Conquer Verfahren jedoch rekursiv angewandt. Das bedeutet: Das Verfahren ruft sich selbst wieder auf, angewandt auf Eingaben kleinerer Länge bzw. mit kleineren Werten.

MERGE SORT rekursiv formal.

Eingabe: Array A mit n Elementen, die an den Stellen $A[p]$ bis $A[r]$ stehen.

Ausgabe: n Elemente sortiert in Ergebnisarray B .

MERGE SORT($A; p, r$)

1. Falls $p < r$ gilt, dann setze $q := \lfloor \frac{p+r}{2} \rfloor$
2. $\left. \begin{array}{l} \text{MERGE SORT}(A; p, q) \\ \text{MERGE SORT}(A; q+1, r) \end{array} \right\}$ rekursive Aufrufe
3. $B := \text{MERGE}(A; p, q, r)$.

Die Hauptarbeit besteht hier bei MERGE SORT, wie auch bei den meisten Divide-and-Conquer Verfahren, im „Zusammensetzen“ der Teillösungen, bei MERGE SORT also in MERGE.

MERGE informell: Durchlaufe A einerseits von $A[p]$ bis $A[q]$ und andererseits von $A[q+1]$ bis $A[r]$ und vergleiche die Elemente jeweils paarweise. Das kleinere der Elemente wird „weggeschrieben“ (in ein Ergebnisarray) und in dem entsprechenden Teil von A um eine Position weitergegangen. Ist man bei $A[q]$ oder $A[r]$ angelangt, wird der restliche Teil des anderen Teilarrays von A an das Ergebnisarray angehängt. Die Einträge aus dem Ergebnisarray werden an die Stellen $A[p]$ bis $A[q]$ kopiert.

Wir haben gezeigt, daß MERGE eine Laufzeit hat, die linear in der Länge der Eingabe ist, also in $\mathcal{O}(n)$ ist. Aus der rekursiven Beschreibung für MERGE SORT können wir als Laufzeit $T(n)$ insgesamt ablesen:

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \underbrace{k \cdot n}_{\text{für MERGE}} \quad \text{mit } k \geq 0 \text{ Konstante.}$$

Solche Rekursionsgleichungen sind typisch für rekursiv beschriebene Divide-and-Conquer-Algorithmen. Wie schätzt man eine Laufzeitfunktion, die als Rekursionsgleichung (bzw. -ungleichung) gegeben ist, möglichst gut ab?

Zur Erinnerung: Bei MERGE SORT ist $T(n) \in \mathcal{O}(n \cdot \log n)$. Der Beweis wird induktiv geführt, vgl. Informatik II: Algorithmen und Datenstrukturen.

1.2 Methoden zur Analyse von Rekursionsabschätzungen

Substitutionsmethode: Wir vermuten eine Lösung und beweisen deren Korrektheit induktiv.

Iterationsmethode: Die Rekursionsabschätzung wird in eine Summe umgewandelt, und dann mittels Techniken zur Abschätzung von Summen aufgelöst.

Meistermethode: Man beweist einen allgemeinen Satz zur Abschätzung von rekursiven Ausdrücken der Form

$$T(n) = a \cdot T(n/b) + f(n), \text{ wobei } a \geq 1 \text{ und } b > 1.$$

Technische Details. Normalerweise ist die Laufzeitfunktion eines Algorithmus nur für ganze Zahlen definiert. Entsprechend steht in einer Rekursionsabschätzung eigentlich $T(\lfloor n/b \rfloor)$ oder $T(\lceil n/b \rceil)$. Außerdem haben Laufzeitfunktionen $T(n)$ die Eigenschaft, daß zwar $T(n) \in \mathcal{O}(1)$ ist für kleine n , allerdings oft mit großer \mathcal{O} -Konstante. Meistens kann man diese Details jedoch vernachlässigen.

1.2.1 Die Substitutionsmethode

Beispiel.

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n.$$

Dies ist etwa die Laufzeitfunktion von MERGE SORT, von der wir wissen, daß sie in $\mathcal{O}(n \log n)$ liegt.

