4 Rekursionsformeln

Wir haben in den vorangegangenen Kapiteln bereits rekursive Algorithmen (die Algorithmen rufen sich selbst auf) kennengelernt. Bei der Analyse des Laufzeitverhaltens bzw. des Speicherplatzbedarfs eines solchen entstehen oft *Rekursionsgleichungen* (*recurrence*, *Rekursionsformel*), die behandelt und gelöst werden müssen. Oft sind wir hierbei „nur“ am asymptotischen Verhalten der Lösung interessiert.

4.1 Grundlagen und einleitende Beispiele

Wir bezeichnen im Folgenden mit $\lfloor a \rfloor$, $a \in \mathbb{R}$, die größte ganze Zahl kleiner gleich a, d.h. a abgerundet auf eine ganze Zahl.

Beispiel 4.1. Ein Algorithmus habe in Abhängigkeit der Eingabegröße $n \in \mathbb{N}$ die Laufzeitfunktion $T(n)$. Durch rekursive Aufrufe ergebe sich $T(n)$ durch

$$T(n) = \begin{cases} 1 & \text{falls } n = 1 \\ 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n & \text{falls } n > 1 \end{cases}.$$

Wir wollen eine obere Schranke bestimmen und vermuten, dass

$$T(n) = \mathcal{O}(n \log(n))$$

die Lösung der Rekursionsgleichung ist. Wir müssen dazu zeigen, dass ein $c > 0$ und ein $n_0 \in \mathbb{N}$ existiert, so dass für alle $n \geq n_0$

$$T(n) \leq cn \log(n)$$

gilt. Der Nachweis der Korrektheit einer solchen Vermutung wird oft durch einen Induktionsbeweis geführt. Dazu gilt es ein n_0 für den Induktionsanfang zu finden. In unserem Fall führt $n_0 = 1$ nicht zum Erfolg, da

$$T(1) = 1 > c \cdot 1 \cdot \log(1) = c \cdot 0 = 0$$

für alle $c > 0$. Jedoch erfüllt $n_0 = 2$ alle Bedingungen für den Induktionsanfang. Wir führen einen Induktionsbeweis von $\left\lfloor \frac{n}{2} \right\rfloor$ nach n durch.

Induktionsanfang: Für $n = n_0 = 2$ gilt

$$T(2) = 2T(1) + 2 = 2 \cdot 1 + 2 = 4 \leq c \cdot 2 \log(2) = c \cdot 2$$

für alle $c > 1$.

Induktionsannahme: Es gelte für alle $t \in \mathbb{N}$, $t \leq \left\lfloor \frac{n}{2} \right\rfloor$

$$T(t) \leq ct \log(t).$$

Induktionsschritt $\left\lfloor \frac{n}{2} \right\rfloor \to n$:

$$\begin{aligned} T(n) &\leq 2\left(c\left\lfloor \frac{n}{2} \right\rfloor \log\left(\left\lfloor \frac{n}{2} \right\rfloor\right)\right) + n \\ &\leq cn \log\left(\frac{n}{2}\right) + n \\ &= cn\log(n) - cn\log(2) + n \\ &= cn\log(n) - cn + n \\ &\overset{c>1}{\leq} cn\log(n). \end{aligned}$$

Somit haben wir unsere Vermutung bewiesen mit beliebigem $c > 1$ und $n_0 = 2$.

Der Leser mache sich klar, dass es sich bei dem Beweis aus dem vorangegangenen Beispiel 4.1 tatsächlich um einen vollständigen Induktionsbeweis handelt und in der Tat so die Aussage für alle $n \geq 2$ gezeigt ist.

In der Praxis haben sich verschiedene Vorgehensweisen und Vereinfachungen im Zusammenhang mit Rekursionsformeln durchgesetzt, die wir in den nachfolgenden Anmerkungen zusammenfassen wollen.

Anmerkung 4.1. (*Ignorieren des Auf- und Abrundens*)
Bei der Lösung von Rekursionsformeln vernachlässigen wir oft gewisse technische Details, wie das Auf- und Abrunden der Größen der rekursiven Teilprobleme. Dadurch wird beispielsweise $T\left(\left\lceil \frac{n}{2} \right\rceil\right)$ zu $T\left(\frac{n}{2}\right)$. Diese Änderung wirkt sich asymptotisch nicht aus.

