Big O Notation

Maximilian Bosch

Weekly Dev Basics

Benchmarking

- Hard to get right
- Wie identifiziere ich Bottlenecks?
  - Probleme in Infrastruktur & System
  - → **ineffiziente Implementierung**
- **Fragestellung:** wie **teuer** ist eine Funktion?
  - Big-O-Notation bietet Formalismus für diese Frage.

Benchmarking

- Hard to get right
- Wie identifiziere ich Bottlenecks?
  - Probleme in Infrastruktur & System
  - $\rightarrow$ **ineffiziente Implementierung**
- **Fragestellung:** wie **teuer** ist eine Funktion?
  - Big-O-Notation bietet Formalismus für diese Frage.

Benchmarking

- Hard to get right
- Wie identifiziere ich Bottlenecks?
  - Probleme in Infrastruktur & System

  $\rightarrow$ **ineffiziente Implementierung**
- **Fragestellung:** wie **teuer** ist eine Funktion?
  - Big-O-Notation bietet Formalismus für diese Frage.

Benchmarking

- Hard to get right
- Wie identifiziere ich Bottlenecks?
  - Probleme in Infrastruktur & System

  $\rightarrow$ **ineffiziente Implementierung**
- **Fragestellung:** wie **teuer** ist eine Funktion?
  - Big-O-Notation bietet Formalismus für diese Frage.

Benchmarking

- Hard to get right
- Wie identifiziere ich Bottlenecks?
  - Probleme in Infrastruktur & System

  → **ineffiziente Implementierung**
- **Fragestellung:** wie **teuer** ist eine Funktion?
  - Big-O-Notation bietet Formalismus für diese Frage.

Benchmarking

- Hard to get right
- Wie identifiziere ich Bottlenecks?
  - Probleme in Infrastruktur & System

  $\rightarrow$ **ineffiziente Implementierung**
- **Fragestellung:** wie **teuer** ist eine Funktion?
  - Big-O-Notation bietet Formalismus für diese Frage.

Simples Beispiel

```
1  function snens(list) {
2    const n = list.length
3    for (let i = 0; i < n; i++) {
4      list[i] *= 2
5    }
6    return list
7  }
```

```
> snens([1,2,3,4])
[ 2, 4, 6, 8 ]
```

- Wie messen wir Komplexität?
- Idee: Anzahl der Operationen zählen

Simples Beispiel

```
1  function snens(list) {
2    const n = list.length
3    for (let i = 0; i < n; i++) {
4      list[i] *= 2
5    }
6    return list
7  }
```

```
> snens([1,2,3,4])
[ 2, 4, 6, 8 ]
```

- Wie messen wir Komplexität?
- Idee: Anzahl der Operationen zählen

Simples Beispiel

```
1  function snens(list) {
2    const n = list.length
3    for (let i = 0; i < n; i++) {
4      list[i] *= 2
5    }
6    return list
7  }
```

```
> snens([1,2,3,4])
[ 2, 4, 6, 8 ]
```

- Wie messen wir Komplexität?
- Idee: Anzahl der Operationen zählen

Ergebnis

list.length	Anzahl Operationen
0	4
1	7
2	10
3	13

In Abhängigkeit der Größe der Liste (n)

$n \rightarrow 4 + 3n$ (**lineares** Wachstum)

$snens \in O(4 + 3n) = O(n)$

Ergebnis

list.length	Anzahl Operationen
0	4
1	7
2	10
3	13

In Abhängigkeit der Größe der Liste (n)

$n \rightarrow 4 + 3n$ (**lineares** Wachstum)

$snens \in O(4 + 3n) = O(n)$

Ergebnis

list.length	Anzahl Operationen
0	4
1	7
2	10
3	13

In Abhängigkeit der Größe der Liste (n)

$n \to 4 + 3n$ (**lineares** Wachstum)

$$snens \in O(4 + 3n) = O(n)$$

Ergebnis

list.length	Anzahl Operationen
0	4
1	7
2	10
3	13

In Abhängigkeit **der Größe der Liste** (n)

