

Algorithmen kapiieren

Visuell lernen und verstehen

» Hier geht's
direkt
zum Buch

DIE LESEPROBE



In diesem Kapitel:

- Die Grundlagen für das Verständnis dieses Buchs.
- Du wirst einen ersten Suchalgorithmus programmieren (eine binäre Suche).
- Du wirst erfahren, wie man die Laufzeit eines Algorithmus mit der Landau-Notation beschreibt.

1.1 Einführung

Ein *Algorithmus* ist eine Reihe von Anweisungen, die eine Aufgabe ausführen. Man könnte eigentlich jeden Codeschnipsel als Algorithmus bezeichnen, aber dieses Buch befasst sich mit den interessanteren Aspekten. Die Algorithmen in diesem Buch habe ich ausgewählt, weil sie schnell sind, interessante Aufgabenstellungen lösen oder beides. Hier sind einige der Highlights:

- Dieses Kapitel beschreibt die binäre Suche und führt vor, wie ein Algorithmus deinen Code beschleunigen kann. In einem der Beispiele wird die Anzahl der erforderlichen Schritte von 4 Milliarden auf nur 32 reduziert!
- Ein GPS-Empfänger verwendet Graphenalgorithmus (die in Kapitel 6, Kapitel 9 und Kapitel 10 erörtert werden), um die kürzeste Route zu einem Ziel zu berechnen.
- Du kannst die dynamische Programmierung (siehe Kapitel 11) dazu verwenden, einen Algorithmus zu schreiben, der Dame spielt.

Ich werde hierfür jeweils einen Algorithmus erläutern und ein Beispiel dafür zeigen. Anschließend betrachten wir die Laufzeit des Algorithmus mithilfe der Landau-Notation (Landau-Symbole). Und schließlich erfährst du, welche anderen Aufgabenstellungen mit demselben Algorithmus gelöst werden können.

1.1.1 Performance

Die gute Nachricht ist, dass Implementierungen der Algorithmen, die in diesem Buch beschrieben werden, sehr wahrscheinlich in deiner Lieblingsprogrammiersprache verfügbar sind. Du musst die Algorithmen also nicht alle selbst schreiben! Allerdings sind diese Implementierungen nutzlos, wenn du deren Vor- und Nachteile nicht verstehst. In diesem Buch lernst du, die Vor- und Nachteile verschiedener Algorithmen miteinander zu vergleichen: Solltest du für eine bestimmte Aufgabe Mergesort oder lieber Quicksort verwenden? Ist ein Array oder eine Liste besser geeignet? Die Verwendung einer anderen Datenstruktur kann einen sehr großen Unterschied ausmachen.

1.1.2 Problemlösungen

Du wirst zudem Verfahren zur Problemlösung für Aufgaben kennenlernen, an die du dich bislang vielleicht nicht herangetraut hast. Einige Beispiele:

- Falls du Videospiele magst, kannst du ein System für die Künstliche Intelligenz (KI) programmieren, das Graphenalgorithmen verwendet und das den User im Spiel verfolgt.
- Du wirst erfahren, wie du mit dem k-Nächste-Nachbarn-Algorithmus (k-Nearest-Neighbors-Algorithmus) ein Empfehlungssystem entwickeln kannst.
- Manche Aufgaben lassen sich nicht in angemessener kurzer Zeit lösen. Der Abschnitt des Buchs über NP-vollständige Probleme beschreibt, wie du solche Probleme erkennen kannst und stellt einen Algorithmus vor, der dir eine Näherungslösung liefert.

Allgemeiner formuliert: Nach der Lektüre dieses Buchs werden dir einige der meisten verbreiteten und für sehr viele Fälle anwendbaren Algorithmen vertraut sein. Mit dem Wissen aus diesem Buch kannst du dich spezielleren Algorithmen für die KI, für Datenbanken usw. zuwenden oder dich noch größeren Herausforderungen stellen.

Erforderliche Kenntnisse

Für die weitere Lektüre des Buchs benötigst du Grundkenntnisse der Algebra. Betrachte beispielsweise die folgende Funktion: $f(x) = x \times 2$. Welchen Wert besitzt dann $f(5)$? Wenn deine Antwort 10 lautet, bist du bereit.