Anmerkung 4.2. (*Ignorieren der Anfangswertbedingung*)
Eine Anfangswertbedingung der Form $T(1) = c$, c eine Konstante, legt meist nur den konkreten konstanten Faktor bei der Lösungsfunktion fest. Ohne Angabe einer solchen ist die Lösungsmenge einer Rekursionsgleichung eine Funktionenschar. Bei der asymptotischen Notation ignorieren wir jedoch diese konstanten Faktoren und brauchen daher oft keine Anfangswertbedingungen zu betrachten.

Anmerkung 4.3. (*Asymptotische Notation in der Rekursionsformel*)
Wir sind an Lösungen der Rekursionsgleichung in der Form $T(n) = \Theta(...)$ interessiert. Deshalb können wir (mit der gebotenen Vorsicht) die asymptotische Notation bereits in der Rekursionsformel selbst verwenden. Dennoch müssen wir uns über die dahinterstehenden *exakten* Formulierungen (mit den enthaltenen Konstanten) im Klaren sein, wie wir in Beispiel 4.3 sehen werden.

Beispiel 4.2. Die asymptotische Laufzeit $T(n)$ des MergeSort-Algorithmus 7.2.3 für die Eingabegröße n ist

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n = 1 \\ T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + \Theta(n) & \text{falls } n > 1 \end{cases}.$$

Die angesprochene Vereinfachung durch Verzicht auf Auf- und Abrunden liefert

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{falls } n > 1 \end{cases}.$$

Wenn die Konstante für die Eingabegröße 1 in $\Theta(1)$ ist (also eine Konstante), so schreiben wir in der Regel

$$T(n) = \begin{cases} c & \text{falls } n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{falls } n > 1 \end{cases}$$

oder noch einfacher

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n),$$

da man oft asymptotische Aussagen treffen kann, ohne die Anfangswertbedingung verwendet zu haben, denn letztere wirkt sich höchstens auf die in beispielsweise der Θ-Notation versteckten Konstanten aus.

Beispiel 4.3. (*Fehlerhafte Verwendung der asymptotischen Notation*)
Betrachten wir folgende Rekursionsgleichung, die wir iterativ ineinander einsetzen

$$\begin{aligned} T(n) &= \Theta(n) + T\left(\frac{n}{2}\right) \\ &= \Theta(n) + \underbrace{\Theta\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right)}_{=T\left(\frac{n}{2}\right)} \\ &\vdots \\ &= \Theta(n) + \Theta\left(\frac{n}{2}\right) + \Theta\left(\frac{n}{4}\right) + \ldots, \end{aligned}$$

so dürfen wir *nicht* den letzten Ausdruck zu

$$\Theta(n) + \Theta(n) + \Theta(n) + ...$$

umformen, da in den verschiedenen Θ-Notationen *immer dieselben* Konstanten stecken, was so auf ein *falsches Ergebnis* führt. Wir gehen nun davon aus, dass obige Rekursionsformel aus dem Vernachlässigen von $\lceil . \rceil$ entstanden ist und $T(1)$ eine Konstante ist. Wir erhielten in diesem Fall bei sorglosem Umgang mit der asymptotischen Notation

$$\underbrace{\Theta(n) + \Theta(n) + ... + \Theta(n)}_{\lceil \log(n) \rceil} + \Theta(1) = \lceil \log(n) \rceil \cdot \Theta(n) = \Theta(n \log(n)),$$

was *falsch* ist, wie wir an dieser Stelle ausführlich nachweisen wollen. Wir führen einen Widerspruchsbeweis.

Annahme: $T(n) \in \Theta(n \log(n))$,
also existieren $c_1, c_2 > 0$ und ein $n_0 \in \mathbb{N}$, so dass für alle $n \geq n_0$

$$c_1 n \log(n) \leq T(n) \leq c_2 n \log(n)$$

gilt. In

$$T(n) = \Theta(n) + T\left(\frac{n}{2}\right)$$

steht $\Theta(n)$ für eine konkrete Funktion $f(n) \in \Theta(n)$ für die es Konstanten $\widetilde{c}_1, \widetilde{c}_2 > 0$ und ein $\widetilde{n}_0 \in \mathbb{N}$ gibt, so dass für alle $n \geq \widetilde{n}_0$

$$\widetilde{c}_1 n \leq f(n) \leq \widetilde{c}_2 n$$

gilt. Also erhalten wir

$$\begin{aligned} c_1 n \log(n) &\leq T(n) \\ &= f(n) + T\left(\frac{n}{2}\right) \\ &\leq \widetilde{c}_2 n + T\left(\frac{n}{2}\right) \\ &\leq \widetilde{c}_2 n + c_2 \frac{n}{2} \log\left(\frac{n}{2}\right) \\ &= \widetilde{c}_2 n + c_2 \frac{n}{2} \log(n) - c_2 \frac{n}{2}. \end{aligned}$$