$n \rightarrow 4 + 3n$ (**lineares** Wachstum)

$$snens \in O(4 + 3n) = O(n)$$

Rekapitulation

In Abhängigkeit der Größe der Liste (n)

$$n \rightarrow 4 + 3n \text{ (\textbf{lineares} Wachstum)}$$

$$snens \in O(4 + 3n) = O(n)$$

- Aussage: Anzahl Operationen steigt linear mit Größe der Eingabe.
- Einsatz in der Informatik: Anstieg der Laufzeit bei Vergrößerung der Eingabe.
- Formal als Menge definiert.

Rekapitulation

In Abhängigkeit der Größe der Liste (n)

$$n \rightarrow 4 + 3n \text{ (\textbf{lineares} Wachstum)}$$

$$snens \in O(4 + 3n) = O(n)$$

- Aussage: Anzahl Operationen steigt linear mit Größe der Eingabe.
- Einsatz in der Informatik: Anstieg der Laufzeit bei Vergrößerung der Eingabe.
- Formal als Menge definiert.

Rekapitulation

In Abhängigkeit der Größe der Liste (n)

$$n \rightarrow 4 + 3n \text{ (\textbf{lineares} Wachstum)}$$

$$snens \in O(4 + 3n) = O(n)$$

- Aussage: Anzahl Operationen steigt linear mit Größe der Eingabe.
- Einsatz in der Informatik: Anstieg der Laufzeit bei Vergrößerung der Eingabe.
- Formal als Menge definiert.

Rekapitulation

In Abhängigkeit **der Größe der Liste** (n)

$n \rightarrow 4 + 3n$ (**lineares** Wachstum)

$$snens \in O(4 + 3n) = O(n)$$

- Aussage: Anzahl Operationen steigt linear mit Größe der Eingabe.
- Einsatz in der Informatik: Anstieg der Laufzeit bei Vergrößerung der Eingabe.
- Formal als Menge definiert.

Theorie

$$\text{z.Z.: } g(n) = 4 + 3n, g \in O(n)$$

- Wähle ein $n_0 \in \mathbb{N}$
- Für alle $n \geq n_0$ muss folgendes gelten:
- $g(n) \leq c \cdot f(n), c \in \mathbb{R}_+$

- Wähle $n_0 = 1$
- Wähle $c = 7$
- $4 + 3n \leq 7n$

Beobachtungen

→ g wächst **höchstens linear.**

→ $g \in O(n^2)$

→ $O(n) \subset O(n^2)$

Theorie

$$\text{z.Z.: } g(n) = 4 + 3n, g \in O(n)$$

- Wähle ein $n_0 \in \mathbb{N}$
- Für alle $n \geq n_0$ muss folgendes gelten:
- $g(n) \leq c \cdot f(n), c \in \mathbb{R}_+$

- Wähle $n_0 = 1$
- Wähle $c = 7$
- $4 + 3n \leq 7n$

Beobachtungen

$\rightarrow$ g wächst **höchstens linear.**

$\rightarrow$ $g \in O(n^2)$

$\rightarrow$ $O(n) \subset O(n^2)$

Theorie

$$\text{z.Z.: } g(n) = 4 + 3n, g \in O(n)$$

- Wähle ein $n_0 \in \mathbb{N}$
- Für alle $n \geq n_0$ muss folgendes gelten:
- $g(n) \leq c \cdot f(n), c \in \mathbb{R}_+$

- Wähle $n_0 = 1$
- Wähle $c = 7$
- $4 + 3n \leq 7n$

Beobachtungen

→ g wächst **höchstens linear.**

→ $g \in O(n^2)$

→ $O(n) \subset O(n^2)$

Theorie

$$\text{z.Z.: } g(n) = 4 + 3n, g \in O(n)$$

- Wähle ein $n_0 \in \mathbb{N}$
- Für alle $n \geq n_0$ muss folgendes gelten:
- $g(n) \leq c \cdot f(n), c \in \mathbb{R}_+$