Darüber hinaus ist dieses Kapitel (wie das ganze Buch) leichter verständlich, wenn dir eine Programmiersprache vertraut ist. Die Beispiele in diesem Buch sind in Python geschrieben. Falls du noch keine Programmiersprache kennst und eine erlernen möchtest, solltest du Python wählen – die Sprache ist hervorragend für Anfänger geeignet. Wenn du eine andere dir bekannte Programmiersprache (wie z. B. Ruby) verwenden möchtest, ist das problemlos möglich.

1.2 Binäre Suche

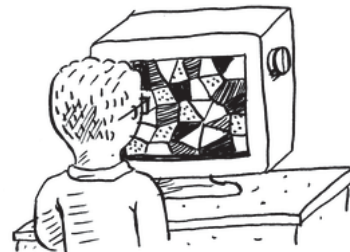
Nehmen wir an, du suchst in einem Telefonbuch nach einer Person (wie altmodisch das klingt!). Der Name beginnt mit *K*. Du könntest nun am Anfang des Telefonbuchs loslegen und so lange blättern, bis du zum Buchstaben *K* gelangst. Wahrscheinlich wirst du mit der Suche jedoch eher in der Mitte anfangen, weil du weißt, dass sich die Einträge mit *K* ungefähr in der Mitte des Telefonbuchs befinden.

Oder du stellst dir vor, dass du einen Begriff, der mit dem Buchstaben *O* beginnt, in einem Wörterbuch suchst. Auch in diesem Fall wirst du mit der Suche in der Nähe der Mitte beginnen.

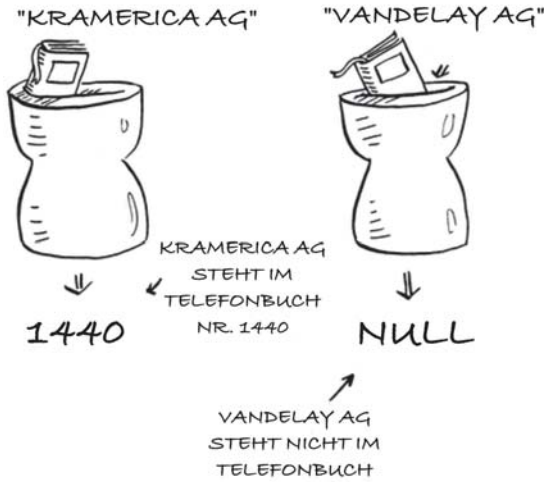
Und nun stelle dir vor, dass du dich bei Facebook anmeldest. Facebook muss dann überprüfen, ob du ein Konto besitzt und dementsprechend in einer Datenbank nach deinem Namen suchen. Nehmen wir an, dein Username lautet *karlmageddon*. Facebook könnte nun beim Buchstaben *A* mit der Suche anfangen – allerdings ist es sinnvoller, irgendwo in der Mitte anzufangen.

Bei dieser Aufgabe handelt es sich um eine Suche. Zur Lösung solcher Aufgaben kommt stets der gleiche Algorithmus zum Einsatz: die *binäre Suche*.

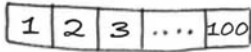
Die binäre Suche ist ein Algorithmus, dessen Eingabe aus einer sortierten Liste von Elementen besteht. (Ich erkläre später, warum die Liste sortiert sein muss.) Wenn das gesuchte Element in dieser Liste enthalten ist, liefert die binäre Suche die Position zurück, an der es sich befindet. Andernfalls gibt die binäre Suche den Wert `null` zurück.



Zum Beispiel:



Hier ist ein Beispiel für die Funktionsweise der binären Suche. Ich denke an eine Zahl zwischen 1 und 100.



Du musst nun meine Zahl mit möglichst wenigen Versuchen erraten. Ich sage dir jeweils, ob die geratene Zahl zu groß, zu klein oder richtig ist.

Nehmen wir an, du nennst die Zahlen der Reihe nach: 1, 2, 3, 4, ... Das sähe dann folgendermaßen aus:



Hierbei handelt es sich um eine *einfache Suche* (vielleicht wäre *eintönige Suche* in diesem Fall eine passendere Bezeichnung). Mit jedem Versuch schließt du nur eine einzige Zahl aus. Wenn ich an die Zahl 99 gedacht hätte, würdest du 99 Versuche benötigen, um meine Zahl zu erraten!

