

Komplexität jenseits der Vorstellungskraft

Eine Reise durch die Welt der großen Zahlen

Bachelorarbeit

Zur Erlangung des
Bachelor of Education (BEd)

an der School of Education der
Paris-Lodron-Universität Salzburg



Eingereicht von
JOHANNES BAISCHER
12127340

Betreuer: Ao.Univ.-Prof. Mag. Dr. Wolfgang Schmid
Fachbereich: Mathematik

Salzburg, Juni 2025

Inhaltsverzeichnis

1	Vorwort	II
2	Abstract	III
3	Einführung in die Komplexitätstheorie	1
3.1	Grundlegende Definitionen	1
3.2	„Groß-O“-Notation	3
4	Die Spielregeln	7
4.1	Big Number Duel	7
4.2	Der Zeichenvorrat	8
4.3	Potentielle Ausnahmen	9
4.4	Die Rangliste	10
5	Naive Versuche einer großen Zahl	11
5.1	Ausreizen der Grundrechnungsarten	11
5.2	Ausreizen weiterer bekannter Operatoren	16
5.3	Die Ackermann-Funktion	19
6	Bekannte Vertreter der absurd großen Zahlen	23
6.1	Graham’s Number	23
6.2	$TREE(3)$	26
6.3	Busy Beaver und Rayo’s Number	31
7	Fazit	34
	Literatur	i

1 Vorwort

Höher, schneller, weiter! Der Mensch hat sich schon immer daran erfreut, Grenzen zu überschreiten und das Unmögliche möglich zu machen. Doch was passiert, wenn wir an die Grenzen des Machbaren stoßen? Was passiert, wenn wir versuchen, Zahlen zu berechnen, die so groß sind, dass sie unsere Vorstellungskraft übersteigen? In dieser Arbeit begeben wir uns auf eine Reise durch die Welt der großen Zahlen und versuchen, die Grenzen des Berechenbaren zu finden.

Besonderer Dank gilt an dieser Stelle all jenen, die mich beim Verfassen dieser Arbeit unterstützt haben, allen voran meinem Betreuer Wolfgang Schmid, für seine Unterstützung und seine wertvollen Anregungen.

Johannes Baischer

Salzburg, 23. Juni 2025

(Komplexitätstheorie, große Zahlen, Big Number Duel)

2 Abstract

Die faszinierende Welt der großen Zahlen ist gefüllt mit mathematischen Konzepten, die es ermöglichen, Zahlen zu erzeugen, die alleine für ihre Darstellung sehr viel Kreativität erfordern, geschweige denn ihre Größe zu visualisieren. Beginnend mit den Grundrechnungsarten und einfachen Algorithmen werden wir schrittweise mächtigere Werkzeuge einführen, um immer größere Zahlen zu erzeugen. Es werden bekannte Zahlen wie Graham's Number, $TREE(3)$, die Busy Beaver-Funktion und Rayo's Number analysiert, die jeweils neue Grenzen der Darstellbarkeit und Berechenbarkeit aufzeigen. Ausgangspunkt wird ein historischer Wettstreit zweier Mathematiker sein, die versuchen, sich gegenseitig mit immer größeren Zahlen zu übertrumpfen. Dabei werden wir die Spielregeln des *Big Number Duel* als Grundlage für unsere Suche nach „der größten Zahl“ verwenden.

Eingangs bietet eine kurze Einführung in die Komplexitätstheorie eine Grundlage, um das Verhalten von Algorithmen und Zahlenfolgen systematisch zu analysieren und zu vergleichen. Ein zentraler Bestandteil dieser Analyse ist die Groß-O-Notation, die verwendet wird, um das asymptotische Wachstum von Funktionen zu beschreiben. Durch die Groß-O-Notation können wir mathematisch präzise verschiedene Strategien zur Erzeugung großer Zahlen vergleichen und deren Wachstum klassifizieren, solange uns dies möglich ist.

3 Einführung in die Komplexitätstheorie

Um uns effektiv auf die Suche nach „der größten Zahl“ begeben zu können, müssen wir uns zunächst darauf einigen, wonach wir genau suchen. Wer als Kind schon einmal versucht hat, mit den Freunden um die größte Zahl zu konkurrieren, weiß, dass dies schnell in Streitereien enden kann. Ist „Eine Millionen-Million“ eine Zahl, was ist mit „immer eins mehr als du“ und hat man gewonnen, wenn man „unendlich“ sagt? Um solche Probleme zu vermeiden, müssen wir uns auf einige einheitliche Definitionen einigen. In diesem Kapitel werden wir uns daher damit beschäftigen, unsere Vorgehensweise zu konkretisieren und eine Methode finden, die uns helfen wird, unsere Ergebnisse zu vergleichen.

Viele der in dieser Arbeit vorkommenden Begriffe und Definitionen stammen aus dem dritten Kapitel von *Grundlagen der Mathematischen Optimierung - Diskrete Strukturen, Komplexitätstheorie, Konvexitätstheorie, Lineare Optimierung, Simplex-Algorithmus, Dualität* [Gri13] und aus Vorlesungsskripten basierend auf dem Buch *Introduction to algorithms* [Cor+09].

3.1 Grundlegende Definitionen

Für unsere Zwecke werden wir in weiterer Folge unterscheiden, ob wir die Null zu den natürlichen Zahlen zählen oder nicht. Dafür definieren wir:

Definition 3.1: Natürliche Zahlen

- $\mathbb{N}_0 = \{0, 1, 2, 3, \dots\}$ als die Menge der nicht-negativen ganzen Zahlen
- $\mathbb{N} = \{1, 2, 3, \dots\}$ als die Menge der positiven ganzen Zahlen

Definition 3.2: Elementare Operationen

Wir nennen *Addition* (+), *Subtraktion* (−), *Multiplikation* (·), *Division* (÷) **elementare arithmetische Operationen**.

Weiters bezeichnen wir Vergleiche (=, ≠, ≤, ≥) als **elementare Vergleichsoperationen**.

Gemeinsam mit der Zuweisung (←) von Werten an Variablen bilden sie die **elementaren Operationen**.

Im selben Zug werden hier oft bestimmte Begriffe aus der Informatik eingeführt, welche für die Formalisierung von Algorithmen eingeführt und stark reduziert werden. Für unsere Zwecke reicht es jedoch zu wissen, dass sich viele Prozesse auf das ständige Wiederholen einer Aufgabe (=Schleife **WHILE DO**) und Reagieren auf bestimmte Bedingungen

(=Verzweigung **IF ELSE**) reduzieren lassen. Damit haben wir bereits die Grundlage für einen der Kernbegriffe dieser Arbeit gelegt: *Algorithmen*.

Definition 3.3: Algorithmus (frei formuliert)

Ein **Algorithmus** $\mathcal{A}$ ist eine endliche Folge von elementaren Operationen, die auf einen Input $\mathcal{I}$ angewendet wird und einen Output $\mathcal{O}$ liefert.
Man sagt ein Algorithmus $\mathcal{A}$ **löst** ein Problem $\mathcal{P}$, wenn er für jede Eingabe $\mathcal{I}$ das korrekte Ergebnis $\mathcal{O}$ liefert, oder entscheidet, dass keine Lösung existiert.

Aus dieser Definition lassen sich bereits viele interessante Eigenschaften ableiten:

1. **Finitheit:** Ein Algorithmus lässt sich in endlich vielen Worten beschreiben.
2. **Ausführbarkeit:** Jeder Schritt eines Algorithmus ist in endlicher Zeit ausführbar.
3. **Platzkomplexität:** Ein Algorithmus benötigt nur endlich viele Variablen, um Zwischenergebnisse zu speichern.
4. **Terminierung:** Ein Algorithmus darf nur endlich viele Schritte benötigen.

Darüber hinaus werden Algorithmen im Allgemeinen weiter eingeschränkt, was sie letztendlich erst praktisch nutzbar macht:

Definition 3.4: Determiniertheit

Ein Algorithmus heißt **determiniert**, wenn er für die selbe Eingabe $\mathcal{I}$ stets das gleiche Ergebnis $\mathcal{O}$ liefert.

Definition 3.5: Determinismus

Ein Algorithmus heißt **deterministisch**, wenn sich in jedem Schritt eindeutig festlegen lässt, welcher Schritt als nächstes ausgeführt wird.

In weiterer Folge werden wir implizit immer von *determinierten*, *deterministischen* Algorithmen ausgehen. Diese Algorithmen können wir für unsere Zwecke nun sehr gut verwenden, um damit möglichst große Zahlen zu erzeugen. Dafür brauchen wir eine letzte Definition:

Definition 3.6: Folge

Eine **Folge** ist eine endliche oder unendliche Liste von Elementen, die in einer bestimmten Reihenfolge angeordnet sind.

Eine Folge a wird definiert als Abbildung:

$$\begin{aligned} a : \mathbb{N} &\rightarrow \mathbb{X} \\ n &\mapsto a_n \end{aligned}$$

wobei a_n das n -te Element der Folge a und $\mathbb{X}$ eine beliebige Menge ist. Für unsere Zwecke werden wir uns auf Folgen von natürlichen Zahlen beschränken (also mit $\mathbb{X} = \mathbb{N}$). Dabei werden wir synonym auch ganz einfach von *ganzzahligen Funktionen* oder schlicht „*Funktionen*“ sprechen.

3.2 „Groß-O“-Notation

Die gesamte bisher geleistete Vorarbeit lässt bereits stark vermuten, dass wir mit simplen mathematischen Größenvergleichen von Zahlen wie $\leq$ oder $\geq$ nicht weit kommen werden. Vielmehr benötigen wir eine Methode, verschiedene „Strategien“ zur Erzeugung großer Zahlen miteinander zu vergleichen. Genauer gesagt wollen wir das Wachstumsverhalten von verschiedenen Funktionen (oder wie in unserem Fall Folgen) miteinander vergleichen. Dafür führen wir die „Groß-O“-Notation ein:

Definition 3.7: Landau-Symbole

Seien f und g zwei Folgen mit $f, g : \mathbb{N} \rightarrow \mathbb{N}$.