- Wähle $n_0 = 1$
- Wähle $c = 7$
- $4 + 3n \leq 7n$

Beobachtungen

$\rightarrow$ g wächst **höchstens linear.**

$\rightarrow$ $g \in O(n^2)$

$\rightarrow$ $O(n) \subset O(n^2)$

Theorie

$$\text{z.Z.: } g(n) = 4 + 3n, g \in O(n)$$

- Wähle ein $n_0 \in \mathbb{N}$
- Für alle $n \geq n_0$ muss folgendes gelten:
- $g(n) \leq c \cdot f(n), c \in \mathbb{R}_+$

- Wähle $n_0 = 1$
- Wähle $c = 7$
- $4 + 3n \leq 7n$

Beobachtungen

$\rightarrow$ g wächst **höchstens linear.**

$\rightarrow$ $g \in O(n^2)$

$\rightarrow$ $O(n) \subset O(n^2)$

Theorie

$$\text{z.Z.: } g(n) = 4 + 3n, g \in O(n)$$

- Wähle ein $n_0 \in \mathbb{N}$
- Für alle $n \geq n_0$ muss folgendes gelten:
- $g(n) \leq c \cdot f(n), c \in \mathbb{R}_+$

- Wähle $n_0 = 1$
- Wähle $c = 7$
- $4 + 3n \leq 7n$

Beobachtungen

$\rightarrow$ g wächst **höchstens linear.**

$\rightarrow$ $g \in O(n^2)$

$\rightarrow$ $O(n) \subset O(n^2)$

Theorie

$$\text{z.Z.: } g(n) = 4 + 3n, g \in O(n)$$

- Wähle ein $n_0 \in \mathbb{N}$
- Für alle $n \geq n_0$ muss folgendes gelten:
- $g(n) \leq c \cdot f(n), c \in \mathbb{R}_+$

- Wähle $n_0 = 1$
- Wähle $c = 7$
- $4 + 3n \leq 7n$

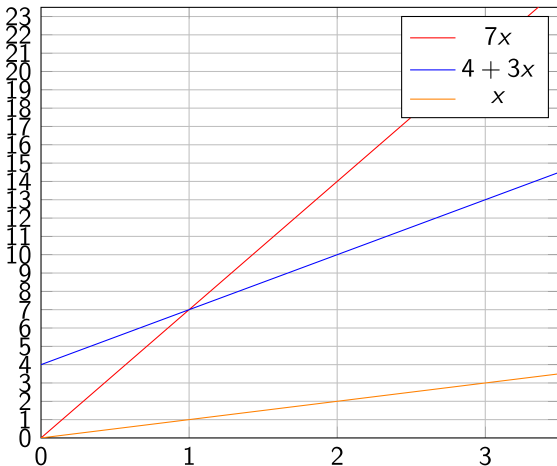
Beobachtungen

$\rightarrow$ g wächst **höchstens linear.**

$\rightarrow$ $g \in O(n^2)$

$\rightarrow$ $O(n) \subset O(n^2)$

Veranschaulichung



Übersicht

- $g \in O(n) \iff g$ wächst **höchstens** linear
- $g \in \Theta(n) \iff g$ wächst **genau** linear
- $g \in \Omega(n) \iff g$ wächst **mindestens** linear

Übersicht

- $g \in O(n) \iff g$ wächst **höchstens** linear
- $g \in \Theta(n) \iff g$ wächst **genau** linear
- $g \in \Omega(n) \iff g$ wächst **mindestens** linear

Übersicht

- $g \in O(n) \iff g$ wächst **höchstens** linear
- $g \in \Theta(n) \iff g$ wächst **genau** linear
- $g \in \Omega(n) \iff g$ wächst **mindestens** linear

QuickSort

Stark vereinfachte Implementierung:

```
1 function qsort(input) {
2   if (input.length == 0) return []
3   let p = input.shift()
4   return qsort(input.filter(num => num <= p))
5     .concat([p])
6     .concat(qsort(input.filter(num => num > p)))
7 }
```

- **Worstcase:** $O(n^2)$ (mit `qsort([1,2,3,4,5])`)
  - $n + (n-1) + (n-2) + (n-3) + (n-4) + (n-5) = \frac{n \cdot (n-1)}{2} \in O(n^2)$
- **Average case:** $O(n \cdot log(n))$
  - `input.filter()` ist $O(n)$
  - $log(n)$ rekursive Aufrufe (Annahme: Liste **halbiert** sich pro Aufruf von `qsort()`).