Wir formen weiter um

$$\begin{aligned} & c_1 n \log(n) && \leq && \widetilde{c}_2 n + c_2 \frac{n}{2} \log(n) - c_2 \frac{n}{2} \\ \Leftrightarrow \quad & \left(c_1 - \frac{c_2}{2}\right) n \log(n) && \leq && \left(\widetilde{c}_2 - \frac{c_2}{2}\right) n \\ \Leftrightarrow \quad & \underbrace{\log(n)}_{\to \infty} && \leq && \underbrace{\frac{\widetilde{c}_2 - \frac{c_2}{2}}{c_1 - \frac{c_2}{2}}}_{\text{Konstante}}, \end{aligned}$$

was ein Widerspruch ist, da sich $\log(n)$ durch keine Konstante nach oben beschränken lässt, also war unsere Annahme falsch und es ist

$$T(n) \notin \Theta(n \log(n)).$$

Mit dem Master-Theorem (Satz 4.1) lässt sich in bequemer Weise bestimmen, dass die Lösung der Rekursionsformel in der Tat *nicht* $\Theta(n \log(n))$ sondern $\Theta(n)$ ist.

In Beispiel 4.1 haben wir die Lösung der gegebenen Rekursionsformel erraten. Dies ist die grundsätzliche Vorgehensweise der *Substitutionsmethode* (*Ersetzungsmethode*). Wir werden im Folgenden drei Verfahren zum Lösen von Rekursionsgleichungen kennenlernen.

1. Substitutionsmethode (Ersetzungsmethode)
2. Rekursionsbäume
3. Master-Methode (Master-Theorem)

4.2 Substitutionsmethode

In Beispiel 4.1 haben wir die Substitutionsmethode bereits angewendet. Wir können die Vorgehensweise wie folgt zusammenfassen:

1. Errate die Form der Lösung
2. Beweise die Lösung und finde entsprechende Konstanten durch eine oder mehrere der folgenden Methoden
   a) (Vollständige) Induktion
   b) Beweis einer stärkeren Aussage
   c) Variablensubstitution

In Beispiel 4.1 haben wir die Lösung erraten und dann mittels Induktion bewiesen. Es gibt aber auch Fälle, in denen die Abschätzung mittels vollständiger Induktion nicht funktioniert.

Beispiel 4.4. Sei

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1.$$

Wir erraten die Lösung $T(n) = \mathcal{O}(n)$, also versuchen wir $T(n) \leq cn$ für ein $c > 0$ ab einem $n_0 \in \mathbb{N}$ mittels induktiver Einsetzungsmethode zu zeigen, was uns auf

$$\begin{aligned} T(n) &= T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1 \\ &\leq c\left\lfloor \frac{n}{2} \right\rfloor + c\left\lceil \frac{n}{2} \right\rceil + 1 \\ &= cn + 1 \end{aligned}$$

führt. Der letzte Ausdruck lässt sich nicht durch cn nach oben abschätzen.

Achtung: Sorgloser Umgang mit der asymptotischen Notation führt oft zu nur schwer nachvollziehbaren Fehlern, wie wir schon in Beispiel 4.3 aufgezeigt haben. Eine Weiterführung der Art

$$\ldots = cn + 1 = \mathcal{O}(n),$$

um letztendlich $T(n) = \mathcal{O}(n)$ zu beweisen, ist hier *falsch*! Natürlich gibt es (sogar für jedes $c > 0$) eine Konstante $c_1 > 0$ mit

$$cn + 1 \leq c_1 n,$$

jedoch liegt hier der Fehler darin, dass zu diesem Zeitpunkt die Konstante c für die (zu beweisende) $\mathcal{O}$-Notation zwar beliebig, aber bereits festgelegt ist. Genauer gilt es zu beweisen, dass $T(n) \leq cn$ mit genau diesem c ist. Würden wir nun oben das so gefundene c_1 ansetzen bzw. einsetzen, würden wir wieder auf das selbe Problem stoßen, was an der selben Stelle auf ein c_2 bei dieser (nun offensichtlich falschen) Vorgehensweise führen würde und so weiter. So erhalten wir keine Konstante c.