- **$\mathcal{O}$ -Notation (Obere Schranke):**

$$f(n) \in \mathcal{O}(g(n)) \iff \exists c > 0, n_0 \in \mathbb{N} \text{ mit } \forall n \geq n_0 : f(n) \leq c \cdot g(n)$$

Die $\mathcal{O}$ -Notation beschreibt eine obere Schranke für das asymptotische Wachstum von f .

- **Ω -Notation (Untere Schranke):**

$$f(n) \in \Omega(g(n)) \iff \exists c > 0, n_0 \in \mathbb{N} \text{ mit } \forall n \geq n_0 : f(n) \geq c \cdot g(n)$$

Die Ω -Notation beschreibt eine untere Schranke für das asymptotische Wachstum von f .

- **Θ -Notation (Exakte Schranke):**

$$f(n) \in \Theta(g(n)) \iff f(n) \in \mathcal{O}(g(n)) \text{ und } f(n) \in \Omega(g(n))$$

Die Θ -Notation beschreibt eine exakte Schranke, d.h., f wächst asymptotisch genauso schnell wie g .

Bemerkung

Es handelt sich beim $\mathcal{O}$ um ein kalligrafische O, welches ursprünglich für den griechischen Buchstaben Omikron steht. Aufgrund der Ähnlichkeit zu einem großen O in den meisten Schriftarten setzte sich jedoch die Bezeichnung „Groß-O“-Notation (engl. *Big-O-Notation*) durch.

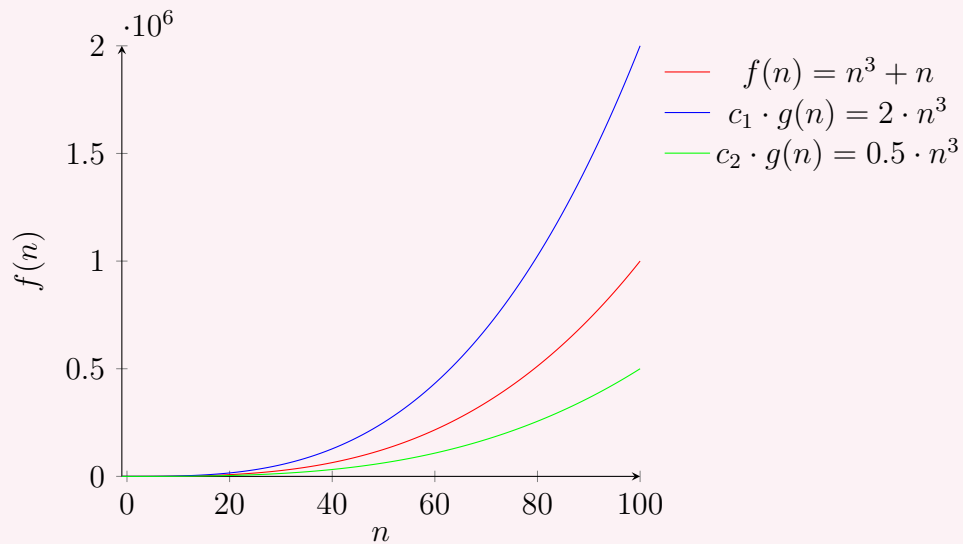
Weiters wird in der Literatur oft nicht zwischen $\mathcal{O}$ und Θ unterschieden, da für Effizienzanalysen meist nur die kleinstmögliche obere Schranke interessant ist. Es wäre demnach zwar richtig zu sagen, dass beispielsweise $n^2 \in \mathcal{O}(n^5)$, also n^5 eine obere Schranke für n^2 ist, jedoch ist dies nur wenig aussagekräftig.

Eine detailliertere Beschreibung mit einigen Beispielen zur Laufzeitbestimmung bietet das 39. Kapitel *Die Landau Symbole* aus *Konkrete Mathematik (nicht nur) für Informatiker* [Wei21].

Beispiel

Seien $f(n) = n^3 + n$ und $g(n) = n^3$.

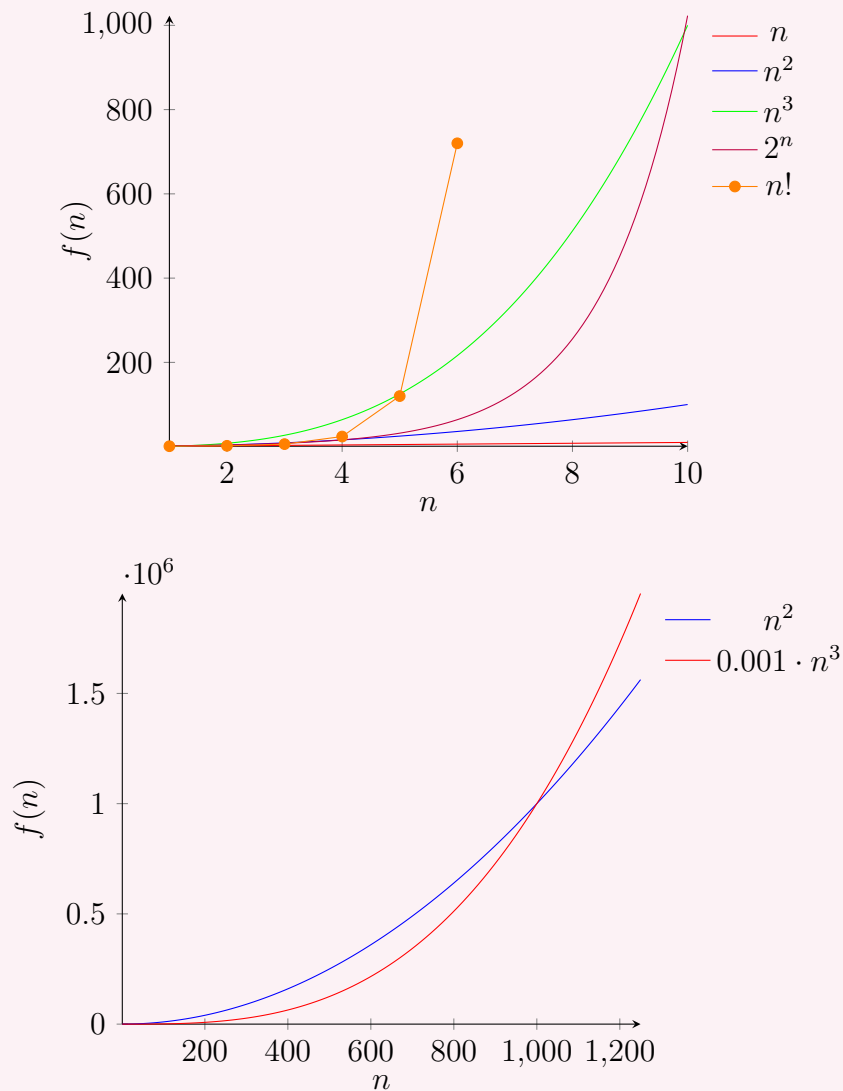
Dann finden wir mit $c_1 = 2$ und $c_2 = 0.5$ zwei Konstanten, sodass $f(n) \in \mathcal{O}(g(n))$ und $f(n) \in \Omega(g(n))$:



Damit gilt: $f(n) \in \Theta(g(n))$. Das bedeutet, dass $f(n)$ asymptotisch genauso schnell wächst wie $g(n)$. Anders formuliert könnten wir also auch sagen: $g(n) \in \Theta(f(n))$. Diese Symmetrie ist eine besondere Eigenschaft der Θ -Notation.

Beispiel

Der Graph zeigt Folgen, die jeweils verschiedene asymptotisches Wachstumsverhalten aufweisen. Das bedeutet: egal welche Koeffizienten wir für jeweils zwei Funktionen wählen würden, eine der beiden wird die andere ab einem bestimmten n abhängen.



4 Die Spielregeln

Um eine faire Basis zu schaffen, auf der wir unsere Vergleiche anstellen können, müssen wir uns auf einige Spielregeln einigen. Wir stellen uns für diesen Zweck ein Duell zwischen zwei Mathematikern vor, die versuchen einander mit immer größeren Zahlen zu übertrumpfen.

4.1 Big Number Duel

Am 26. Jänner 2007 veranstalteten Agustín Rayo, Professor am MIT, und Adam Elga, Professor an der Princeton University, das *Big Number Duel* am MIT in Cambridge. Dabei kämpften sie öffentlich darum, wer von ihnen die „größere Zahl“ an eine Tafel schreiben kann.

Rayo veröffentlichte eine Zusammenfassung des Duells über die MIT Press, woraus wir für unsere Zwecke die Spielregeln übernehmen können [Ray20].

Die Regeln definierten sie wie folgt:

1. Die Teilnehmer schreiben abwechselnd einen Ausdruck auf die Tafel, der eine natürliche Zahl beschreibt.
2. Nicht standardisierte Notation muss dem Publikum ersichtlich dargelegt werden.
3. Primitive Semantik ist nicht erlaubt. (Das bedeutet, Einreichungen wie „Die größte Zahl, die je jemand an eine Tafel geschrieben hat, plus eins“ sind nicht erlaubt.)
4. **Gentlemen's Agreement:** neue Einreichungen müssen so groß sein, dass sie mit den zuvor verwendeten Methoden nicht überboten werden können. (Das heißt, auch zur letzten Zahl etwas hinzuzuzählen oder einen Algorithmus doppelt so oft anzuwenden wäre als „unsportlich“ verboten.)

Für unsere Suche nach der größten Zahl einigen wir uns auf die Spielregeln des *Big Number Duel* und verwenden diese als Anhaltspunkt dafür, neue Zahlen zu finden.

4.2 Der Zeichenvorrat

Da wir keine echte Tafel vor uns haben, versuchen wir erst einmal zu verstehen, welchen Zweck sie beim *Big Number Duel* erfüllt. Alternativ könnte man die jeweils nächste Einreichung auch einfach dem Publikum und dem Kontrahenten verkünden. Das würde in der Theorie jedoch auch erlauben, „10 hoch 10 hoch 10 hoch 10 hoch ...“ ohne Ende aufzusagen. Das würde zwar zu jedem Zeitpunkt, an dem man aufhört, eine natürliche Zahl ergeben, aber eben nur, wenn man aufhört. Muss man die Zahl jedoch an eine Tafel schreiben, ist es nicht mehr möglich, unendlich viele Zeichen zu verwenden.