Wir beweisen, daß $T(n) \leq c \cdot n \cdot \log n$ für geeignetes $c > 0$, c konstant ist. Dazu nehmen wir als Induktionsvoraussetzung an, daß die Abschätzung für Werte kleiner n gilt, also insbesondere

$$\begin{aligned} T(\lfloor n/2 \rfloor) &\leq c \cdot \lfloor n/2 \rfloor \cdot \log(\lfloor n/2 \rfloor) \text{ und} \\ T(\lceil n/2 \rceil) &\leq c \cdot \lceil n/2 \rceil \cdot \log(\lceil n/2 \rceil). \end{aligned}$$

Im Induktionsschritt erhalten wir also:

$$\begin{aligned}
 T(n) &\leq c \cdot \lfloor n/2 \rfloor \cdot \log \lfloor n/2 \rfloor + c \cdot \lceil n/2 \rceil \cdot \log \lceil n/2 \rceil + n \\
 &\leq c \cdot \lfloor n/2 \rfloor \cdot (\log n - 1) + c \cdot \lceil n/2 \rceil \cdot \log n + n \\
 &\leq c \cdot n \cdot \log n - c \cdot \lfloor n/2 \rfloor + n \\
 &\leq c \cdot n \log n \text{ für } c \geq 3, n \geq 2.
 \end{aligned}$$

Für den Induktionsanfang muß natürlich noch die Randbedingung bewiesen werden, z.B. für $T(1) = 1$. Dies ist oft problematisch. Es gilt beispielsweise für kein $c > 0$, daß $T(1) = 1 \leq c \cdot 1 \cdot \log 1 = 0$ ist. Da uns bei Laufzeitfunktionen allerdings asymptotische Abschätzungen genügen, d.h. Abschätzung für $n \geq n_0$, können wir auch mit der Randbedingung $T(4)$ starten, d.h.

$$T(4) = 2 \cdot T(2) + 4 = 4 \cdot T(1) + 8 = 12 \leq c \cdot 4 \cdot \log 4 = c \cdot 8 \text{ für } c \geq 2.$$

Wie kommt man an eine gute Vermutung? Es gibt keine allgemeine Regel für die Substitutionsannahme, aber einige heuristische „Kochrezepte“. Lautet die Rekursionsgleichung etwa

$$T(n) = 2 \cdot T(n/2 + 17) + n,$$

so unterscheidet sich die Lösung vermutlich nicht substantiell von der Lösung für obige Rekursion, da die Addition von 17 im Argument von T für hinreichend große n nicht so erheblich sein kann. In der Tat ist hier wieder $T(n) \in \mathcal{O}(n \log n)$, und dies kann wieder mit Induktion bewiesen werden (vgl. Übung).

Manchmal läßt sich zwar die korrekte Lösung leicht vermuten, aber nicht ohne weiteres beweisen. So ist vermutlich

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \in \mathcal{O}(n).$$

Versucht man jedoch zu beweisen, daß $T(n) \leq c \cdot n$ für geeignetes $c > 0$ ist, so erhält man im Induktionsschritt

$$\begin{aligned}
 T(n) &\leq c \cdot \lfloor n/2 \rfloor + c \cdot \lceil n/2 \rceil + 1 \\
 &= c \cdot n + 1,
 \end{aligned}$$

aber kein $c > 0$ erfüllt $T(n) \leq c \cdot n$.

Trick. Starte mit der schärferen Vermutung $T(n) \leq c \cdot n - b$ für $c > 0$ und einer geeigneten Konstanten $b \geq 0$. Dann gilt

$$\begin{aligned} T(n) &\leq c \cdot \lfloor n/2 \rfloor - b + c \cdot \lceil n/2 \rceil - b + 1 \\ &\leq c \cdot n - 2b + 1 \\ &\leq c \cdot n - b \text{ für } b \geq 1. \end{aligned}$$

Ein weiterer Trick, der oft funktioniert, ist die Variablenersetzung.

Beispiel.

$$T(n) = 2 \cdot T(\lfloor \sqrt{n} \rfloor) + \log n.$$

Setze $m = \log n$, also $n = 2^m$, dann ergibt sich

$$T(2^m) = 2 \cdot T(\lfloor 2^{m/2} \rfloor) + m \leq 2 \cdot T(2^{m/2}) + m.$$

Setzt man nun $S(m) := T(2^m)$, so gilt

$$S(m) \leq 2 \cdot S(m/2) + m \in \mathcal{O}(m \log m).$$

Rückübersetzung von $S(m)$ nach $T(m)$ ergibt dann

$$T(n) = T(2^m) = S(m) \in \mathcal{O}(m \log m) = \mathcal{O}(\log n \log \log n).$$

1.2.2 Die Iterationsmethode

Eine naheliegende Methode zur Auflösung einer Rekursionsgleichung ist deren iterative Auflösung.