1.2.1 Eine bessere Suchmethode

Ein besseres Verfahren ist das folgende: Fang mit der Zahl 50 an.



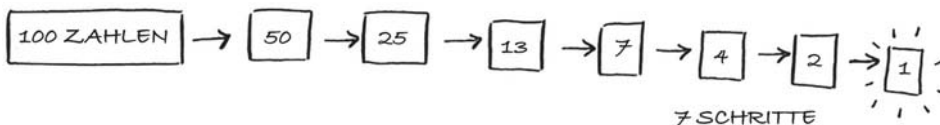
Diese ist zu klein, allerdings hast du soeben *die Hälfte* aller Zahlen ausgeschlossen! Du weißt nun, dass alle Zahlen von 1 bis 50 zu klein sind. Nächster Versuch: 75.



Zu groß, aber du hast wieder die Hälfte der verbliebenen Zahlen ausgeschlossen! *Bei einer binären Suche nennst du die Zahl in der Mitte und schließt dadurch jeweils die Hälfte der noch vorhandenen Zahlen aus.* Nun ist 63 an der Reihe (die Mitte zwischen 50 und 75).



So funktioniert die binäre Suche. Du hast soeben deinen ersten Algorithmus erlernt! So viele Zahlen kannst du mit jedem Rateversuch ausschließen:



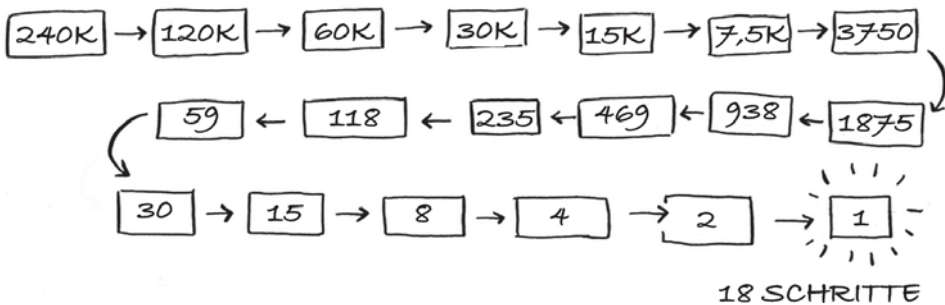
An welche Zahl ich denke, spielt keine Rolle: Du benötigst höchstens 7 Versuche, um sie zu erraten, weil bei jedem Rateversuch so viele Zahlen ausgeschlossen werden.

Nehmen wir wieder an, du suchst nach einem Begriff in einem Wörterbuch, das 240.000 Einträge enthält. Was meinst du, wie viele Schritte für eine Suche im *Worst Case*, also im ungünstigsten Fall, nötig sind?

EINFACHE SUCHE: _____ SCHRITTE

BINÄRE SUCHE: _____ SCHRITTE

Bei der einfachen Suche können 240.000 Schritte notwendig sein, sofern sich das gesuchte Wort ganz am Ende des Wörterbuchs befindet. Bei der binären Suche hingegen wird bei jedem Schritt die Anzahl der verbliebenen Wörter halbiert, bis schließlich nur noch ein Wort übrig ist.



Bei der binären Suche sind also 18 Schritte erforderlich – ein riesiger Unterschied! Verallgemeinert bedeutet das: Bei einer Liste der Länge n benötigt die binäre Suche im *Worst Case* $\log_2 n$ Schritte, bei einer einfachen Suche sind hingegen n Schritte erforderlich.

Logarithmen

Vielleicht erinnerst du dich nicht mehr daran, was Logarithmen sind, aber du weißt vermutlich noch, was Exponentialfunktionen sind. Der Ausdruck $\log_{10} 100$ entspricht der Frage: »Wie viele Zehnen muss man miteinander multiplizieren, um 100 zu erhalten?«. Die Antwort lautet 2: $10 \times 10 = 100$. Also ist $\log_{10} 100 = 2$. Logarithmen sind die Umkehrfunktionen von Exponentialfunktionen.