Nach dieser Idee wollen wir hier auch die Anzahl der möglichen Zeichen begrenzen. Prinzipiell macht es hier keinen Unterschied, worauf wir uns genau einigen, nur dass unser Ausdruck endlich viele Zeichen haben muss. Will man auch die Verwendung von Algorithmen zur Erzeugung großer Zahlen etwas weiter konkretisieren, stößt man bald zusätzlich auf das Problem, eine einheitliche Darstellungsform zu finden. Oft wird hier auf die sog. *Turingmaschine* zurückgegriffen.

Definition 4.1: Turingmaschine

Eine **Turingmaschine** ist ein theoretisches Berechnungsmodell, das aus folgenden Komponenten besteht:

- Einem **Band**, das in beide Richtungen unendlich lang ist und in Zellen unterteilt ist. Jede Zelle kann ein Symbol aus einem endlichen Alphabet (meist $\{\emptyset, 0, 1\}$ oder $\{0, 1\}$) enthalten.
- Einem **Lese-/Schreibkopf**, der sich entlang des Bandes bewegen kann und das Symbol in der aktuellen Zelle lesen oder überschreiben kann.
- Einem **Zustandsregister**, das den aktuellen Zustand der Maschine speichert. Es gibt eine endliche Menge von Zuständen, einschließlich eines Start- und eines Endzustands.
- Einer **Übergangsfunktion**, die bestimmt, wie die Maschine basierend auf dem aktuellen Zustand und dem gelesenen Symbol handelt. Die Übergangsfunktion spezifiziert:
 - Das neue Symbol, das in die aktuelle Zelle geschrieben wird.
 - Die Richtung, in die sich der Kopf bewegt (links, rechts oder bleibt stehen).
 - Den nächsten Zustand der Maschine.

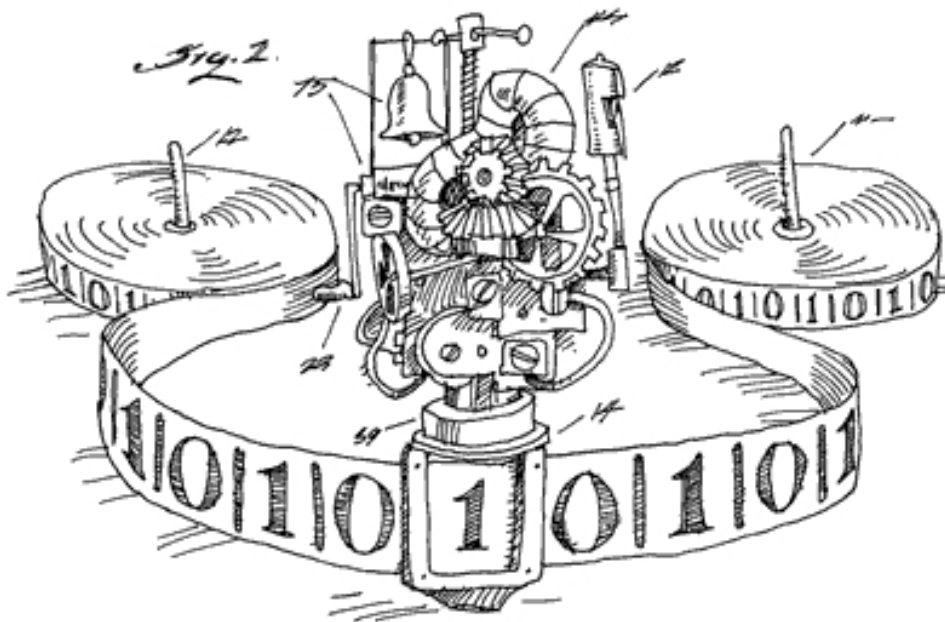


Abbildung 1: Turingmaschine, ill. Felinne Hermans

Turingmaschinen werden oft als *universelle* Maschinen bezeichnet, da sie theoretisch jede berechenbare Funktion ausführen können. Damit sind sie gleich mächtig wie alle gängigen Programmiersprachen, wodurch wir uns bestimmte Algorithmen auch alternativ in diesen Sprachen ansehen können. Es gibt jedoch einen sehr bekannten Vertreter der großen Zahlen, der durch die Einführung dieser Beschränkung sich einen „unfairen“ Vorteil zu erschleichen versucht.

4.3 Potentielle Ausnahmen

Ohne zu stark vorzugreifen werden wir uns in Kapitel 6 mit einigen sehr bekannten großen Zahlen beschäftigen, die verschiedenste Mathematiker definiert haben. Eine davon, nämlich die *Busy Beaver* Funktion $\mathcal{BB}(n)$, ist eine Funktion, die für eine Turingmaschine mit n Zuständen die maximale Anzahl an Schritten zählt, die sie ausführen kann, bevor sie stoppt, ohne dass sie für immer weiterläuft.

Da jeder für das Duell zugelassene Algorithmus ebenso mit n Zuständen auskommen muss, würde $\mathcal{BB}(n)$ größer als jede andere Zahl sein, die mit den erlaubten Algorithmen erzeugt werden kann. Das wirkt sehr ähnlich zu der Aussage „Die größte Zahl, die ich auf die Tafel schreiben kann.“ Warum das kein Totschlagargument ist und wie wir vielleicht sogar

darüber hinaus kommen, werden wir im erwähnten Kapitel betrachten.

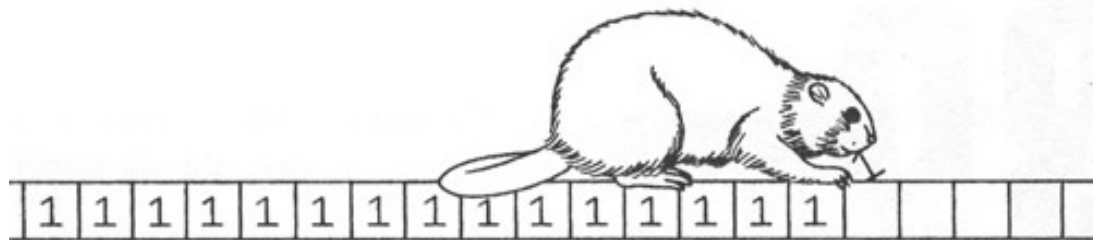


Abbildung 2: Busy Beaver, ill. A. K. Dewdney

4.4 Die Rangliste

Wir werden im nächsten Kapitel eine Vielzahl von Methoden kennenlernen, immer größere Zahlen zu entdecken. Gemäß der Regeln des *Big Number Duel* werden wir dabei nur Zahlen auf die Rangliste setzen, die eine neue Idee mit sich bringen, die anderen Zahlen in den Schatten zu stellen. Gleichzeitig werden wir uns aber nicht damit aufhalten, jede dieser Herangehensweisen an ihr Maximum zu treiben, da die nächste Zahl diese ohnehin bei Weitem übertreffen wird. Demnach spielt es für uns keine Rolle, ob wir beispielsweise die Zahl 10^2 , 10^3 oder 10^{10} auf die Rangliste setzen, da sie dieselbe Idee verfolgen. Fangen wir ganz in diesem Sinne mit der kleinsten Zahl, die wir in diesem Kapitel betrachten werden, an:

Rang	Ausdruck	Abschätzung	Beschreibung
1	?	?	?
...	...	...	...
13	1	< 10	Kleinste natürliche Zahl

5 Naive Versuche einer großen Zahl

In diesem Kapitel werden wir uns mit den ersten Versuchen beschäftigen, große Zahlen zu erzeugen. Dabei werden wir Ideen verfolgen, welche sich auf die Grundrechnungsarten stützen und so auch jedem Menschen ohne großen mathematischen Hintergrund einfallen hätten können. Auch werden wir schnell merken, dass wir schnell an die Grenzen der Darstellbarkeit stoßen werden. Deshalb werden wir im Laufe dieses Kapitels immer wieder Notationen kennenlernen, die es uns ermöglichen, Abschätzungen für diese großen Zahlen anzugeben und diese auch zu vergleichen.

5.1 Ausreizen der Grundrechnungsarten

Versuchen wir zunächst einmal, die vier Grundrechnungsarten zu verwenden, um eine möglichst große Zahl auf unserer begrenzten Tafel darzustellen. Grundsätzlich können wir uns hier gleich auf die **Addition** und **Multiplikation** beschränken, da die *Subtraktion* und *Division* in der Regel unsere Zahl nur kleiner machen.

Ausgehend von einer überschaubaren Zahl, sagen wir 3, könnten wir versuchen, diese Zahl immer wieder zu addieren und so unsere Tafel zu füllen. Für eine Tafel mit bspw. 10 Zeichen könnten wir also

$$3 + 3 + 3 + 3 + 3$$

schreiben, was in Summe 15 ergibt. Damit haben wir aber bereits eine sehr viel kompaktere Darstellung gefunden, dieselbe Zahl darzustellen. Dasselbe gilt auch für die Multiplikation, die mit

$$3 \cdot 3 \cdot 3 \cdot 3 \cdot 3$$

ausgeschöpft wäre (= 243). Wir sehen, dass wir nicht sehr weit damit kommen, die Operatoren explizit anzuführen. Wir wären in diesem Fall also sogar besser dran, die Operatoren gänzlich zu ignorieren und nur die Zahlen zu verwenden:

$$3333333333.$$

Nichtsdestotrotz haben wir hiermit 3 Methoden gefunden, unsere bisher größte Zahl 1 zu übertreffen.

