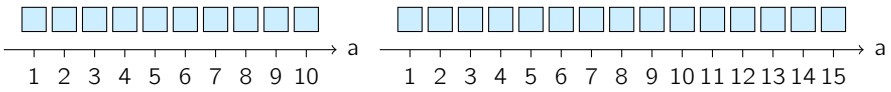
Merkblatt Laufzeit

Im Folgenden sei f(n) die Anzahl der Steine für einen gegebenen Wert n, da `legeStein()` der laufzeitrelevante Aufruf ist. Bei einem Such-Algorithmus entspräche `legeStein()` z. B. einem Vergleich mit einem Element eines Feldes der Länge n.

Lineares Wachstum

```
1 wiederhole a von 1 bis n:
2     legeStein()
```

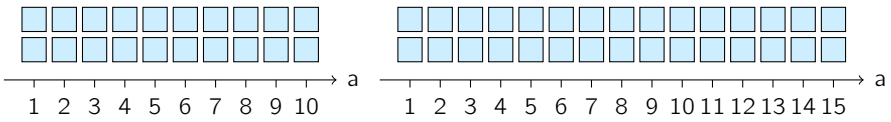


Aufruf mit n = 10 Aufruf mit n = 15

Es gilt f(n) = n und damit f ∈ $\mathcal{O}$(n). Für jeden Wert x, der zu n dazukommt, müssen n + x Steine gelegt werden.

```
1 wiederhole a von 1 bis n:
2     legeStein()
3     legeStein()
```

```
wiederhole a von 1 bis n:
    wiederhole b von 1 bis 2:
        legeStein()
```

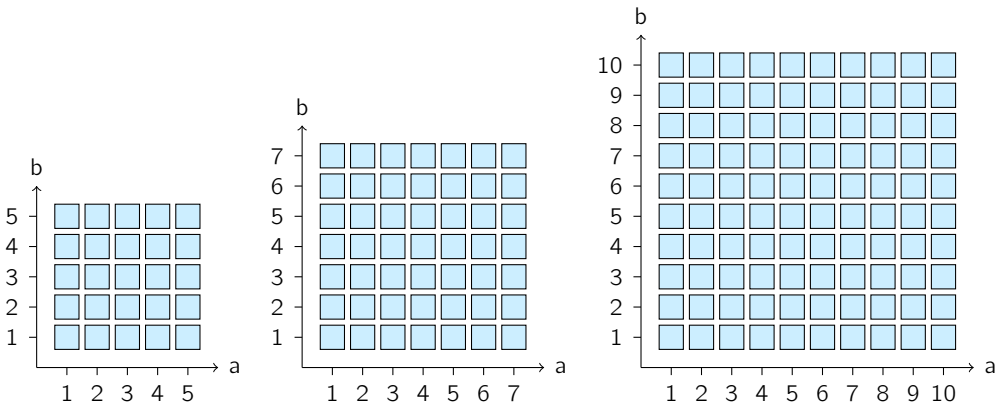


Aufruf mit n = 10 Aufruf mit n = 15

Es gilt f(n) = 2n und damit immer noch f ∈ $\mathcal{O}$(n). Für jeden Wert x, der zu n dazukommt, müssen 2 · (n + x) Steine gelegt werden.

Quadratisches Wachstum

```
1 wiederhole a von 1 bis n:
2     wiederhole b von 1 bis n:
3         legeStein()
```

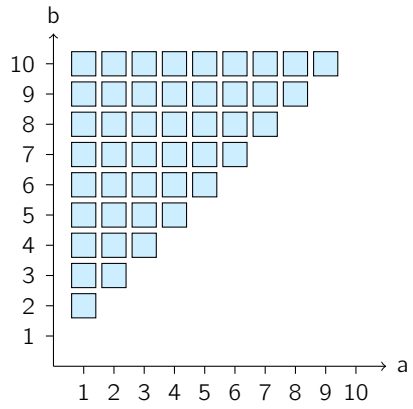


Aufruf mit n = 5 Aufruf mit n = 7 Aufruf mit n = 10

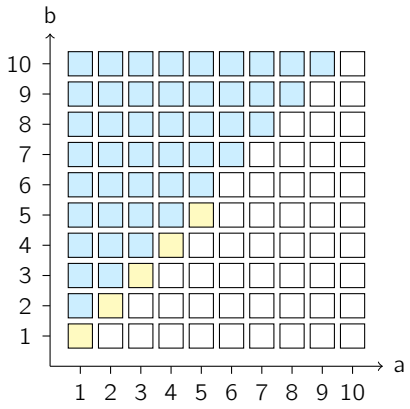
Es gilt f(n) = n^2 und f ∈ $\mathcal{O}(n^2)$.