In diesem Fall lösen wir das Problem, indem wir die *stärkere* Behauptung

$$T(n) \leq cn - b$$

mit $b > 0$ beweisen. Wir ziehen also von unserer eigentlichen Lösungsvermutung eine Konstante ab. Damit erhalten wir dann für den Induktionsschluss

$$\begin{aligned} T(n) &= T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1 \\ &\leq (c\left\lfloor \frac{n}{2} \right\rfloor - b) + (c\left\lceil \frac{n}{2} \right\rceil - b) + 1 \\ &= cn - 2b + 1 \\ &\overset{b\geq 1}{\leq} cn - b. \end{aligned}$$

In manchen Fällen können wir die Rekursionsgleichung erst lösen, wenn wir vorher eine Variablensubstitution durchführen.

Beispiel 4.5. Sei

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

Wir setzen $m := \log(n)$ und erhalten

$$T(2^m) = 2T(2^{\frac{m}{2}}) + m.$$

Dann setzen wir $S(m) := T(2^m)$ und erhalten die Rekursionsgleichung

$$S(m) = 2S\left(\frac{m}{2}\right) + m.$$

Hierfür haben wir in Beispiel 4.1 bereits die Lösung $S(m) = \mathcal{O}(m \log(m))$ bewiesen. Somit ist die Lösung für das hier vorliegende Problem nach Rücksubstitution

$$\begin{aligned} & S(m) = \mathcal{O}(m \log(m)) \\ \overset{m=\log(n)}{\Leftrightarrow} \quad & S(\log(n)) = \mathcal{O}(\log(n) \log(\log(n))) \\ \overset{S(m)=T(2^m)}{\Leftrightarrow} \quad & T(2^{\log(n)}) = \mathcal{O}(\log(n) \log(\log(n))) \\ \Leftrightarrow \quad & T(n) = \mathcal{O}(\log(n) \log(\log(n))). \end{aligned}$$

Das Hauptproblem bei der Substitutionsmethode ist natürlich das Erraten der Lösung bzw. der Lösungsform. Hierzu können wir nur allgemeine Hinweise geben, da sich hier insbesondere reichhaltige Erfahrung auf diesem Gebiet auszahlt und es kein Kochrezept geben kann.

Anmerkung 4.4. (*Erraten der Lösung einer Rekursionsformel*)

1. Erfahrung: Ähnlichkeit mit bekannten Rekursionsformeln
2. Verwende Rekursionsbäume (Abschnitt 4.3), um die asymptotischen Größenordnungen abzuschätzen
3. Verwende untere und obere Schranken, um die asymptotischen Größenordnungen immer weiter einzuschränken

4.3 Rekursionsbäume

Im vorangegangenen Abschnitt wurden Rekursionsbäume bereits angesprochen, um die asymptotischen Laufzeiten eines rekursiven Verfahrens abzuschätzen. Die asymptotische Laufzeit kann dann mittels der Substitutionsmethode bewiesen werden.

Rekursive Algorithmen zerlegen das zu lösende Problem in Teilprobleme. Die Gesamtlösung setzt sich dann durch die Lösungen der Teilprobleme zusammen. Genau diese Zerlegung in Teilprobleme wird bzgl. der Kosten in einem Rekursionsbaum visualisiert. Wir erläutern den Aufbau und die Funktionsweise von Rekursionsbäumen in Form des folgenden Beispiels.

Beispiel 4.6. Wir betrachten die Rekursion

$$T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2),$$

für die wir eine obere Schranke für die Lösung finden wollen. Wir ignorieren das Abrunden und ersetzen (um eine obere Schranke zu erhalten) $\Theta(n^2)$ durch cn^2 mit der nach Definition von Θ existierenden Konstanten $c > 0$. Somit wollen wir die Rekursionsgleichung

$$T(n) = 3T\left(\frac{n}{4}\right) + cn^2$$

lösen. Wir bauen nun zunächst den Rekursionsbaum für die erste Rekusionsstufe auf. Die rekursiven Aufrufe ($T\left(\frac{n}{4}\right)$) werden dabei zu den Blättern, die Wurzel besteht aus den „ablesbaren“ Kosten, genauer die Kosten, die nicht in den folgenden Stufen expandiert werden (cn^2).