Rang	Ausdruck	Abschätzung	Beschreibung
1	?	?	?
...	...	...	...
10	333333333	$> 10^9$	Füllen der Tafel
11	$3 \cdot 3 \cdot 3 \cdot 3 \cdot 3$	< 1000	Explizite Multiplikation
12	$3 + 3 + 3 + 3 + 3$	< 100	Explizite Addition
13	1	< 10	Kleinste natürliche Zahl

Um zu zeigen, dass der Ausdruck mit Multiplikation als Operator wirklich größere Zahlen erzeugt als die Addition, betrachten wir die Algorithmen für diese Operatoren.

Algorithmus 5.1: Multiplikation

```

1: Input:  $a, b \in \mathbb{N}$ 
2: Output:  $a \cdot b$ 
3:  $result \leftarrow 0$ 
4: while  $b > 0$  do
5:    $result \leftarrow result + a$ 
6:    $b \leftarrow b - 1$ 
7: end while
8: return  $result$ 

```

Will man also die Zahl n mit sich selbst multiplizieren, wird in Zeile 5 n -mal eine Addition von n ausgeführt. Damit können wir mit der *Groß-O-Notation* formal zeigen, dass die Multiplikation asymptotisch schneller wächst als die Addition:

Wir wählen als Basis eine Zahl a (in unserem Beispiel 3), für die wir uns das asymptotische Wachstum ansehen wollen.

$$\begin{aligned}
 f(n) &= a + a + a + \dots + a \\
 &= n \cdot a \\
 &\in \mathcal{O}(n)
 \end{aligned}$$

$$\begin{aligned}
 g(n) &= a \cdot a \cdot a \dots a \\
 &= a^n \\
 &\in \mathcal{O}(a^n)
 \end{aligned}$$

Damit sehen wir zwei wichtige Eigenschaften:

- Für das asymptotische Wachstum spielt es bei der Addition keine Rolle, welche Zahl wir als Basis verwenden, da wir immer eine Konstante c finden können, damit $f(n) \cdot c$ eine obere Schranke für $f(n)$ ist.
- Das asymptotische Wachstum der wiederholten Multiplikation lässt sich mit $O(a^n)$ beschreiben, wodurch wir keine Konstante c finden können, die eine obere Schranke für $g(n)$ ist. Damit wächst $g(n)$ asymptotisch schneller als $f(n)$ und hat somit in der Rangliste einen Platz weiter oben verdient.

Wer die Spielregeln gut im Kopf hat und sich jetzt fragt, warum wir nicht einfach den oben verwendeten Algorithmus anstatt der explizit ausgeschriebenen Multiplikation einreichen, ist etwas ganz Heißem auf der Spur. Wir müssen den Algorithmus nämlich nicht vergrößern, um immer größere Zahlen zu erzeugen. Gleichzeitig ist es auch möglich die danach übrigen Zeichen viel effizienter als Eingabeparameter zu verwenden. Es sei zu diesem Zeitpunkt dahingestellt, ob wir wirklich ein neues asymptotisches Wachstum erreicht haben. Um das herauszufinden, definieren wir uns zunächst einmal die **Wiederholte Selbst-Addition und Multiplikation**:

Algorithmus 5.2: Wiederholte Selbst-Addition $f_{WSA}(n)$

```
1: Input:  $n \in \mathbb{N}$ 
2: Output:  $a \in \mathbb{N}$ 
3:  $a \leftarrow 0, i \leftarrow n$ 
4: while  $i > 0$  do
5:    $a \leftarrow a + n$ 
6:    $i \leftarrow i - 1$ 
7: end while
8: return  $a$ 
```

Algorithmus 5.3: Wiederholte Selbst-Multiplikation $f_{WSM}(n)$

```
1: Input:  $n \in \mathbb{N}$ 
2: Output:  $a \in \mathbb{N}$ 
3:  $a \leftarrow 1, i \leftarrow n$ 
4: while  $i > 0$  do
5:    $a \leftarrow a \cdot n$ 
6:    $i \leftarrow i - 1$ 
7: end while
8: return  $a$ 
```

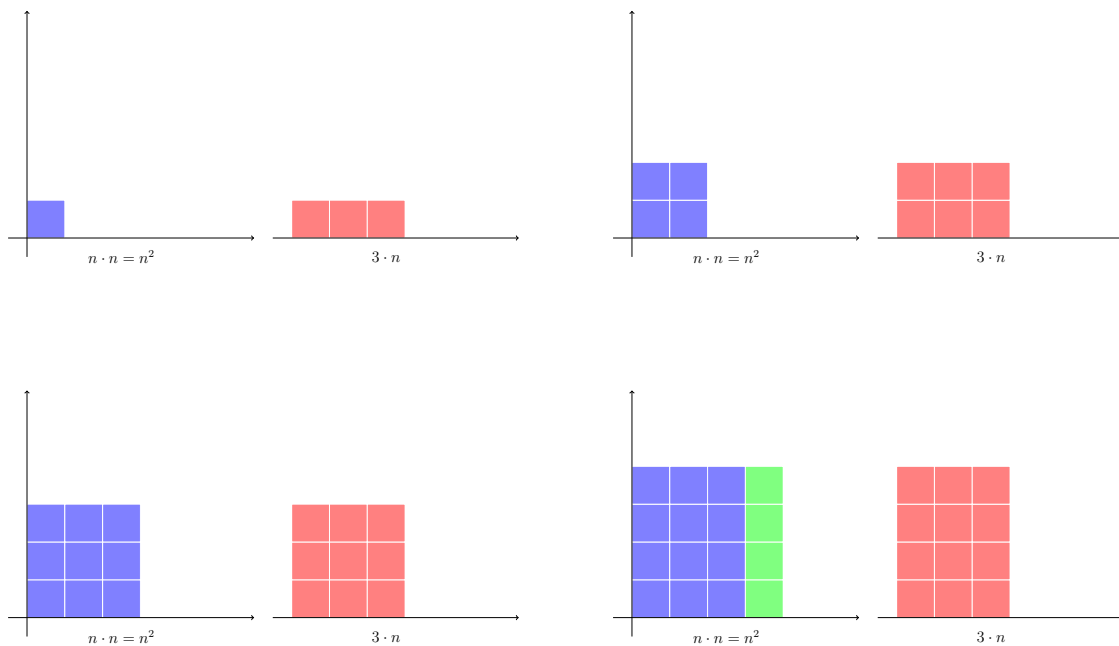
Als Parameter für Addition und Multiplikation dient in beiden Fällen die selbe Zahl n ,

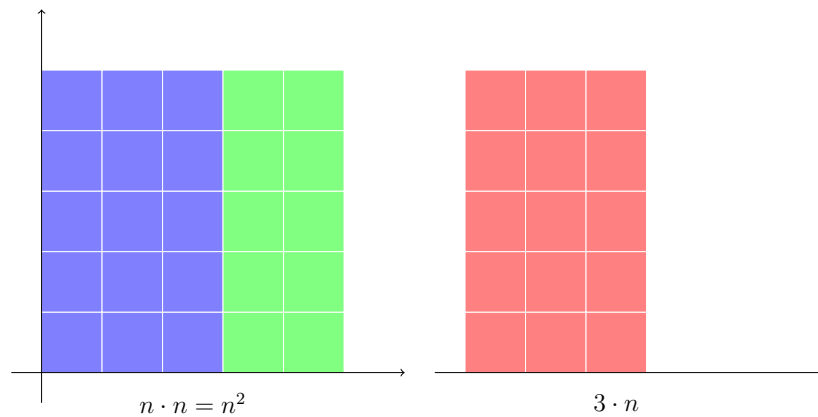
welche gleichzeitig die Anzahl der Wiederholungen angibt. Vereinfacht man die beiden Darstellungen mittels gängiger Notation fällt schnell auf, dass wir uns hier tatsächlich in einer anderen Größenordnung des Wachstumsverhaltens bewegen.

$$\text{Addition : } \underbrace{n + n + n + \cdots + n}_{n\text{-mal}} = n \cdot n = n^2$$

$$\text{Multiplikation : } \underbrace{n \cdot n \cdot n \cdots n}_{n\text{-mal}} = n^n$$

Schauen wir uns einen sehr visuellen Beweis dafür an, warum $f_{WSA}(n)$ ab einem groß genug gewähltem n die Addition mit einer fixen Basis in den Schatten stellt. Wählen wir dafür als Basis 3:



Abbildung 3: n^2 vs. $3 \cdot n$

Damit sehen wir, dass ab einer bestimmten Größenordnung von n die Addition mit einer fixen Basis nicht mehr mithalten kann, da die Basis von $f_{WSA}(n)$ immer größer wird, bis sie schließlich die fixe Basis übersteigt, und somit pro Folgenglied eine größere Zahl addiert wird. Auch formal sehen wir, dass wir keine Konstante c finden können, die eine obere Schranke für $f_{WSA}(n)$ ist, da wir die Konstante mit der Basis kombinieren können:

Angenommen wir hätten eine Konstante c gefunden, die eine obere Schranke für $f_{WSA}(n)$ mit dem Wachstum der 3 - Addition bildet, dann gilt:

$$\begin{aligned} f_{WSA}(n) &\leq c \cdot 3n \quad \forall n \geq n_0, n_0 \in \mathbb{N} \\ f_{WSA}(n) &\leq (3c)n \quad \forall n \geq n_0, n_0 \in \mathbb{N} \end{aligned}$$

Dabei wissen wir aber schon mit dem selben Argument, dass $f_{WSA}(n)$ asymptotisch schneller wächst als die Addition mit einer fixen Basis $3c$, was einen Widerspruch darstellt.

Damit können wir unsere Rangliste um die beiden neuen Einträge erweitern:

Rang	Ausdruck	Abschätzung	Beschreibung
1	?	?	?
...	...	...	...
8	$f_{WSM}(100)$	$\geq 10^{200}$	Wiederholte Selbst-Multiplikation
9	$f_{WSA}(10^6)$	$\geq 10^{12}$	Wiederholte Selbst-Addition
10	33333333	$> 10^9$	Füllen der Tafel
...	...	...	...