Beispiel.

$$\begin{aligned} T(n) &= 3 \cdot T(\lfloor n/4 \rfloor) + n \\ &= n + 3 \cdot (\lfloor n/4 \rfloor + 3 \cdot T(\lfloor n/16 \rfloor)) \\ &= n + 3 \cdot (\lfloor n/4 \rfloor + 3 \cdot (\lfloor n/16 \rfloor + 3 \cdot T(\lfloor n/64 \rfloor))) \\ &= n + 3 \cdot \lfloor n/4 \rfloor + 9 \cdot \lfloor n/16 \rfloor + 27 \cdot T(\lfloor n/64 \rfloor). \end{aligned}$$

Wie weit muß man die Rekursion iterativ auflösen, bis man die Randbedingungen erreicht?

Der i -te Term ist $3^i \cdot \lfloor n/4^i \rfloor$. Die Iteration erreicht im Argument 1, wenn $\lfloor n/4^i \rfloor \leq 1$, d.h. wenn $i \geq \log_4 n$. Dann ergibt sich

$$\begin{aligned} T(n) &\leq n + 3 \cdot n/4 + 9 \cdot n/16 + 27 \cdot n/64 + \dots + 3^{\log_4 n} \cdot c_1 \\ &\quad \text{für } c_1 \geq 0 \text{ konstant} \\ &\leq n \cdot \sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^i + c_1 \cdot n^{\log_4 3}, \quad \text{da } 3^{\log_4 n} = n^{\log_4 3} \\ &= 4n + c_1 \cdot n^{\log_4 3} \in \mathcal{O}(n), \quad \text{da } \log_4 3 < 1. \end{aligned}$$

Die Iterationsmethode führt oft zu aufwendigen (nicht unbedingt trivialen) Rechnungen. Durch iterative Auflösung einer Rekursionsabschätzung kann man jedoch manchmal zu einer guten Vermutung für die Substitutionsmethode gelangen.

1.3 Der Aufteilungs-Beschleunigungssatz

Es gibt einen sehr allgemeinen Satz über das asymptotische Wachstum rekursiv beschriebener Laufzeitfunktionen. Wir werden eine vereinfachte Form des Satzes ausführlich beweisen. Der Beweis des allgemeinen Satzes geht im Prinzip genauso; er ist nur technisch aufwendiger.

Satz 1.1 (allgemeine Form) Seien $a \geq 1$ und $b > 1$ Konstanten, $f(n)$ eine Funktion in n und $T(n)$ über nichtnegative ganze Zahlen definiert durch

$$T(n) = a \cdot T(n/b) + f(n),$$

wobei n/b für $\lfloor n/b \rfloor$ oder $\lceil n/b \rceil$ steht.

Dann gilt

- (i) $T(n) \in \Theta(n^{\log_b a})$, falls $f(n) \in \mathcal{O}(n^{\log_b a - \varepsilon})$ für eine Konstante $\varepsilon > 0$.
- (ii) $T(n) \in \Theta(n^{\log_b a} \cdot \log n)$, falls $f(n) \in \Theta(n^{\log_b a})$.
- (iii) $T(n) \in \Theta(f(n))$, falls $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ für eine Konstante $\varepsilon > 0$ und $a \cdot f(n/b) \leq c \cdot f(n)$ für eine Konstante $c < 1$ und für $n \geq n_0$.

Beispiel. MERGE SORT

$$b = 2, \quad a = 2, \quad f(n) \in \Theta(n) = \Theta(n^{\log_2 2})$$

Aus Fall (ii) folgt $T(n) \in \Theta(n \cdot \log n)$.

Wir schränken den Satz auf den Fall ein, daß $f(n) \in \mathcal{O}(n)$ ist, und beweisen ihn nur für Potenzen von b , d.h. $n = b^q$. Letztere Einschränkung wird aus technischen Gründen vorgenommen; man kann für $n \neq b^q$ die Analyse unter Betrachtung der nächsten Potenzen von b , d.h. $n_2 = b^{q+1}$ für $n_1 = b^q < n < n_2$ durchführen. Die Einschränkung auf $f(n) \in \mathcal{O}(n)$ vereinfacht den Beweis insofern, daß man nicht ausführlich $f(n)$ gegenüber $a \cdot T(n/b)$ abschätzen muß.