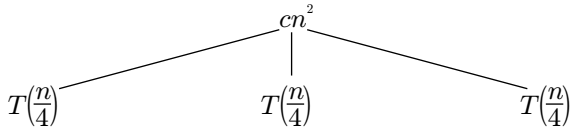

Abbildung 4.1: Rekursionsbaum der ersten Rekursionsstufe

In den nachfolgenden Stufen erweitern wir den Baum in der gleichen Art und Weise, indem die im Baum vorhandene Rekursion durch weitere Stufen expandiert wird.

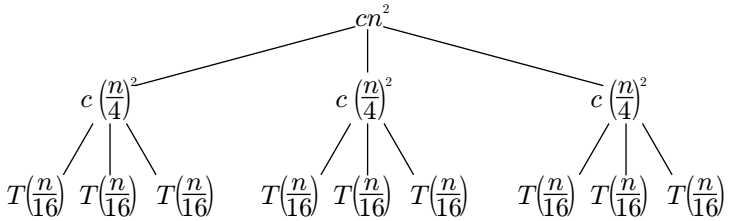

Abbildung 4.2: Rekursionsbaum der zweiten Rekursionsstufe

Wir führen dies solange fort, bis wir idealerweise eine Vorstellung von der Lösung gewonnen haben.

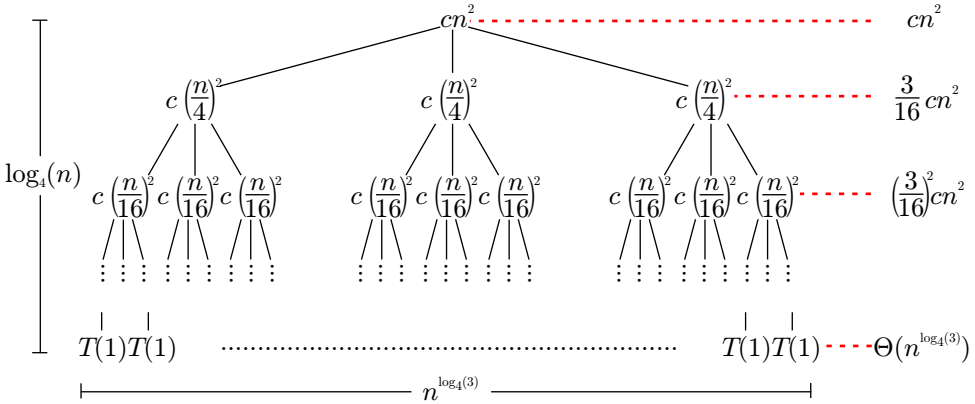

Abbildung 4.3: Rekursionsbaum mit Auswertung

Wir vermuten also, dass gilt

$$\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + ... + \left(\frac{3}{16}\right)^{\log_4(n)-1} cn^2 + \Theta(n^{\log_4(3)}) \\
&= cn^2 \sum_{i=0}^{\log_4(n)-1} \left(\frac{3}{16}\right)^i + \Theta(n^{\log_4(3)}) \\
&= \frac{\left(\frac{3}{16}\right)^{\log_4(n)} - 1}{\left(\frac{3}{16}\right) - 1} cn^2 + \Theta(n^{\log_4(3)}) \\
&\overset{n\geq 4}{\leq} 2cn^2 + \Theta(n^{\log_4(3)})
\end{aligned}$$

also

$$T(n) \in \mathcal{O}(n^2).$$

Mittels Substitutionsmethode zeigen wir nun, dass $T(n) \leq dn^2$ ist für ein geeignetes $d > 0$. Es ist

$$\begin{aligned}
T(n) &\leq 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + cn^2 \\
&\leq 3d\left\lfloor \frac{n}{4} \right\rfloor^2 + cn^2 \\
&\leq 3d\left(\frac{n}{4}\right)^2 + cn^2 \\
&= \frac{3}{16}dn^2 + cn^2
\end{aligned}$$

und in der Tat gilt für alle d mit

$$d \geq \frac{16}{13}c,$$

dass

$$\frac{3}{16}dn^2 + cn^2 \leq dn^2$$

ist und somit insgesamt

$$T(n) \leq dn^2$$

für $d \geq \frac{16}{13}c$.