5.2 Ausreizen weiterer bekannter Operatoren

Die Analyse des Wachstumsverhaltens der beiden oben vorgestellten Algorithmen gibt uns schon einen kurzen Einblick darin, dass wir mit Addition und Multiplikation die gängigen Operatoren bei weitem noch nicht ausgereizt haben. Beispielsweise können wir auch die Potenzierung verwenden, um eine noch größere Zahl zu erzeugen. So würde bereits die Zahlenfolge $a_{10}(n) := 10^{(n^n)}$ sehr viel schneller wachsen, als alles andere, was wir bisher gesehen haben.

Auch die Fakultät $n! = n \cdot (n-1) \cdot (n-2) \dots 3 \cdot 2 \cdot 1$ wächst sehr viel schneller als jede einfache Potenz und weist damit eines der größten Wachstumsverhalten auf, das wir mit den gängigen Operatoren erreichen können.

Beispiel : Veranschaulichung 52!

Es gibt eine sehr bekannte absurde Methode, die Zahl 52! zu veranschaulichen. Frei übersetzt aus einem Online-Beitrag von US Data Scientist Scott Czepl [Cze]:

Eine Person umrundet die Erde entlang des Äquators mit je einem Schritt alle eine Milliarde Jahre. Nach jeder vollständigen Umrundung entfernt sie einen Tropfen Wasser aus dem Pazifik. Ist der Ozean vollständig geleert, wird ein Blatt Papier auf den Boden gelegt, der Ozean aufgefüllt, und der Prozess beginnt von neuem. Dieser Vorgang wird so oft wiederholt, bis der Papierstapel die Entfernung von der Erde zur Sonne erreicht. Doch selbst nach tausend solcher Zyklen verbleiben noch immer über $5 \cdot 10^{67}$ Sekunden (etwa ein Drittel der ursprünglichen Zeitspanne.)

Das oben genannte Beispiel lässt langsam ahnen, auf welche Größenordnungen wir uns ab sofort immer schneller zubewegen werden. Tatsächlich wächst unsere Funktion $f_{WSM}(n)$, die wir mit Wachstum $\mathcal{O}(n^n)$ abschätzen können, asymptotisch schneller als die Fakultät $n!$, was ganz einfach zu zeigen ist:

$$\begin{aligned}
 n^n &= \underbrace{n \cdot n \dots n \cdot n \cdot n}_{n\text{-mal}} \\
 n! &= n \cdot \underbrace{(n-1)}_{<n} \dots \underbrace{(3)}_{<n} \cdot \underbrace{(2)}_{<n} \cdot \underbrace{(1)}_{<n} \\
 &\quad \underbrace{\hspace{10em}}_{n\text{-mal}}
 \end{aligned}$$

Wir müssen uns hier aber natürlich nicht zurückhalten, denn wir haben noch einiges an Platz auf unserer imaginären Tafel. Alleine das erneute Anwenden der Fakultät auf die Fakultät $52!$ ergibt eine unvorstellbar große Zahl, die wir nicht einmal mehr mit einem geeigneten Gedankenspiel visualisieren können:

$$52! = 80658175170943878571660636856403766975289505440883277824000000000000$$

$$52!! = (52!)! = \text{!}$$

Bemerkung

In mancher Literatur hat die Notation $n!!$ (*Doppelte Fakultät* oder *Alternative Fakultät*) eine andere Bedeutung, die sich auf die Berechnung der Fakultät mit entweder nur geraden oder nur ungeraden Zahlen bezieht.

$$n!! = \prod_{i=0}^{\lceil \frac{n}{2} \rceil - 1} (n - 2i) = n \cdot (n - 2) \cdot (n - 4) \dots$$

In unserem Fall sei aber explizit das erneute Anwenden der Fakultät, also $n!! = (n!)!$ gemeint.

Gleichmaßen können wir auch die Potenzierung mehrmals anwenden, um einen sog. *Potenzurm* (eng. *Power Tower*) zu erzeugen. Die Reihenfolge der Potenzierung ist hier wichtig, Potenztürme werden in der Regel von „oben nach unten“ ausgewertet. Das bedeutet, dass wir zuerst die oberste Potenz ausrechnen und dann die nächste Potenz mit dem Ergebnis der vorherigen Potenz.

$$2^{3^4} = 2^{(3^4)} = 2^{81} = 2417851639229258349412352 \neq 4096 = 8^4 = (2^3)^4$$

Formal gesehen handelt es sich hier um das wiederholte Anwenden des Operators *Potenzierung*. Die Potenzierung ist aber bereits die wiederholte Multiplikation, welche wiederum eine sich ständig wiederholende Addition ist. Es handelt sich hier also um einen interessanten Zusammenhang, den wir nutzen können, um uns für unsere Zwecke sehr mächtige Operatoren (in weiterer Folge Algorithmen) zu erzeugen.

Der Potenzturm wird als Operator formal oft als *Tetration* bezeichnet und mit zwei Pfeilen nach oben dargestellt.

Definition 5.1: Tetration

Die **Tetration** ist eine Form der wiederholten Potenzierung, die wie folgt definiert ist:

$$a \uparrow\uparrow b = \underbrace{a^{a^{\dots^a}}}_{b \text{ Kopien von } a}$$

Dabei ist a die Basis und b die „Höhe“ des Potenzturms (inklusive Basis).

Bemerkung

Die Pfeil-Notation wird auf Donald Knuth zurückgeführt, der sie in den 1970er Jahren eingeführt hat (*eng. Knuth's up-arrow notation*).

Wir können ganz nach dieser Notation auch Exponenten definieren, indem wir nur einen einfachen Pfeil nach oben verwenden: $a \uparrow b = a^b$.
 $a \uparrow\uparrow b$ ist also gleichbedeutend mit $a \uparrow (a \uparrow (a \dots))$, wobei b die Anzahl der Operanden angibt.

Beispiel : Beispielwerte Tetration

- $3 \uparrow\uparrow 2 = 3 \uparrow 3 = 3^3 = 27$
- $2 \uparrow\uparrow 3 = 2 \uparrow (2 \uparrow 2) = 2 \uparrow 4 = 2^4 = 16$
- $4 \uparrow\uparrow 2 = 4 \uparrow 4 = 4^4 = 256$
- $2 \uparrow\uparrow 4 = 2 \uparrow (2 \uparrow (2 \uparrow 2)) = 2 \uparrow (2 \uparrow 4) = 2 \uparrow 16 = 2^{16} = 65536$

Warum aber sollten wir hier aufhören? Ganz ähnlich können wir die wiederholte Tetration mit drei Pfeilen nach oben definieren und mit *Pentation* $a \uparrow\uparrow\uparrow b = a \uparrow\uparrow (a \uparrow\uparrow (a \dots))$ bezeichnen. Diese Ausdrücke werden auch für bereits sehr kleine Inputs extrem groß und übersteigen alle bisherigen Einreichungen bei Weitem (auch die oben angeschnittene mehrfache Fakultät). Wir können also eine überschaubar kleine Zahl einsetzen und sie mit gutem Gewissen auf die Rangliste setzen.

Rang	Ausdruck	Abschätzung	Beschreibung
1	?	?	?
...	...	...	...
6	$2 \uparrow\uparrow\uparrow 4$	$= 2 \uparrow\uparrow 65536 = 2^{2^{\cdot^{\cdot^{\cdot^2}}}}$ (Höhe 65536)	Pentation
7	$(52!)!$	$> 10^{10^{70}}$	Doppelte Fakultät
...	...	...	...

5.3 Die Ackermann-Funktion

Würden wir die Idee, den vorherigen Operator einfach wiederholt anzuwenden weiter verfolgen, würden wir irgendwann an einen Punkt kommen, an dem unsere Pfeil-Notation nicht mehr aussagekräftig ist (*der 5000. Operator würde womöglich nicht mehr auf die Tafel passen*). Deswegen wollen wir hier versuchen, diese Idee etwas zu formalisieren und werden dabei auf die *Ackermann-Funktion* stoßen.

Wir wollen nun ausgehend von der Addition eine Funktion definieren, die rekursiv zur Berechnung den nächst kleineren Operator wiederholt anwendet. Eine modernisierte Version der Ackermann-Funktion basiert auf einer dreistelligen Funktion $\varphi(a, b, n)$, die rekursiv über die dritte Variable definiert ist:

Definition 5.2: Ackermann Funktion (historisch)

$$\begin{aligned}\varphi(a, b, 0) &= a + b \\ \varphi(a, 0, n + 1) &= \alpha(a, n) \\ \varphi(a, b + 1, n + 1) &= \varphi(a, \varphi(a, b, n + 1), n)\end{aligned}$$

Dabei ist $\alpha(a, n)$ eine Hilfsfunktion für die Definition der Basisfälle $a + 0, a \cdot 0, a^0, \dots$ mit den Startwerten:

$$\alpha(a, n) = \begin{cases} 0 & \text{wenn } n = 0 \\ 1 & \text{wenn } n = 1 \\ a & \text{wenn } n > 1 \end{cases}$$

Die Funktion φ ist so konstruiert, dass sie auf sehr einfachem Wege mit steigenden Werten von n Addition, Multiplikation, Potenzierung usw. simulieren kann. Für $n = 0$ entspricht sie

der Addition, für $n = 1$ etwa der Multiplikation, und für größere n wächst die Komplexität (und der Wert) der Funktion rasant.