Wenn es in diesem Buch um die Laufzeit und die Landau-Notation (die in Kürze erklärt wird) geht, bedeutet \log stets \log_2 . Wenn du für die Suche nach einem Element eine einfache Suche verwendest, musst du im *Worst Case* jedes einzelne

Element überprüfen. Bei einer Liste von 8 Zahlen musst du höchstens 8 Zahlen überprüfen. Bei einer binären Suche musst du im Worst Case $\log n$ Elemente überprüfen. Für eine Liste mit 8 Elementen gilt $\log 8 = 3$, denn $2^3 = 8$. Du musst also höchstens 3 Zahlen überprüfen (und dann kannst du mit dem 4. Versuch das richtige Ergebnis nennen). Für eine Liste mit 1.024 Elementen gilt $\log 1.024 = 10$, denn $2^{10} = 1.024$. Bei einer Liste von 1.024 Zahlen musst du also höchstens 10 Zahlen überprüfen.

$$10^2 = 100 \iff \log_{10} 100 = 2$$

$$10^3 = 1000 \iff \log_{10} 1000 = 3$$

$$2^3 = 8 \iff \log_2 8 = 3$$

$$2^4 = 16 \iff \log_2 16 = 4$$

$$2^5 = 32 \iff \log_2 32 = 5$$

Hinweis

In diesem Buch geht es des Öfteren um die logarithmische Laufzeit, deshalb sollte dir das Konzept von Logarithmen vertraut sein. Sollte dies nicht der Fall sein, findest du bei der Khan Academy (<https://www.khanacademy.org>) ein anschauliches englisches Video, das dieses Konzept verdeutlicht.

Hinweis

Die binäre Suche funktioniert nur dann, wenn die zu durchsuchende Liste sortiert ist. Die Namen in einem Telefonbuch sind beispielsweise alphabetisch sortiert. Hier kannst du also für die Suche nach einem Namen eine binäre Suche verwenden. Wie sähe es aus, wenn die Namen nicht sortiert wären?

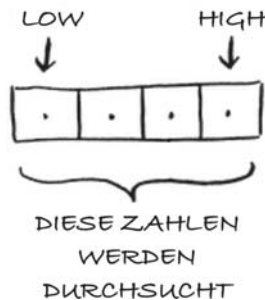
Sehen wir uns doch einmal an, wie man eine binäre Suche in Python programmiert. Der Beispielpcode verwendet Arrays. Falls du nicht weißt, wie Arrays funktionieren, keine Sorge, sie werden im nächsten Kapitel erklärt. Du brauchst nur zu wissen, dass sie eine Sequenz von Elementen in einer Reihe aufeinanderfolgender Behälter speichern können, deren Gesamtheit als Array bezeichnet wird. Die Behälter werden, angefangen bei 0, durchnummeriert: Der erste befindet sich an der Position #0, der zweite ist #1, der dritte #2 usw.

Hinweis

Ich verwende die Begriffe *list* und *array* im Code synonym. Das liegt daran, dass Arrays in Python als Listen bezeichnet werden.

Die Funktion `binary_search` nimmt ein sortiertes Array und ein Objekt entgegen. Wenn das Objekt in diesem Array enthalten ist, liefert die Funktion dessen Position zurück. Du führst darüber Buch, welcher Teil des Arrays zu durchsuchen ist. Anfangs handelt es sich um das gesamte Array:

```
low = 0  
high = len(list) - 1
```



Du überprüfst jeweils das mittlere Element:

```
mid = (low + high) // 2 ❶  
guess = list[mid]
```

❶ Der Wert der `mid`-Funktion wird von Python automatisch abgerundet, sofern $(low + high)$ keine gerade Zahl ist.