4.4 Master-Methode

Kombiniert man die bereits angesprochenen Methoden (induktives Einsetzen u.s.w.) so kann man auch allgemeine Aussagen über Lösungen von Rekursionsformeln bestimmter Form beweisen. Wir betrachten nun eine Klasse von Rekursionsgleichungen, wie sie typischerweise bei Divide&Conquer-Algorithmen auftreten

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \text{ mit } a, b \in \mathbb{N},\ a \geq 1,\ b > 1 \text{ und } f(n) \geq 0 \text{ für alle } n \geq n_0.$$

Hierbei darf an die Stelle von $T\left(\frac{n}{b}\right)$ auch $T\left(\left\lfloor \frac{n}{b} \right\rfloor\right)$ oder $T\left(\left\lceil \frac{n}{b} \right\rceil\right)$ treten, wie das in der Regel der Fall sein wird, da n nicht notwendigerweise eine Potenz von b sein muss. Um die Vorgehensweisen dennoch möglichst intuitiv und verständlich darlegen zu können, werden wir im Folgenden jedoch davon ausgehen, dass n eine Potenz von b ist. Die Berechnungen und Beweise für beliebige Zahlen können im Lehrbuch *Introduction to Algorithms* von *Cormen et al.* nachgelesen werden.

Ein Rekursionsbaum für diese von uns nun betrachtete Klasse von Rekursionsgleichungen sieht wie folgt aus

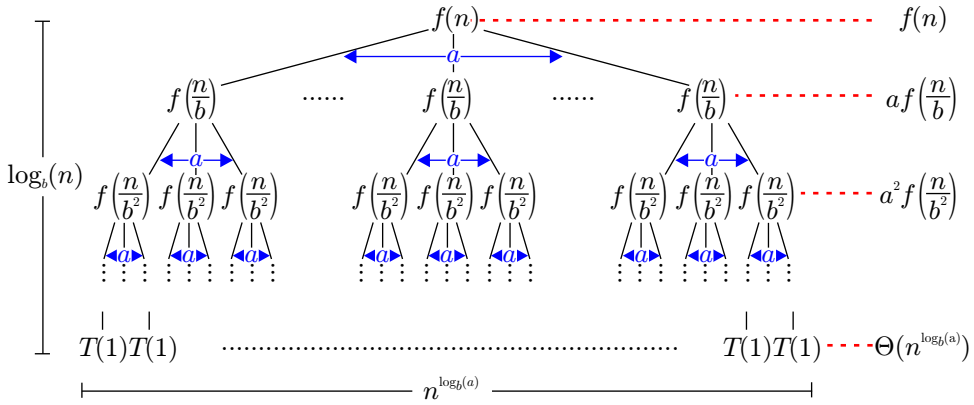

Abbildung 4.4: Rekursionsbaum für die Rekursionsgleichungsklasse des Master-Theorems

Summieren wir nun die Kosten aller Schichten des Baumes auf, so erhalten wir als Gesamtkosten

$$\begin{aligned} T(n) &= f(n) + af\left(\frac{n}{b}\right) + a^2 f\left(\frac{n}{b^2}\right) + \ldots + a^{\log_b(n)-1} f\left(\frac{n}{b^{\log_b(n)-1}}\right) + \Theta(n^{\log_b(a)}) \\ &= \underbrace{\sum_{i=0}^{\log_b(n)-1} a^i f\left(\frac{n}{b^i}\right)}_{=:g(n)} + \Theta(n^{\log_b(a)}) \\ &= g(n) + \Theta(n^{\log_b(a)}) \end{aligned}$$

mit $g(n) = \sum_{i=0}^{\log_b(n)-1} a^i f\left(\frac{n}{b^i}\right)$.

Wir werden nun im Folgenden drei Fälle für $f(n)$ unterscheiden, anhand derer wir die jeweilige Lösung für $T(n)$ der Rekursionsgleichung in den einzelnen Fällen herleiten.

Erster Fall. Wir betrachten zunächst den Fall, dass

$$f(n) = \mathcal{O}(n^{\log_b(a)-\varepsilon})$$

für eine Konstante $\varepsilon > 0$. Dann ist

$$f\left(\frac{n}{b^i}\right) = \mathcal{O}\left(\left(\frac{n}{b^i}\right)^{\log_b(a)-\varepsilon}\right),$$

also existieren $c > 0$ und $n_0 \in \mathbb{N}$, so dass für alle $n \geq n_0$

$$f\left(\frac{n}{b^i}\right) \leq c\left(\frac{n}{b^i}\right)^{\log_b(a)-\varepsilon}$$