Für die Wachstumsanalyse betrachtet man oft eine leicht modifizierte Version der Funktion, die als Basis immer 2 verwendet, und somit nur mehr 2 Parameter hat:

Definition 5.3: Ackermann Funktion (modifiziert)

$$\begin{aligned} A_1(n) &= 2 \cdot n, & \text{wenn } n \geq 1 \\ A_k(1) &= 2, & \text{wenn } k \geq 2 \\ A_k(n) &= A_{k-1}(A_k(n-1)), & \text{wenn } n \geq 2, k \geq 2 \end{aligned}$$

Dadurch ergibt sich der Zusammenhang: $A_k(n) = \varphi(2, n, k)$

Die Berechnung der Multiplikation $2 \cdot 3$ würde mit den jeweiligen Ackermann Funktionen wie folgt aussehen: (*Man beachte, Multiplikation hat in Ackermann Notation den Index 1*):

Beispiel : $A_1(3) = 2 \cdot 3$

$$\begin{aligned} \varphi(2, 3, 1) &= \varphi(2, \varphi(2, 2, 1), 0) \\ &= \varphi(2, \varphi(2, \varphi(2, 1, 1), 0), 0) \\ &= \varphi(2, \varphi(2, \varphi(2, \varphi(2, 0, 1), 0), 0), 0) \\ &= \varphi(2, \varphi(2, \varphi(2, 0, 0), 0), 0) \\ &= \varphi(2, \varphi(2, 2, 0), 0) \\ &= \varphi(2, 4, 0) \\ &= 6 \end{aligned}$$

$$\begin{aligned} A_1(3) &= 2 \cdot 3 \\ &= 6 \end{aligned}$$

Beispiel : $A_2(3) = 2^3$

$$\begin{aligned}
 \varphi(2, 3, 2) &= \varphi(2, \varphi(2, 2, 2), 1) \\
 &= \varphi(2, \varphi(2, \varphi(2, 1, 2), 1), 1) \\
 &= \varphi(2, \varphi(2, \varphi(2, \varphi(2, 0, 2), 1), 1), 1) \\
 &= \varphi(2, \varphi(2, \varphi(2, 1, 1), 1), 1) \\
 &= \varphi(2, \varphi(2, \varphi(2, \varphi(2, 0, 1), 0), 1), 1) \\
 &= \varphi(2, \varphi(2, \varphi(2, 0, 0), 1), 1) \\
 &= \varphi(2, \varphi(2, 2, 1), 1) \\
 &= \varphi(2, \varphi(2, \varphi(2, 1, 1), 0), 1) \\
 &= \varphi(2, \varphi(2, \varphi(2, \varphi(2, 0, 1), 0), 0), 1) \\
 &= \varphi(2, \varphi(2, \varphi(2, 0, 0), 0), 1) \\
 &= \varphi(2, \varphi(2, 2, 0), 1) \\
 &= \varphi(2, 4, 1) \\
 &= \varphi(2, \varphi(2, 3, 1), 0) \\
 &= \varphi(2, \varphi(2, \varphi(2, 2, 1), 0), 0) \\
 &= \varphi(2, \varphi(2, \varphi(2, \varphi(2, 1, 1), 0), 0), 0) \\
 &= \varphi(2, \varphi(2, \varphi(2, \varphi(2, \varphi(2, 0, 1), 0), 0), 0), 0) \\
 &= \varphi(2, \varphi(2, \varphi(2, \varphi(2, 0, 0), 0), 0), 0) \\
 &= \varphi(2, \varphi(2, \varphi(2, 2, 0), 0), 0) \\
 &= \varphi(2, \varphi(2, 4, 0), 0) \\
 &= \varphi(2, 6, 0) \\
 &= 8
 \end{aligned}$$

$$\begin{aligned}
 A_2(3) &= A_1(A_2(2)) \\
 &= A_1(A_1(A_2(1))) \\
 &= A_1(A_1(2)) \\
 &= A_1(4) \\
 &= 8
 \end{aligned}$$

Wie wir im Beispiel oben sehen, steigt die Anzahl der Rekursionsschritte sehr stark an, sobald wir k erhöhen. Für $k = 3$ und $n = 5$ ergibt sich bereits eine sehr große Zahl, für die wir die Rekursionsgleichung nicht mehr sinnvoll aufschreiben können:

$$A_3(5) = 2^{2^{2^2}} = 2^{65536} \text{ (Zahl mit 19729 Ziffern)}$$

Durch die Rekursion $A_4(5) = A_3(A_4(4))$ entsteht ein Potenzturm mit nicht mehr nur 5 Gliedern, sondern $A_4(4)$ vielen Gliedern, was an sich schon eine unfassbar große Zahl ist. Wir sind hier durch die rekursive Definition anstelle von expliziter Pfeil Notation in der Lage, den Index des Operators nach Belieben zu erhöhen. Setzen wir also unsere Parameter auf $k = 6$ und $n = 6$ und fügen diese enorme Zahl guten Gewissens auf die Rangliste:

Rang	Ausdruck	Abschätzung	Beschreibung
1	?	?	?
...	...	...	...
5	$A_6(6)$	$= \underbrace{6 \uparrow\uparrow\uparrow (6 \uparrow\uparrow\uparrow (6 \dots))}_{\text{6-fache Anwendung der Pentation}}$	Ackermann Funktion
6	$2 \uparrow\uparrow\uparrow 4$	$= 2 \uparrow\uparrow 65536 = 2^{2^{\dots^2}} \quad (H\ddot{o}he\ 65536)$	Pentation
...	...	...	...

6 Bekannte Vertreter der absurd großen Zahlen

In diesem Kapitel werden wir uns mit einigen der bekanntesten und beeindruckendsten Zahlen beschäftigen, die in der Mathematik definiert wurden. Diese Zahlen sind so groß, dass sie unsere Vorstellungskraft weit übersteigen und selbst mit den mächtigsten Notationen und Algorithmen kaum darstellbar sind. Viele von ihnen wurden aus der Begeisterung für ihre enorme Größe erdacht und haben eine faszinierende Geschichte, manchmal haben sie aber sogar eine Verbindung zu realen Problemen in der Mathematik oder Informatik.

Für diese Arbeit werden wir uns mit vier der bekanntesten Vertreter der großen Zahlen auseinandersetzen. Wir beginnen mit *Graham's Number*, einer Zahl, die ursprünglich in der *Ramsey-Theorie* auftauchte und durch ihre schiere Größe bekannt wurde. Anschließend betrachten wir $TREE(3)$, eine Zahl die aus der Anwendung des *Satzes von Kruskal* entsteht. Danach wenden wir uns Rayo's Number zu, einer Zahl, die im Rahmen des eingangs vorgestellten *Big Number Duel* eingeführt wurde und die Grenzen der Notation auslotet. Schließen werden wir mit der auch bereits kurz angeschnittenen Busy Beaver-Funktion, die wir im Kontext der *Berechenbarkeit* betrachten werden.

Jede dieser Zahlen hat ihre eigene Geschichte und Bedeutung, und wir werden uns mit den zugrunde liegenden Konzepten und ihrer Relevanz für die Mathematik und Informatik auseinandersetzen.

6.1 Graham's Number

Die folgenden Informationen stammen aus einem Interview mit Ronald Graham selbst, das am 21.04.2014 veröffentlicht wurde [GN14].

Graham's Number ist historisch gesehen ein besonders interessanter Vertreter der großen Zahlen, da sie lange Zeit als die größte Zahl galt, die jemals in einem mathematischen Beweis verwendet wurde. Sie wurde von dem Mathematiker Ronald Graham in den 1970er Jahren eingeführt und ist das Ergebnis eines Problems in der *Ramsey-Theorie*. Die Ramsey-Theorie, benannt nach Frank Plumpton Ramsey, ist ein Teilgebiet der Kombinatorik innerhalb der Diskreten Mathematik. Sie untersucht, wie viele Elemente einer strukturierten Menge ausgewählt werden müssen, damit in jeder Teilmenge eine bestimmte Eigenschaft oder Struktur garantiert auftritt.

Ein klassisches Beispiel ist das Schubfachprinzip, das besagt, dass beim Verteilen von $k + 1$ Objekten auf k Schubfächer mindestens ein Schubfach mindestens zwei Objekte enthalten muss. Ein weiteres Beispiel kann sehr anschaulich auf Freundschaften bezogen werden: Treffen sechs Personen aufeinander, wobei jede Beziehung zwischen zwei Personen entweder eine Freundschaft oder keine Freundschaft ist, so gibt es immer entweder eine

Dreiergruppe, in der alle miteinander befreundet sind, oder eine Dreiergruppe, in der niemand miteinander befreundet ist.

Für die Verbindung zu *Graham's Number* betrachten wir den Graphen eines Würfels mit n Dimensionen (*d.h. explizit nicht nur Würfel in drei Dimensionen*), bei dem jeder Knoten eine Kante mit jedem anderen Knoten hat. Wir nennen so einen Graphen „vollständig“. Betrachten wir nur ausgewählte Knoten und deren Kanten, so nennen wir den herausgenommenen Teil einen *Teilgraph*. Liegt dieser Teilgraph (aufgefasst als geometrisches Objekt) dazu noch in einer einzigen Ebene, so nennen wir ihn *planar*.

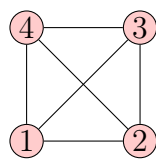


Abbildung 4: Vollständiger $n = 2$ Graph
(ist gleichzeitig planar)

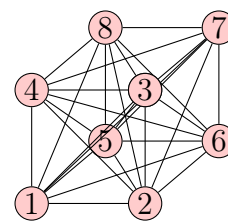


Abbildung 5: Vollständiger $n = 3$ Graph

Bemerkung

In der Literatur werden vollständige Graphen üblicherweise nicht mit ihrer Dimension bezeichnet, sondern mit der Anzahl ihrer Knoten. Ein vollständiger Graph mit n Knoten wird als K_n bezeichnet. Der vollständige „ n -dimensionale Würfelgraph“ wäre demnach für Dimension $n = 2$ der Graph $K_{2^2} = K_4$ und für Dimension $n = 3$ der Graph $K_{2^3} = K_8$.

Das Problem der Ramsey-Theorie, das Graham in seiner Arbeit behandelt hat, besteht darin, die Kanten eines vollständigen Graphen in n Dimensionen so zu färben, dass es keine monochromatischen (*d.h. einfärbigen*) vollständigen Teilgraphen gibt, die planar sind. Gesucht ist hierbei die kleinste Dimension, in der ein solcher Teilgraph in jeder möglichen Färbung auftritt, falls dies überhaupt in einer Dimension der Fall ist.