Sollte der geratene Wert zu klein sein, aktualisierst du `low` dementsprechend:

```
if guess < item:  
    low = mid + 1
```



Und sollte der geratene Wert zu groß sein, aktualisierst du `high`. Hier ist der vollständige Code:

```
def binary_search(list, item):  
    low = 0 ❶  
    high = len(list)-1 ❶  
  
    while low <= high: ❷  
        mid = (low + high) // 2 ❸  
        guess = list[mid]  
        if guess == item: ❹  
            return mid  
        if guess > item: ❺  
            high = mid - 1  
        else: ❻  
            low = mid + 1  
    return None ❼  
  
my_list = [1, 3, 5, 7, 9] ❸  
  
print binary_search(my_list, 3) # => 1 ❹  
print binary_search(my_list, -1) # => None ❺
```

- ❶ `low` und `high` führen darüber Buch, welcher Teil der Liste durchsucht wird.
- ❷ Solange der Suchbereich mehr als ein Element umfasst ...
- ❸ ... wird das mittlere Element überprüft.
- ❹ Das gesuchte Objekt wurde gefunden.
- ❺ Der geratene Wert war zu groß.
- ❻ Der geratene Wert war zu klein.
- ❼ Das gesuchte Objekt ist in der Liste nicht enthalten.
- ❸ Testen der Funktion.
- ❹ Denk daran, dass die Nummerierung der Listenelemente bei 0 beginnt. Das zweite Element hat den Index 1
- ❺ »None« bedeutet in Python nil. Es zeigt an, dass ein Objekt nicht gefunden wurde.



Übungen

- 1.1 Eine sortierte Liste enthält 128 Namen. Du durchsuchst sie mit einer binären Suche. Wie viele Schritte sind dafür maximal erforderlich?
- 1.2 Nehmen wir an, du verdoppelst die Größe der Liste. Wie viele Schritte sind nun maximal erforderlich?

1.2.2 Laufzeit

Bei allen vorgestellten Algorithmen werde ich die Laufzeit erklären. Ob nun die Laufzeit oder aber der Speicherplatzbedarf optimiert werden soll: Im Allgemeinen möchte man den effizientesten Algorithmus auswählen.

Zurück zur binären Suche. Wie viel Zeit sparst du durch ihre Verwendung? Beim ersten Ansatz wurden alle Zahlen der Reihe nach überprüft. Wenn die Liste 100 Zahlen enthält, sind bis zu 100 Rateversuche erforderlich. Wenn die Liste 4 Milliarden Zahlen enthält, sind bis zu 4 Milliarden Rateversuche notwendig. Die maximale Anzahl der Rateversuche entspricht also der Größe der Liste. Man bezeichnet das als lineare Laufzeit.



Bei der binären Suche verhält es sich anders. Wenn die Liste 100 Objekte enthält, sind höchstens 7 Rateversuche erforderlich. Enthält die Liste 4 Milliarden Objekte, sind höchstens 32 Rateversuche notwendig. Ziemlich leistungsstark, oder? Die binäre Suche benötigt eine *logarithmische Laufzeit*. Die folgende Tabelle fasst unsere Ergebnisse zusammen.

EINFACHE SUCHE	BINÄRE SUCHE
100 OBJEKTE	100 OBJEKTE
↓	↓
100 VERSUCHE	7 VERSUCHE
-----	-----
4.000.000.000 OBJEKTE	4.000.000.000 OBJEKTE
↓	↓
4.000.000.000 VERSUCHE	32 VERSUCHE
-----	-----
$O(n)$	$O(\log n)$
↑	↑
LINEARE LAUFZEIT	LOGARITHMISCHE LAUFZEIT

Handwritten notes: Two callouts labeled "VERSUCHE GESPART!" with arrows pointing to the reduction in attempts for the binary search (7 vs 100 and 32 vs 4 billion).

1.3 Landau-Notation

Die *Landau-Notation* (engl. *Big-O-Notation*, der Begriff Landau-Symbole ist ebenfalls gebräuchlich) ist eine spezielle Schreibweise, die angibt, wie schnell ein Algorithmus ist. Warum ist das wichtig? Du wirst feststellen, dass du häufig Algorithmen verwendest, die andere Leute entwickelt haben. In diesem Fall ist es praktisch zu wissen, wie schnell oder langsam diese Algorithmen sind. In diesem Abschnitt werde ich dir zeigen, was die Landau-Notation bedeutet und eine Liste der gängigsten Laufzeiten der Algorithmen präsentieren, die Landau-Symbole verwenden.

1.3.1 Die Laufzeiten von Algorithmen nehmen unterschiedlich schnell zu

Bob programmiert einen Suchalgorithmus für die NASA. Sein Algorithmus kommt zum Einsatz, wenn eine Rakete auf dem Mond landen soll und kann den Landeplatz berechnen.