Bei BubbleSort muss man immer nur die restlichen Felder ab der Stelle a überprüft werden. Das sieht dann so aus:

```
1 wiederhole a von 1 bis (n - 1):
2     wiederhole b von (a + 1) bis n:
3         legeStein()
```



Aufruf mit n = 10



Aufruf mit n = 10 mit Platzhaltern

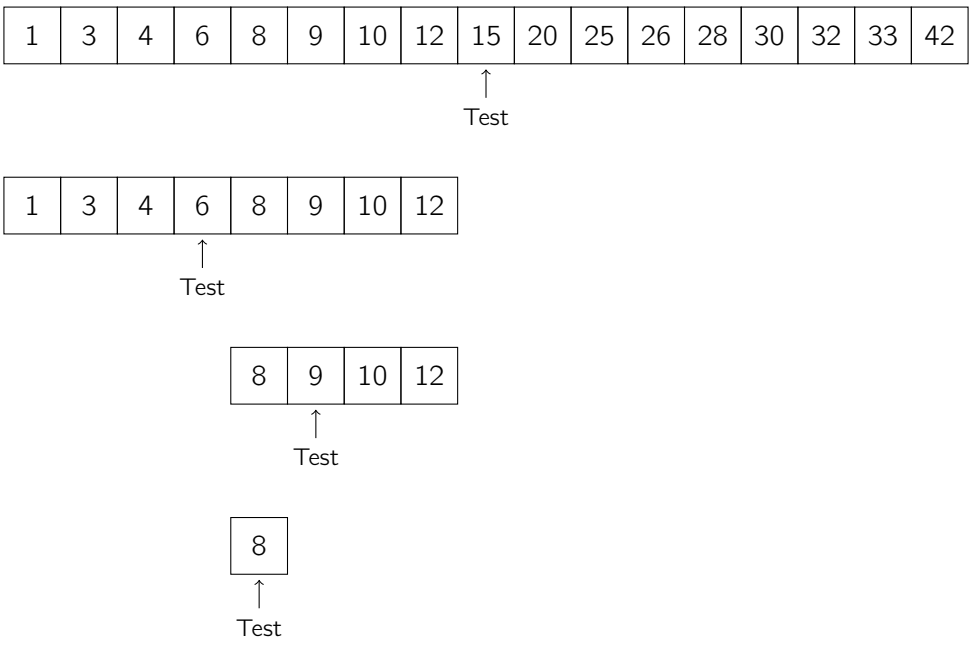
Es gibt bei n = 10 insgesamt

$$9+8+7+6+5+4+3+2+1=\frac{9\cdot 10}{2}=45$$

blaue Steine (Gaußsche Summenformel). Optisch betrachtet belegen die blauen Steine immer nur einen dreieckigen Teil eines Quadrats von $n^2 = 100$ Steinen. Es gibt $\frac{1}{2}n^2$ weiße Steine und $\frac{1}{2}n$ gelbe Steine. Es gilt daher $f(n) = n^2 - \frac{1}{2}n^2 - \frac{1}{2}n = \frac{1}{2}n^2 - \frac{1}{2}n$ und es bleibt bei $f \in \mathcal{O}(n^2)$.