gilt. Somit gilt für alle $n \geq n_0$

$$\begin{aligned}
g(n) &= \sum_{i=0}^{\log_b(n)-1} a^i f\left(\frac{n}{b^i}\right) &&\leq c\sum_{i=0}^{\log_b(n)-1} a^i \left(\frac{n}{b^i}\right)^{\log_b(a)-\varepsilon} \\
&= cn^{\log_b(a)-\varepsilon} \sum_{i=0}^{\log_b(n)-1} \left(\frac{ab^\varepsilon}{b^{\log_b(a)}}\right)^i &&= cn^{\log_b(a)-\varepsilon} \sum_{i=0}^{\log_b(n)-1} b^{\varepsilon i} \\
&= cn^{\log_b(a)-\varepsilon} \frac{b^{\varepsilon \log_b(n)} - 1}{b^\varepsilon - 1} &&= cn^{\log_b(a)-\varepsilon} \frac{n^\varepsilon - 1}{b^\varepsilon - 1} \\
&= \frac{c}{b^\varepsilon - 1} \cdot \Big(1 - \underbrace{\frac{1}{n^\varepsilon}}_{\in(0,1)}\Big) n^{\log_b(a)} &&< \underbrace{\frac{c}{b^\varepsilon - 1}}_{\text{Konstante}} n^{\log_b(a)}
\end{aligned}$$

also

$$g(n) \in \mathcal{O}(n^{\log_b(a)}).$$

Damit ist dann

$$T(n) = \Theta(n^{\log_b(a)}) + g(n) = \Theta(n^{\log_b(a)}) + \mathcal{O}(n^{\log_b(a)}) = \Theta(n^{\log_b(a)}).$$

Zweiter Fall. Wir betrachten nun den Fall

$$f(n) = \Theta(n^{\log_b(a)}).$$

Damit ist analog zum obigen Fall

$$f\left(\frac{n}{b^i}\right) = \Theta\left(\left(\frac{n}{b^i}\right)^{\log_b(a)}\right),$$

also gibt es $c_1, c_2 > 0$ und $n_0 \in \mathbb{N}$, so dass für alle $n \geq n_0$

$$c_1\left(\frac{n}{b^i}\right)^{\log_b(a)} \leq f\left(\frac{n}{b^i}\right) \leq c_2\left(\frac{n}{b^i}\right)^{\log_b(a)}$$

gilt. Wegen

$$\begin{aligned}\sum_{i=0}^{\log_b(n)-1} a^i \left(\frac{n}{b^i}\right)^{\log_b(a)} &= n^{\log_b(a)} \sum_{i=0}^{\log_b(n)-1} \left(\frac{a}{b^{\log_b(a)}}\right)^i \\ &= n^{\log_b(a)} \sum_{i=0}^{\log_b(n)-1} 1 \\ &= n^{\log_b(a)} \log_b(n)\end{aligned}$$

ist also

$$c_1 n^{\log_b(a)} \log_b(n) \leq g(n) \leq c_2 n^{\log_b(a)} \log_b(n)$$

für $n \geq n_0$ und somit

$$g(n) \in \Theta(n^{\log_b(a)} \log_b(n)).$$

Damit ist

$$T(n) = \Theta(n^{\log_b(a)}) + g(n) = \Theta(n^{\log_b(a)}) + \Theta(n^{\log_b(a)} \log_b(n)) = \Theta(n^{\log_b(a)} \log_b(n)).$$

Dritter Fall. Wir nehmen in diesem Fall an, dass ein $n_0 \in \mathbb{N}$ existiert, so dass

$$af\left(\frac{n}{b}\right) \leq cf(n)$$

für eine Konstante $0 < c < 1$ und für alle $n \geq n_0$. Dann ist

$$f\left(\frac{n}{b}\right) \leq \frac{c}{a} f(n)$$

und durch rekursives Einsetzen erhalten wir so

$$f\left(\frac{n}{b^i}\right) \leq \left(\frac{c}{a}\right)^i f(n)$$

für $i \in \mathbb{N}$. Damit ist

$$g(n) = \sum_{i=0}^{\log_b(n)-1} a^i f\left(\frac{n}{b^i}\right) \leq \sum_{i=0}^{\log_b(n)-1} c^i f(n) = f(n) \sum_{i=0}^{\log_b(n)-1} c^i = f(n) \frac{1 - c^{\log_b(n)}}{1 - c}.$$