Hierbei handelt es sich um ein Beispiel dafür, dass die Laufzeiten zweier Algorithmen auf unterschiedliche Weise zunehmen können. Bob muss sich zwischen einer einfachen Suche und einer binären Suche entscheiden. Der Algorithmus muss sowohl schnell sein als auch fehlerlos funktionieren. Einerseits ist die binäre Suche schneller. Bob hat nämlich nur *10 Sekunden* Zeit, um den Landeplatz zu ermitteln – denn sonst kommt die Rakete vom Kurs ab. Andererseits lässt sich die einfache Suche leichter programmieren. Und Bob möchte *wirklich* nicht, dass der Code zum Landen einer Rakete Bugs enthält! Vorsichtshalber will Bob die Laufzeiten beider Algorithmen messen, wenn er eine Liste verwendet, die 100 Elemente enthält.

Nehmen wir an, das Überprüfen eines Elements dauert eine Millisekunde (ms). Bei der einfachen Suche muss Bob 100 Elemente überprüfen, also dauert das Ausführen der Suche 100 ms. Bei der binären Suche muss er hingegen lediglich 7 Elemente überprüfen. $\log_2 100$ ist ungefähr 7, also dauert die Suche 7 ms. Realistisch betrachtet wird die Liste tatsächlich eher eine Milliarde Elemente enthalten. Wie lange würde dann die einfache Suche dauern? Und wie lange die binäre Suche? Vergewissere dich, dass du diese beiden Fragen beantworten kannst, bevor du weiterliest.





Bob führt eine binäre Suche mit einer Liste aus, die eine Milliarde Elemente enthält. Sie dauert 30 ms ($\log_2 1.000.000.000$ ist ungefähr 30). »30 ms!«, geht ihm durch Kopf. »Die binäre Suche ist rund 15 Mal schneller als die einfache Suche, denn bei 100 Elementen dauerte die einfache Suche 100 ms und die binäre Suche 7 ms. Also sollte die einfache Suche $30 \times 15 = 450$ ms dauern, richtig? Das ist deutlich unter den zulässigen 10 Sekunden.« Bob entschließt sich also dazu, die einfache Suche zu verwenden. Aber ist das die richtige Entscheidung?

Nein, denn wie sich zeigen wird, liegt Bob falsch. Völlig falsch. Die Laufzeit der einfachen Suche beträgt bei einer Milliarden Objekte eine Milliarde ms – das sind mehr als 11 Tage! Das Problem besteht hier darin, dass die Laufzeiten für binäre und einfache Suche *nicht auf dieselbe Weise zunehmen*.

	EINFACHE SUCHE	BINÄRE SUCHE
100 ELEMENTE	100ms	7ms
10.000 ELEMENTE	10 Sekunden	14ms
1.000.000.000 ELEMENTE	11 TAGE	32ms

DIE LAUFZEITEN NEHMEN MIT SEHR
UNTERSCHIEDLICHER GESCHWINDIGKEIT ZU!

Wenn die Anzahl der Objekte zunimmt, benötigt die binäre Suche zur Ausführung etwas mehr Zeit. Die einfache Suche jedoch benötigt *sehr viel* mehr Zeit zur

Ausführung. Deshalb wird die binäre Suche sehr viel schneller als die einfache Suche, wenn die Zahlenliste größer wird. Bob dachte, dass die binäre Suche 15 Mal schneller als die einfache Suche ist, aber das stimmt nicht. Wenn die Liste eine Milliarden Objekte enthält, ist die binäre Suche rund 33 Millionen Mal schneller. Aus diesem Grund reicht es nicht aus zu wissen, wie lange ein Algorithmus zur Ausführung benötigt – man muss auch wissen, wie die Laufzeit anwächst, wenn die Größe der Liste zunimmt. Hier kommt die Landau-Notation ins Spiel.