Beispiel : Verbotene Färbung bei $n = 3$ 

Für die Fälle $n = 2$ und $n = 3$ ist es jedoch ganz einfach möglich, eine Färbung zu finden, die keinen monochromatischen planaren Teilgraphen enthält. Tatsächlich wurde für alle Fälle bis hin zu $n = 13$ (gezeigt von Jerome Barkley in 2008) bewiesen, dass monochromatische planare Teilgraphen vermieden werden können. Eine obere Grenze, also eine Dimension, in der ein solcher Teilgraph garantiert auftritt, lieferte eine Veröffentlichung des amerikanischen Mathematikers Martin Gardner, in der er einen von Graham unveröffentlichten Beweis für eine so absurd große Obergrenze fand, dass er davon ausging, dass es die größte Zahl sein müsse, die jemals in einem mathematischen Beweis verwendet wurde. Diese Zahl ist nun als *Graham's Number* bekannt geworden und wird mit G_{64} (manchmal einfach G) abgekürzt.

Für die Definition von Graham's Number verwenden wir die bereits eingeführte Pfeildarstellung von Knuth, die wir rekursiv an die Spitze treiben:

Definition 6.1: Graham's Number

Sei $a \uparrow^n b$ die kurze Notation für $a \underbrace{\uparrow \uparrow \uparrow \dots \uparrow}_{n \text{ mal}} b$ mit n Pfeilen.

Dann nennen wir mit $G_0 = 4$ und $G_{n+1} = 3 \uparrow^{G_n} 3$ für $n \geq 0$ die Zahl G_{64} **Graham's Number**.

Diese Definition ist es wert, sie noch einmal ausführlicher zu betrachten. Wir haben bereits ein sehr gutes Verständnis dafür wie groß $3 \uparrow \uparrow \uparrow 3$ ist (= Pentationsoperator). Wir beginnen hier aber für die erste rekursive Zahl G_1 jedoch bereits mit 4 Pfeilen. Noch einmal zum Vergleich, $3 \uparrow \uparrow \uparrow 3 = 3 \uparrow \uparrow (3 \uparrow \uparrow 3) = 3 \uparrow \uparrow 7.625.597.484.987$, was einen Potenzurm mit 7.625.597.484.987 Gliedern ergibt. G_1 können wir daraufhin gar nicht mehr sinnvoll in so einer Schreibweise angeben. Wir definieren daraufhin als $G_2 = 3 \uparrow^{G_1} 3$. Jeder einzelne hinzugefügte Pfeil hat die resultierende Zahl um ein unvorstellbares Maß vergrößert. Jetzt fügen wir aber so viele Pfeile hinzu, dass wir deren Anzahl nicht mehr sinnvoll angeben

können. Die deutsche Sprache verliert zu diesem Zeitpunkt die Fähigkeit, der unvorstellbaren Größe dieser Zahl gerecht zu werden. Wir sind aber noch nicht fertig. Wir definieren auf die selbe Art $G_3, G_4, \dots$ bis G_{64} . Wir haben zu diesem Zeitpunkt keine Chance mehr, irgendwelche Vergleiche zu G_{64} zu ziehen, alles was uns bleibt ist diese rekursive Definition. Es ist also sehr verständlich, dass der Mathematiker Gardner annahm, dass dies die größte je verwendete Zahl in einem mathematischen Beweis sei.

Rang	Ausdruck	Abschätzung	Beschreibung
1	?	?	?
...	...	...	...
4	G_{64}	!	Graham's Number
5	$A_6(6)$	$=$ $\underbrace{6 \uparrow\uparrow\uparrow (6 \uparrow\uparrow\uparrow (6 \dots))}_{\text{6-fache Anwendung der Pentation}}$	Ackermann Funktion
...	...	...	...

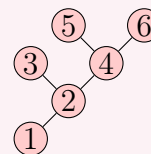
6.2 TREE(3)

Wie der Name der Funktion bereits andeutet, handelt es sich bei $TREE(3)$ um eine Funktion, die sich mit der Analyse von Bäumen beschäftigt. Sie entsteht aus der Anwendung des *Satzes von Kruskal* über die „Ordnung“ dieser graphentheoretischen Bäume. Wir werden uns hier nur mit den nötigsten Definitionen beschäftigen, um zu verstehen, wie die Glieder der $TREE$ -Funktion entstehen. Das bedeutet im speziellen auch, dass wir nicht weiter auf die Details des Satzes von Kruskal eingehen werden, sondern uns nur mit den Eigenschaften der Bäume beschäftigen werden.

Definition 6.2: Baum

Ein **Baum** ist ein zusammenhängender, ungerichteter Graph, in welchem zwei Knoten **durch genau einen Pfad** verbunden sind.

Beispiel : Baum



Die Folge $TREE(n)$ gibt die maximale Anzahl an Bäumen an, die wir mit einer bestimmten Vorschrift konstruieren können. Um diese Vorschrift zu verstehen, müssen wir noch den Begriff des *Nachfolgers* in einem Baum definieren:

Definition 6.3: Nachfolger und Einbettung

Gegeben sei ein Baum T mit einer Wurzel und Knoten v, w . Wir nennen w einen **Nachfolger** von v , wenn der eindeutige Pfad von der Wurzel zu w den Knoten v enthält. Wir nennen w einen **direkten Nachfolger** von v , wenn zusätzlich der Pfad von v zu w keinen weiteren Knoten enthält.

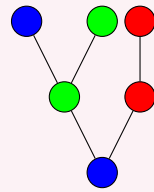
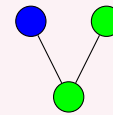
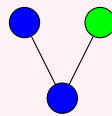
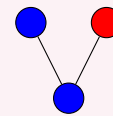
Sei X eine halbgeordnete Menge. Wenn T_1, T_2 Wurzelbäume mit Knoten sind, die in X beschriftet sind, dann sagen wir, dass T_1 **einbettbar** (eng. inf-embeddable) in T_2 ist und schreiben $T_1 \leq T_2$, falls es eine injektive Abbildung F von den Knoten von T_1 zu den Knoten von T_2 gibt, sodass:

1. *(Für alle Knoten v von T_1 die Beschriftung von v der Beschriftung von $F(v)$ in der Ordnung von X vorangeht.) - Formalität für Satz von Kruskal*
2. Wenn w ein Nachfolger von v in T_1 ist, dann ist $F(w)$ ein Nachfolger von $F(v)$ in T_2 .
3. Wenn w_1, w_2 zwei verschiedene direkte Nachfolger von v in T_1 sind, dann enthält der Pfad von $F(w_1)$ nach $F(w_2)$ in T_2 den Knoten $F(v)$.

Bemerkung

Die „Beschriftung“ der Knoten in einem Baum ist hier nicht zu verwechseln mit der „Färbung“ der Knoten, die wir bereits aus dem Graphenbeispiel kennen. Die Beschriftung ist hier einfach eine eindeutige Zuordnung von Knoten zu Elementen einer halbgeordneten Menge X . Diese Beschriftung ist für die formale Definition des Satzes von Kruskal und dessen Folgerung nötig, ist aber für das Verständnis der Konstruktion der Bäume in $TREE(n)$ unbedingt erforderlich.

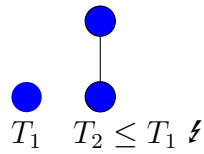
Beispiel : Baum Einbettung

 T  $T_1 \leq T$  $T_2 \not\leq T$  $T_3 \leq T$

Wir wollen nun für die Folge $TREE(n)$ die maximale Anzahl an Bäumen angeben, die wir nacheinander konstruieren können, ohne dass sich ein vorangegangener Baum in einem späteren Baum einbettet. Dazu gibt uns die Zahl n die Menge der möglichen Knoten vor, die wir verwenden können (vgl. mögliche Farben der Knoten). Als letzte Regel legen wir fest, dass der i -te Baum T_i maximal i Knoten haben darf.

 $TREE(1)$:Mögliche Knoten: {  }

Wir sind hier sehr eingeschränkt, welche Bäume wir konstruieren können. Der erste Baum darf aber ohnehin nur einen Knoten besitzen, also konstruieren wir diesen. Für den zweiten Baum haben wir daraufhin 2 Möglichkeiten. Entweder wir verwenden den ersten Knoten wieder oder wir verwenden einen zweiten Knoten. In beiden Fällen kann der erste Baum aber in den zweiten eingebettet werden. Das bedeutet, dass wir nur einen Baum konstruieren können.

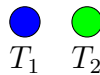


Damit ergibt sich für $TREE(1) = 1$.

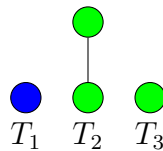
$TREE(2)$:

Mögliche Knoten: {   }

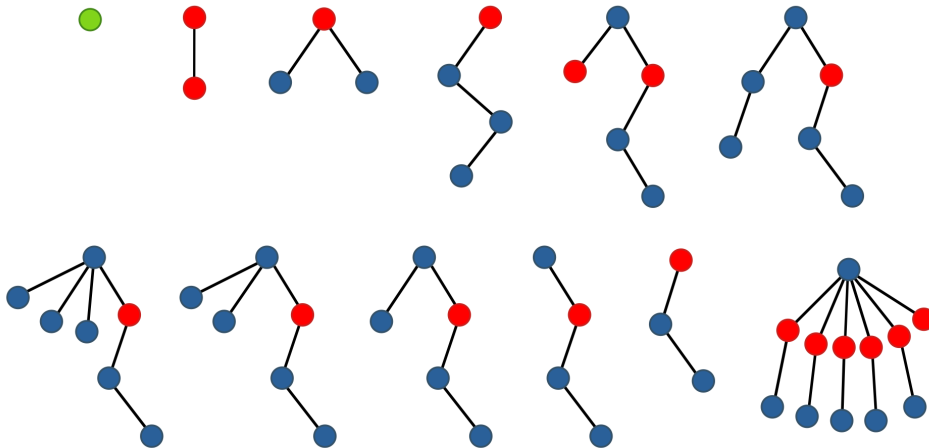
Wir haben jetzt 2 Knoten zur Verfügung, die wir verwenden können. Der erste Baum kann wieder nur einen Knoten haben. Wir haben jetzt aber eine Möglichkeit, einen zweiten Baum zu konstruieren, ohne dass der vorhergehende Baum in ihn eingebettet werden kann, da wir eine weitere Farbe zur Verfügung haben.