Satz 1.2 (eingeschränkte Form) Seien $a \geq 1, b > 1, c_1 > 0$ und $c_2 > 0$ Konstanten und $T(n)$ über nichtnegative ganze Zahlen definiert durch

$$\begin{aligned} c_1 \leq T(1) &\leq c_2 \quad \text{und} \\ a \cdot T(n/b) + c_1 n &\leq T(n) \leq a \cdot T(n/b) + c_2 \cdot n. \end{aligned}$$

Für $n = b^q$ gilt

- (i) $T(n) \in \Theta(n)$, falls $b > a$.
- (ii) $T(n) \in \Theta(n \cdot \log n)$, falls $a = b$.
- (iii) $T(n) \in \Theta(n^{\log_b a})$, falls $b < a$.

Beweis. Durch Induktion über q beweisen wir daß

$$T(n) \leq c_2 \cdot n \cdot \sum_{i=0}^q (a/b)^i.$$

Für $q = 0$ ergibt sich $T(1) \leq c_2$. Die Behauptung gelte also für $q > 0$. Betrachte $q + 1$:

$$\begin{aligned} T(b^{q+1}) &\leq a \cdot T(b^q) + c_2 \cdot b^{q+1} \\ &\leq a \cdot \left(c_2 \cdot b^q \cdot \sum_{i=0}^q (a/b)^i \right) + c_2 \cdot b^{q+1} \\ &= c_2 \cdot b^{q+1} \cdot \left(\sum_{i=0}^q (a/b)^{i+1} + 1 \right) \\ &= c_2 \cdot b^{q+1} \cdot \left(\sum_{i=1}^{q+1} (a/b)^i + 1 \right) \\ &= c_2 \cdot b^{q+1} \cdot \sum_{i=0}^{q+1} (a/b)^i. \end{aligned}$$

Analog läßt sich auch folgern

$$T(n) \geq c_1 \cdot n \cdot \sum_{i=0}^q (a/b)^i.$$

Fall $b > a$. Dann ist $a/b < 1$ und es gibt Konstante $k_1, k_2 > 0$, so daß

$$c_1 \cdot n \cdot k_1 \leq T(n) \leq c_2 \cdot n \cdot k_2, \text{ d.h. } T(n) \in \Theta(n).$$

Fall $b = a$.

$$\begin{aligned} T(n) &= T(b^q) \leq c_2 \cdot b^q \cdot \sum_{i=0}^q 1^i = c_2 \cdot b^q \cdot (q+1) \text{ und} \\ T(n) &\geq c_1 \cdot b^q \cdot (q+1), \text{ also } T(n) \in \Theta(n \log n). \end{aligned}$$

Fall $b < a$.

$$\begin{aligned} T(n) &= T(b^q) \leq c_2 \cdot b^q \cdot \sum_{i=0}^q (a/b)^i = c_2 \cdot \sum_{i=0}^q a^i \cdot b^{q-i} \\ &= c_2 \cdot \sum_{i=0}^q a^{q-i} \cdot b^i = c_2 \cdot a^q \cdot \sum_{i=0}^q (b/a)^i. \end{aligned}$$

Dann gibt es Konstante $k_1, k_2 > 0$, so daß

$$c_1 \cdot \underbrace{a^{\log_b n}}_{=n^{\log_b a}} \cdot k_1 \leq T(n) \leq c_2 \cdot \underbrace{a^{\log_b n}}_{=n^{\log_b a}} \cdot k_2$$

und damit $T(n) \in \Theta(n^{\log_b a})$.

■

Bemerkung. Gelten in der Voraussetzung von Satz 1.2 nur die Beschränkungen nach oben (unten), so gilt immer noch $T(n) \in \mathcal{O}(n)$ (bzw. $T(n) \in \Omega(n)$).

1.3.1 Beispiel: Matrix-Multiplikation nach Strassen

(Informatik II, Übung)

Problem. Gegeben zwei $(n \times n)$ -Matrizen A, B . Berechne $A \cdot B$ mit möglichst wenig Operationen.

Herkömmlich: $C = A \cdot B = (a_{ij}) \cdot (b_{ij}) = (c_{ij})$ mit $c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$. Die Laufzeit ist insgesamt in $\mathcal{O}(n^3)$.