Die Landau-Symbole geben an, wie schnell ein Algorithmus ist. Betrachte beispielsweise eine Liste der Größe n . Bei der einfachen Suche muss jedes einzelne Element überprüft werden, daher sind n Operationen erforderlich. Für diese Laufzeit wird das Landau-Symbol $O(n)$ verwendet. Aber wo sind die Sekunden angegeben? Es gibt keine – die Geschwindigkeit wird nicht in Sekunden angegeben. *Die Landau-Notation ermöglicht es, die Anzahl der Operationen zu vergleichen.* Sie teilt dir mit, wie schnell die Anzahl der Operationen des Algorithmus anwächst.



Hier ist ein weiteres Beispiel: Die binäre Suche benötigt $\log n$ Operationen, um eine Liste der Größe n zu überprüfen. Wie sieht das als Landau-Notation aus? Hierfür schreibt man $O(\log n)$. Im Allgemeinen notiert man ein Landau-Symbol folgendermaßen:

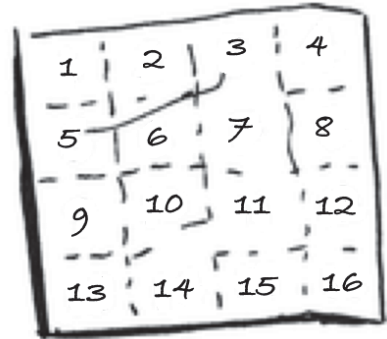
$O(n)$
GROSSES O ANZAHL DER OPERATIONEN

Du kannst daran ablesen, wie viele Operationen der Algorithmus ausführen wird. Im Englischen spricht man auch von der *Big-O-Notation*, weil der Anzahl der Operationen ein großes »O« vorangestellt wird. (Klingt wie ein Scherz, ist aber tatsächlich wahr!)

Betrachten wir einige Beispiele. Kannst du die Laufzeiten der Algorithmen herausfinden?

1.3.2 Visualisierung verschiedener Laufzeiten

Zunächst ein praktisches Beispiel, das du mit ein paar Blättern Papier und einem Bleistift leicht nachvollziehen kannst. Deine Aufgabe besteht darin, ein Gitter zu zeichnen, das aus 16 Kästchen besteht.



Algorithmus 1

Eine Möglichkeit besteht darin, der Reihe nach 16 einzelne Kästchen zu zeichnen. Wie du weißt, gibt die Landau-Notation die Anzahl der Operationen an. In diesem Beispiel stellt das Zeichnen eines Kästchens eine Operation dar. Und du musst 16 Kästchen zeichnen. Wie viele Operationen sind also erforderlich, wenn du die Kästchen einzeln zeichnest?

WELCHER ALGORITHMUS
WÄRE ZUM ZEICHNEN
DES GITTERS GEEIGNET?



Zum Zeichnen von 16 Kästchen sind 16 Schritte notwendig. Welche Laufzeit benötigt dieser Algorithmus demzufolge?

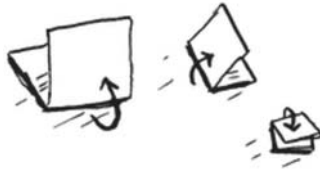
Algorithmus 2

Probiere nun den folgenden Algorithmus aus. Falte das Papierblatt.

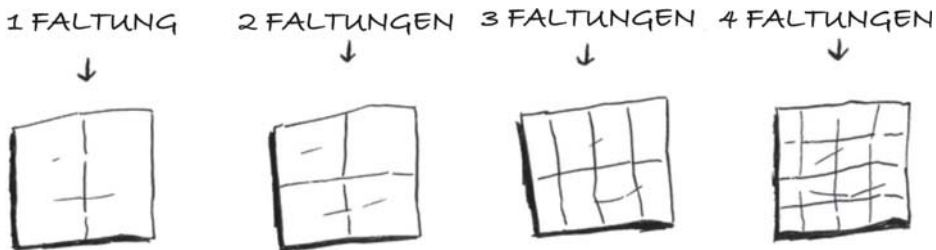


Bei diesem Beispiel ist jedes Falten des Papierblatts eine Operation. Du hast soeben also zwei Kästchen mit nur einer Operation erstellt!

Falte das Papierblatt ein zweites, drittes und viertes Mal.



Falte das Papierblatt nach der vierten Faltung wieder auf. Und siehe da – ein tadelloses Gitter. Mit jeder Faltung verdoppelst du die Anzahl der Kästchen. Du hast also mit 4 Operationen 16 Kästchen erstellt!