Hier stellt sich noch die Frage, ob wir eine länger Folge an Bäumen finden können. Dazu rufen wir uns noch einmal in Erinnerung, dass es verboten ist, dass ein **vorhergehender** Baum in einen **späteren** Baum eingebettet werden kann. Diese Einschränkung gilt aber nicht für die umgekehrte Richtung. Wir können also auch die folgende Baumfolge konstruieren:



Da weder $T_1 \leq T_2$ noch $T_2 \leq T_3$ gilt, haben wir hier eine gültige Baumfolge gefunden. Das ist dann tatsächlich auch die längste Folge, die wir konstruieren können, und damit ergibt sich für $TREE(2) = 3$.

Abbildung 6: Beispiel Sequenz-Start $TREE(3)$ ill. LightbulbMEOW

Rang	Ausdruck	Abschätzung	Beschreibung
1	?	?	?
2	?	...	...
3	$TREE(3)$	$\lll$	Baumfolge
4	G_{64}	$\ll$	Graham's Number
...	...	...	...

6.3 Busy Beaver und Rayo's Number

Wir sollten uns abschließend noch mit zwei weiteren Zahlen beschäftigen, die für unsere Rangliste und auch historisch für das *Big Number Duel* von Bedeutung sind. Es handelt sich hierbei um zwei sehr ähnliche Ideen, wobei wir die erste davon, nämlich den *Busy Beaver*, bereits kurz im Kontext der *Turing Maschine* erwähnt haben. Definieren wir sie also hier formal:

Definition 6.4: Busy Beaver

Sei $\mathbf{BB}(n)$ die größte Anzahl an Schritten, die eine **Turingmaschine** mit n Zuständen und einem Band mit unendlich vielen leeren Feldern ausführen kann, bevor sie stoppt.

Diese Definition wirkt zuerst so, also würde sie gegen die Regel verstoßen, die verbietet, keine semantischen Tricks zu verwenden. Immerhin wissen wir per Definition, dass alle Algorithmen, die wir formulieren können, auf einer Turing Maschine ausgeführt werden können, gegeben sie hat genügend verschiedene Zustände und der Algorithmus an sich ist „berechenbar“. Ein Beispiel für einen nicht berechenbaren Algorithmus wäre eine Vorschrift, die sich in einer Endlosschleife befindet und somit nie ein Ergebnis liefert. Herauszufinden, ob ein Algorithmus berechenbar ist, ist eines der bekanntesten Probleme der Informatik und wird auch als *Halting Problem* bezeichnet. Es ist bewiesen, dass es keine Turingmaschine gibt, die für alle Algorithmen entscheiden kann, ob sie in einer Endlosschleife stecken bleiben oder nicht. Daraus abgeleitet wissen wir, dass wir auch keine allgemeine berechenbare Obergrenze für unsere Busy Beaver Funktion angeben können.

Der springende Punkt für unsere Zwecke ist die Implikation daraus, dass es keine *berechenbare* Funktion gibt, die asymptotisch schneller wächst als $BB(n)$. Damit haben wir eine Folge gefunden, die schneller wächst als alle Algorithmen, die wir formulieren könnten, welche uns große Zahlen liefern. Damit haben wir eine erste Grenze für das Wachstum von Zahlen gefunden, da alle bis jetzt betrachteten Zahlen auf berechenbare Algorithmen zurückzuführen sind.

Man könnte hier argumentieren, dass wir hier wieder einen semantischen Trick verwenden, da wir ja auch die Busy Beaver Funktion nicht berechnen können, sondern nur definieren, dass seine Glieder existieren. Tatsächlich wurde der Busy Beaver im *Big Number Duel* aber aufgebracht und als gültige Einreichung akzeptiert, weshalb wir uns hier anschließen wollen. Rayo stellte im Rahmen des Duells ohnehin eine noch schneller wachsende Folge vor, indem er sich über die Verwendung von Turingmaschinen hinausbewegt hat. Diese Zahl, die aus dieser Funktion entsteht, hat daraufhin den Wettkampf entschieden und ist seither als *Rayo's Number* bekannt.

Definition 6.5: Rayo's Number

Frei übersetzt nach der Definition, die Rayo auf seiner Website zum *Big Number Duel* angibt:

*Die Gewinner-Einreichung nutzte einen Trick aus der Philosophie. Es handelt sich um eine Version von: Die kleinste Zahl, die größer ist als jede endliche Zahl, die durch einen Ausdruck in der Sprache der **Mengenlehre mit höchstens einem Googol Symbolen** benannt wird. Dabei wurde der primitive semantische Begriff „benannt“ durch eine Formel der Mengenlehre zweiter Ordnung ersetzt, die keinen primitiven semantischen Wortschatz enthält, um die oben genannte Einschränkung (=Verbot von Semantik im Regelwerk) zu umgehen.*

Wir schreiben Rayo's Number üblicherweise als **Rayo(10^{100})**.

Bemerkung

Ein *Googol* ist der gängige Name für die Zahl 10^{100} , also eine Eins gefolgt von 100 Nullen. Der Name einer identisch klingenden Suchmaschine wird auf diese Zahl zurückgeführt.

Die Vorschrift, aus der Rayo's Number entsteht, ist, ähnlich wie die Busy Beaver Funktion, nicht berechenbar. Tatsächlich gibt es gute heuristische Argumente dafür, dass wir $BB(n)$ mithilfe von Mengenlehre und ein paar Millionen Zeichen definieren können. Für Rayo's Number haben wir aber sehr viel mehr Zeichen zur Verfügung, weshalb sehr überzeugend anzunehmen ist, dass $Rayo(10^{100}) \gg \gg BB(10^{100})$ gilt. Rayo hat also durch das Hinwegsetzen über Algorithmen eine mächtigere Ausdrucksform für große Zahlen gefunden, wodurch er letztendlich das Duell für sich entscheiden konnte. Damit ergänzen auch wir unsere Rangliste abschließend mit diesen beiden enormen Zahlen:

Rang	Ausdruck	Abschätzung	Beschreibung
1	$Rayo(10^{100})$	Unberechenbar!	Rayo's Number
2	$BB(10^{100})$	Unberechenbar!	Busy Beaver
3	$TREE(3)$	$\aleph_3$	Baumfolge
...	...	...	...

7 Fazit

Beginnend mit den Grundrechnungsarten und einfachen Algorithmen haben wir uns schrittweise immer mächtigeren Konzepten angenähert, die es uns ermöglichten, Zahlen von unvorstellbarer Größe zu definieren. Wir haben gesehen, wie die Einführung neuer Operatoren wie der Tetration oder der Ackermann-Funktion das Wachstum von Zahlen exponentiell beschleunigt. Mit Graham's Number, $TREE(3)$ und der Busy Beaver-Funktion haben wir einige der bekanntesten Vertreter absurd großer Zahlen betrachtet und mit Rayo's Number eine Zahl kennengelernt, die die Frage aufwirft, was es bedeutet, eine Zahl zu definieren. Die Beschäftigung mit diesen großen Zahlen ist nicht nur ein intellektuelles Abenteuer, sondern auch ein Weg, die fundamentalen Grenzen unseres Wissens und unserer Werkzeuge zu erforschen.

Rang	Ausdruck	Abschätzung	Beschreibung
1	$Rayo(10^{100})$	Unberechenbar!	Rayo's Number
2	$BB(10^{100})$	Unberechenbar!	Busy Beaver
3	$TREE(3)$	$\gg$	Baumfolge
4	G_{64}	$\nearrow$	Graham's Number
5	$A_6(6)$	$=$ $\underbrace{6 \uparrow\uparrow\uparrow (6 \uparrow\uparrow\uparrow (6 \dots))}_{\text{6-fache Anwendung der Pentation}}$	Ackermann Funktion
6	$2 \uparrow\uparrow\uparrow 4$	$= 2 \uparrow\uparrow 65536 =$ $2^{2^{\dots^2}}_{\cdot 2} \text{ (Höhe 65536)}$	Pentation
7	$(52!)!$	$> 10^{10^{70}}$	Doppelte Fakultät
8	$f_{WSM}(100)$	$\geq 10^{200}$	Wiederholte Selbst-Multiplikation
9	$f_{WSA}(1000000)$	$\geq 10^{12}$	Wiederholte Selbst-Addition
10	333333333	$> 10^9$	Füllen der Tafel
11	$3 \cdot 3 \cdot 3 \cdot 3 \cdot 3$	< 1000	Explizite Multiplikation
12	$3 + 3 + 3 + 3 + 3$	< 100	Explizite Addition
13	1	< 10	Kleinste natürliche Zahl

Literatur

- [Cor+09] Thomas H. Cormen u. a. *Introduction to algorithms*. MIT Press, 2009.
- [Cze] Scott Czepiel. *52cards*. Accessed: 2025-04-14. URL: <https://czep.net/weblog/52cards.html>.
- [GN14] Ronald L. Graham und Numberphile. *What is Graham's Number? (feat Ron Graham)*. Accessed: 2025-04-19. 2014. URL: <https://www.youtube.com/watch?v=HX8bihEe3nA>.
- [Gri13] Peter Gritzmann. *Grundlagen der Mathematischen Optimierung - Diskrete Strukturen, Komplexitätstheorie, Konvexitätstheorie, Lineare Optimierung, Simplex-Algorithmus, Dualität*. Berlin Heidelberg New York: Springer-Verlag, 2013. ISBN: 978-3-834-82011-2.
- [Ray20] Agustín Rayo. *The Big Number Duel*. Accessed: 2025-04-12. 2020. URL: <http://web.mit.edu/arayo/www/brink-duel.pdf>.
- [Wei21] Edmund Weitz. „Die Landau-Symbole“. In: *Konkrete Mathematik (nicht nur) für Informatiker: Mit vielen Grafiken und Algorithmen in Python*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2021, S. 479–494. ISBN: 978-3-662-62618-4. DOI: 10.1007/978-3-662-62618-4_39. URL: https://doi.org/10.1007/978-3-662-62618-4_39.