Logarithmisches Wachstum

Angenommen wir suchen in einem sortierten Feld einen bestimmten Wert. Unsere Suchstrategie ist, immer zuerst auf das mittlere Feldelement bei einer ungeraden Feldlänge und auf das rechteste Feldelement der linken Hälfte bei einer geraden Feldlänge zu tippen. Im Folgenden suchen wir in einem sortierten Feld der länge 17 den Wert 8:



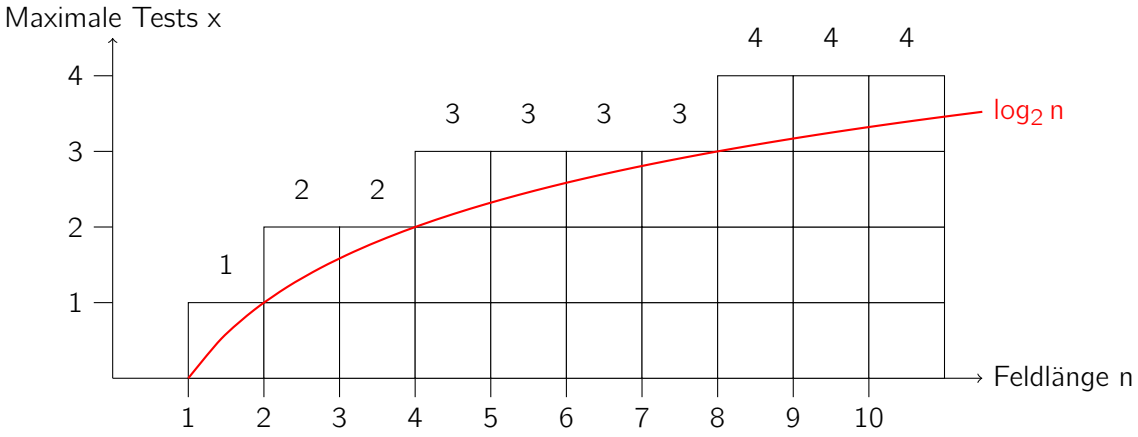
Hätten wir den Wert 6 gesucht, wären wir früher fertig gewesen. Wir interessieren uns aber für den schlechtesten Fall bei der Suche. Das ist, wenn wir das Feld solange teilen, bis nur noch ein Feldelement als Möglichkeit übrig bleibt. Die mathematische Frage ist also, wie oft man durch zwei teilen kann, bis kein Feld mehr übrig ist, das wir testen müssen:

$$n \cdot \underbrace{\frac{1}{2} \cdot \frac{1}{2} \cdot \dots \cdot \frac{1}{2}}_{x\text{–mal}} < 1$$

Bei einem Feld der Länge 1 müssen wir einen Test machen ($1 \cdot \frac{1}{2} = 0{,}5 < 1$). Bei einem Feld der Länge 2 müssen wir im schlechtesten Fall zweimal testen ($2 \cdot \frac{1}{2} \cdot \frac{1}{2} = 0{,}5 < 1$). Aber auch bei einem Feld der Länge 3 reichen maximal zwei Test aus ($3 \cdot \frac{1}{2} \cdot \frac{1}{2} = 0{,}75 < 1$). Wie lässt sich dieses x direkt berechnen. Dazu formen wir folgendermaßen um:

$$\begin{aligned} n \cdot \underbrace{\frac{1}{2} \cdot \frac{1}{2} \cdot \dots \cdot \frac{1}{2}}_{x\text{–mal}} &< 1 \\ n \cdot \frac{1}{2^x} &< 1 \\ n &< 2^x \\ \log_2 n &< x \end{aligned}$$

Natürlich sind wir an dem kleinsten $x \in \mathbb{N}$ interessiert. Betrachten wir das graphisch:



Bis zu welcher Feldlänge kommen wir mit maximal 4 Test aus? Bis zu einer Feldlänge von 15! Bei 16 benötigen wir maximal 5 Tests. Haben wir ein Feld der Länge 16, so erhalten wir mit einem Test im schlechtesten Fall ein Feld mit der Länge 8, das wir weiter überprüfen müssen. Der nächste Anstieg ist dann erst bei einer Feldlänge von 32, dann wieder bei 64 usw. Die Anzahl der Tests lässt sich so berechnen:

```
1 f(n) = floor(log2(n)) + 1
```

Die `floor` Funktion schneidet die Nachkommastellen ab und mit der Addition von 1 haben wir das kleinste x für das $\log_2 n < x$ gilt. Für die Laufzeit gilt $f \in \mathcal{O}(\log n)$.