Da $c < 1$, also insbesondere $0 < c^{\log_b(n)} \to 0$ für $n \to \infty$, existiert ein $n_c \in \mathbb{N}$ so dass für alle $n \geq n_c$ gilt

$$c^{\log_b(n)} \leq \frac{1}{2}.$$

Also erhalten wir für $n \geq \max\{n_0, n_c\}$

$$\frac{1}{2} \frac{1}{1-c} f(n) \leq g(n) \leq \frac{1}{1-c} f(n),$$

also

$$g(n) \in \Theta(f(n)).$$

Nehmen wir nun an, dass

$$f(n) = \Omega(n^{\log_b(a)+\varepsilon})$$

für eine Konstante $\varepsilon > 0$, so liefert uns das

$$T(n) = \Theta(n^{\log_b(a)}) + g(n) = \Theta(n^{\log_b(a)}) + \Theta(f(n)) \overset{f(n)=\Omega(n^{\log_b(a)+\varepsilon})}{=} \Theta(f(n)).$$

Die Erkenntnisse der obigen drei Fälle, die wir unter der Voraussetzung, dass n eine Potenz von b ist, hergeleitet haben, werden wir nun in folgendem Satz zusammenfassen. Wir weisen darauf hin, dass sich die Aussagen für beliebige n mit Hilfe der obigen Herleitungen beweisen lassen, wie dies in *Introduction to Algorithms* von *Cormen et al.* aufgezeigt wird.

Satz 4.1. *(Master-Theorem)*
Seien $a \geq 1$ und $b > 1$ Konstanten, sei $f(n)$ eine (asymptotisch positive) Funktion und sei $T(n)$ die Rekursion der Form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

wobei $\frac{n}{b}$ entweder als $\lfloor \frac{n}{b} \rfloor$ oder als $\lceil \frac{n}{b} \rceil$ interpretiert werden kann. Dann gilt

1. *Ist $f(n) = \mathcal{O}(n^{\log_b(a)-\varepsilon})$ für ein $\varepsilon > 0$, so gilt*

$$T(n) = \Theta(n^{\log_b(a)}).$$

2. *Ist $f(n) = \Theta(n^{\log_b(a)})$ dann ist*

$$T(n) = \Theta(n^{\log_b(a)} \log(n)).$$

3. *Ist $f(n) = \Omega(n^{\log_b(a)+\varepsilon})$ für ein $\varepsilon > 0$ und ist $af\left(\frac{n}{b}\right) \leq cf(n)$ für eine Konstante $c < 1$ und genügend großes n, so gilt*

$$T(n) = \Theta(f(n)).$$

Beispiel 4.7. Die Laufzeit $T(n)$ mit

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

des Algorithmus MergeSort 7.2.3 ist in $\Theta(n \log(n))$, denn nach Satz 4.1 ist mit $a = b = 2$ und daher $\log_b(a) = \log_2(2) = 1$

$$f(n) = \Theta(n^{\log_b(a)}) = \Theta(n^1) = \Theta(n).$$

Somit folgt aus dem zweiten Fall

$$T(n) = \Theta(n \log(n)).$$

Beispiel 4.8. Wir betrachten nun die Rekursion

$$T(n) = 9T\left(\frac{n}{3}\right) + n.$$

Mit $a = 0$, $b = 3$, $f(n) = n$ und somit $\log_b(a) = \log_3(9) = 2$ ist

$$f(n) = \mathcal{O}(n^{\log_b(a)-\varepsilon}) \overset{\varepsilon:=1}{=} \mathcal{O}(n^{2-1}) = \mathcal{O}(n).$$

Also folgt aus dem ersten Fall von Satz 4.1

$$T(n) = \Theta(n^2).$$

Beispiel 4.9. Wir betrachten die Rekursion

$$T(n) = 3T\left(\frac{n}{4}\right) + n\log(n).$$

Mit $a = 3$, $b = 4$, $f(n) = n\log(n)$ und somit $\log_b(a) = \log_4(3) \approx 0.793$ ist

$$f(n) = \Omega(n^{\log_4(3)+\varepsilon}) = \Omega(n)$$

mit $\varepsilon \approx 0.207$. Für große n gilt nun

$$3f\left(\frac{n}{4}\right) = 3\frac{n}{4}\log\left(\frac{n}{4}\right) \leq \frac{3}{4}n\log(n) = \frac{3}{4}f(n)$$

und somit folgt aus dem dritten Fall von Satz 4.1

$$T(n) = \Theta(f(n)) = \Theta(n\log(n)).$$