Ein Divide-and-Conquer Ansatz für die Matrixmultiplikation ist:

$$A \cdot B = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix} = \begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = C,$$

wobei die a_i, b_i, c_i ($n/2 \times n/2$)-Matrizen sind. Dann berechnet sich C durch

$$\left. \begin{array}{l} c_1 = a_1 b_1 + a_2 b_3 \\ c_2 = a_1 b_2 + a_2 b_4 \\ c_3 = a_3 b_1 + a_4 b_3 \\ c_4 = a_3 b_2 + a_4 b_4 \end{array} \right\} \begin{array}{l} 8 \text{ Matrixmultiplikationen} \\ \text{von } (n/2 \times n/2)\text{-Matrizen und} \\ 4 \text{ Additionen von } (n/2 \times n/2)\text{-Matrizen} \end{array}$$

Addition zweier ($n \times n$)-Matrizen ist in $\Theta(n^2)$. Damit ergibt sich

$$\begin{aligned} T(n) &= 8 \cdot T(n/2) + c \cdot n^2, \quad c > 0 \text{ Konstante,} \\ c \cdot n^2 &\in \mathcal{O}(n^{\log_2 8 - \varepsilon}) = \mathcal{O}(n^{3 - \varepsilon}) \quad \text{mit } 0 < \varepsilon = 1 \\ \implies T(n) &\in \Theta(n^{\log_2 8}) = \Theta(n^3) \quad (\text{leider!}) \end{aligned}$$

Für die Laufzeit ist offensichtlich die Anzahl der Multiplikationen „verantwortlich“. Das heißt: Wenn man die Anzahl der Multiplikationen der ($n/2 \times n/2$)-Matrizen auf weniger als 8 reduzieren könnte bei ordnungsmäßig gleicher Anzahl von Additionen erhält man eine bessere Laufzeit.

Es gibt einen Divide-and-Conquer Ansatz von Strassen, der nur 7 Multiplikationen bei 18 Additionen verwendet. Als Laufzeit ergibt sich dann

$$\begin{aligned} T(n) &= 7 \cdot T(n/2) + c \cdot n^2 \\ \implies T(n) &\in \Theta(n^{\log_2 7}). \end{aligned}$$

Schema:

$$\left. \begin{array}{l} c_1 = P_5 + P_4 - P_2 + P_6 \\ c_2 = P_1 + P_2 \\ c_3 = P_3 + P_4 \\ c_4 = P_5 + P_1 - P_3 - P_7 \end{array} \right\} \text{ mit } \left\{ \begin{array}{l} P_1 = a_1 \cdot (b_2 - b_4) \\ P_2 = (a_1 + a_2) \cdot b_4 \\ P_3 = (a_3 + a_4) \cdot b_1 \\ P_4 = a_4 \cdot (b_3 - b_1) \\ P_5 = (a_1 + a_4) \cdot (b_1 + b_4) \\ P_6 = (a_2 - a_4) \cdot (b_2 + b_4) \\ P_7 = (a_1 - a_3) \cdot (b_1 + b_3) \end{array} \right.$$

1.3.2 Beispiel: Das Auswahlproblem (Selection)

Gegeben. Eine Folge von n Elementen, auf denen es eine Ordnung \leq gibt, und $i \in \{1, \dots, n\}$.

Problem. Gib das i -te (i -kleinste) Element aus.

Wir nehmen an, daß alle Elemente verschieden sind. (Falls gleiche Elemente auftreten, müßte zwischen diesen eine künstliche Ordnung festgelegt werden.) Dann ist x das i -te Element genau dann, wenn x größer als genau $i - 1$ Elemente der Folge ist. Ein einfacher Sortier-Algorithmus macht folgendes:

- Sortiere die Elemente in aufsteigender Reihenfolge und gib das i -te der sortierten Folge aus.