Du kannst die Anzahl der Kästchen mit jeder Faltung verdoppeln, also kannst du mit 4 Schritten 16 Kästchen »zeichnen«. Welche Laufzeit benötigt dieser Algorithmus? Lege die Laufzeiten für die beiden Algorithmen fest, bevor du weitermachst.

Antworten: Algorithmus 1 benötigt die Laufzeit $O(n)$ und Algorithmus 2 die Laufzeit $O(\log n)$.

1.3.3 Die Landau-Notation beschreibt die Laufzeit im Worst Case

Nehmen wir an, du verwendest eine einfache Suche, um in einem Telefonbuch nach einer Person zu suchen. Wie du weißt, benötigt die einfache Suche die Laufzeit $O(n)$, was im Worst Case bedeutet, dass du sämtliche Einträge deines Telefonbuchs überprüfen musst. Im vorliegenden Fall suchst du nach meinem Namen »Adit«. Hierbei handelt es sich um den ersten Eintrag in deinem Telefonbuch. Du musst also gar nicht alle Einträge überprüfen – du hast das Gesuchte schon beim ersten Versuch gefunden. Heißt das nun, dass dieser Algorithmus die Laufzeit $O(n)$ benötigt? Oder sogar nur $O(1)$, weil du die Person beim ersten Versuch gefunden hast?

Die einfache Suche benötigt nach wie vor die Laufzeit $O(n)$. In diesem Fall hast du zwar sofort gefunden, was du gesucht hast, aber hierbei handelt es sich um den günstigsten Fall. Die Landau-Notation beschreibt jedoch den *Worst Case*. Du kannst also sagen, dass du im *Worst Case* sämtliche Einträge im Telefonbuch einmal überprüfen musst – und das entspricht der benötigten Laufzeit $O(n)$. Dabei handelt es sich also um eine Art Zusicherung (eine obere Schranke für die Laufzeit) – du kannst dir sicher sein, dass die einfache Suche niemals langsamer als $O(n)$ sein wird.

Hinweis

Neben der Laufzeit im Worst Case, also im ungünstigsten Fall, spielt auch die Laufzeit im Average Case, also im durchschnittlichen Fall, eine wichtige Rolle. Diese sogenannten *Zeitkomplexitäten* (der Worst Case und der Average Case) werden einander in Kapitel 4 gegenübergestellt.

1.3.4 Typische Laufzeiten gebräuchlicher Algorithmen

Nachstehend sind fünf Laufzeiten (sortiert nach abnehmender Geschwindigkeit) aufgeführt, die dir häufig begegnen werden:

- $O(\log n)$, die auch als *logarithmische Laufzeit* bezeichnet wird. Beispiel: Die binäre Suche.
- $O(n)$, die auch als *lineare Laufzeit* bezeichnet wird. Beispiel: Die einfache Suche.
- $O(n * \log n)$. Beispiel: Ein schneller Sortieralgorithmus wie Quicksort (das in Kapitel 4 erörtert wird).
- $O(n^2)$. Beispiel: Ein langsamer Sortieralgorithmus wie Selectionsort (das in Kapitel 2 erörtert wird).
- $O(n!)$. Beispiel: Ein richtig langsamer Algorithmus, wie der Algorithmus zur Lösung des Problems des Handlungsreisenden (mehr dazu in Kürze).

Nehmen wir an, du zeichnest wieder die 16 Kästchen und du kannst dir dafür einen von 5 verschiedenen Algorithmen aussuchen. Wenn du den ersten Algorithmus verwendest, wirst du eine Laufzeit von $O(\log n)$ benötigen, um die Kästchen zu zeichnen. Du kannst pro Sekunde 10 Operationen ausführen. Bei einer Laufzeit von $O(\log n)$ benötigst du 4 Operationen, um 16 Kästchen zu erzeugen ($\log 16 = 4$). Du benötigst also 0,4 Sekunden zum Zeichnen des Gitters. Und wenn du 1.024 Kästchen zeichnen müsstest? Zum Zeichnen von 1.024 Kästchen sind $\log 1.024 = 10$ Operationen bzw. 1 Sekunde Zeit erforderlich. Diese Angaben gelten für den ersten Algorithmus.