Mit MERGE SORT beispielsweise ergibt sich eine Laufzeit in $\Theta(n \log n)$. Ein Verfahren mit linearer worst-case-Laufzeit ist SELECT(n, i):

- Teile die n Elemente in $\lfloor n/5 \rfloor$ Gruppen mit jeweils 5 Elementen, und einer weiteren Gruppe mit den restlichen $n \bmod 5$ Elementen auf.
- Bestimme aus jeder der $\lfloor n/5 \rfloor$ Gruppen das mittlere Element, wobei in der letzten Gruppe das größere der beiden mittleren Elemente genommen wird, falls diese gerade Kardinalität hat.
- Rufe SELECT rekursiv auf, um das mittlere Element m der $\lfloor n/5 \rfloor$ mittleren Elemente zu bestimmen, d.h. SELECT($\lfloor n/5 \rfloor, \lfloor n/10 \rfloor$). Wenn $\lfloor n/5 \rfloor$ gerade ist, wird wiederum das größere Element genommen.
- Teile die Folge aller Elemente in die Teilfolge A_1 der Elemente $\leq m$, und die Teilfolge A_2 der Elemente $> m$ auf. Sei m das k -te Element unter allen n Elementen.
- Rufe SELECT rekursiv auf, um nun das i -te Element in der entsprechenden Teilfolge zu finden. D. h. falls $i \leq k$ ist, dann rufe SELECT(A_1, i) auf, und sonst SELECT($A_2, i - k$).

Laufzeitanalyse von SELECT. Es gilt: Die Anzahl der Elemente $> m$ ist mindestens $3n/10 - 6$. Begründung: Mindestens die Hälfte der mittleren Elemente aus den $\lfloor n/5 \rfloor$ Gruppen (Schritt 2) sind $> m$. Also steuert die Hälfte der $\lfloor n/5 \rfloor$ Gruppen jeweils 3 Elemente zu A_2 bei, außer evtl. der letzten Gruppe und der Gruppe, die m enthält, d.h.

$$\begin{aligned} |A_2| &\geq 3 \cdot \left(\frac{1}{2} \cdot \lfloor n/5 \rfloor - 2 \right) \geq \frac{3}{10}n - 6 \\ &\implies |A_1| \leq \frac{7}{10}n + 6. \end{aligned}$$

Entsprechend ist die Anzahl der Elemente kleiner als m ebenfalls mindestens $3 \cdot (\frac{1}{2} \cdot \lfloor n/5 \rfloor - 2)$, d.h.

$$|A_1| \geq \frac{3}{10}n - 6 \implies |A_2| \leq \frac{7}{10}n + 6.$$

Der rekursive Aufruf von SELECT in 5. wird also auf höchstens $\frac{7}{10}n + 6$ Elemente angewandt.

- Die Schritte 1., 2., 4. sind in $\mathcal{O}(n)$.
- Schritt 3. benötigt $T(\lfloor n/5 \rfloor)$ Zeit.
- Schritt 5. benötigt $T(\frac{7}{10}n + 6)$ Zeit; dabei ist $T(n)$ die Laufzeit von SELECT(n, i) mit $1 \leq i \leq n$.

Zunächst stellen wir fest, daß ab $n > 20$ gilt $\frac{7}{10}n + 6 < n$. Wir erhalten also als Abschätzung für die Laufzeit:

$$T(n) \leq \begin{cases} c_0 \cdot n & \text{für } n \leq n_0, \\ T(\lceil n/5 \rceil) + T(\frac{7}{10}n + 6) + c_1 \cdot n & \text{für } n > n_0. \end{cases}$$

$n_0 > 20$ geeignet gewählt

Wir beweisen $T(n) \in \mathcal{O}(n)$ durch Substitution. Sei also $T(n) \leq c \cdot n$ für eine Konstante $c > 0$ und alle $n \leq n_0$. Für $n > n_0$ folgt dann per Induktion

$$\begin{aligned} T(n) &\leq c \cdot \lceil n/5 \rceil + c \cdot \left(\frac{7}{10}n + 6 \right) + c_1 \cdot n \\ &\leq c \cdot n/5 + c + c \cdot \frac{7}{10}n + c \cdot 6 + c_1 \cdot n \\ &= 9 \cdot c \cdot \frac{n}{10} + 7 \cdot c + c_1 \cdot n. \end{aligned}$$

Damit $9 \cdot c \cdot \frac{n}{10} + 7 \cdot c + c_1 \cdot n \leq c \cdot n$ ist, muß c so gewählt werden können, daß $c_1 \cdot n \leq c \cdot \underbrace{\left(\frac{n}{10} - 7 \right)}_{>0 \text{ nötig}}$. Dazu muß $n > 70$ gelten. In der Abschätzung sollten wir also $n_0 > 70$ wählen.