QuickSort

Stark vereinfachte Implementierung:

```
1  function qsort(input) {
2    if (input.length == 0) return []
3    let p = input.shift()
4    return qsort(input.filter(num => num <= p))
5      .concat([p])
6      .concat(qsort(input.filter(num => num > p)))
7  }
```

- **Worstcase:** $O(n^2)$ (mit `qsort([1,2,3,4,5])`)
  - $n + (n-1) + (n-2) + (n-3) + (n-4) + (n-5) = \frac{n \cdot (n-1)}{2} \in O(n^2)$
- **Average case:** $O(n \cdot log(n))$
  - `input.filter()` ist $O(n)$
  - $log(n)$ rekursive Aufrufe (Annahme: Liste **halbiert** sich pro Aufruf von `qsort()`).

QuickSort

Stark vereinfachte Implementierung:

```
1  function qsort(input) {
2    if (input.length == 0) return []
3    let p = input.shift()
4    return qsort(input.filter(num => num <= p))
5      .concat([p])
6      .concat(qsort(input.filter(num => num > p)))
7  }
```

- **Worstcase:** $O(n^2)$ (mit `qsort([1,2,3,4,5])`)
  - $n + (n-1) + (n-2) + (n-3) + (n-4) + (n-5) = \frac{n \cdot (n-1)}{2} \in O(n^2)$
- **Average case:** $O(n \cdot log(n))$
  - `input.filter()` ist $O(n)$
  - $log(n)$ rekursive Aufrufe (Annahme: Liste **halbiert** sich pro Aufruf von `qsort()`).

QuickSort

Stark vereinfachte Implementierung:

```
1  function qsort(input) {
2    if (input.length == 0) return []
3    let p = input.shift()
4    return qsort(input.filter(num => num <= p))
5      .concat([p])
6      .concat(qsort(input.filter(num => num > p)))
7  }
```

- **Worstcase:** $O(n^2)$ (mit `qsort([1,2,3,4,5])`)
  - $n + (n-1) + (n-2) + (n-3) + (n-4) + (n-5) = \frac{n \cdot (n-1)}{2} \in O(n^2)$
- **Average case:** $O(n \cdot log(n))$
  - `input.filter()` ist $O(n)$
  - $log(n)$ rekursive Aufrufe (Annahme: Liste **halbiert** sich pro Aufruf von `qsort()`).

QuickSort

Stark vereinfachte Implementierung:

```
1 function qsort(input) {
2   if (input.length == 0) return []
3   let p = input.shift()
4   return qsort(input.filter(num => num <= p))
5     .concat([p])
6     .concat(qsort(input.filter(num => num > p)))
7 }
```

- **Worstcase:** $O(n^2)$ (mit `qsort([1,2,3,4,5])`)
  - $n + (n-1) + (n-2) + (n-3) + (n-4) + (n-5) = \frac{n \cdot (n-1)}{2} \in O(n^2)$
- **Average case:** $O(n \cdot log(n))$
  - `input.filter()` ist $O(n)$
  - $log(n)$ rekursive Aufrufe (Annahme: Liste **halbiert** sich pro Aufruf von `qsort()`).

Speicherkomplexität

Definition

Bestimmung des Speicherbedarfs einer Funktion in Abhängigkeit der Eingabe

Es gilt: $snens \in O(n)$

```
1 function rangeSum(n) {
2   let c = 0;
3   for (let i = 0; i <= n; i++) {
4     c += i;
5   }
6   return c;
7 }
```

Zeitkomplexität: $O(n)$
Speicherkomplexität: $O(1)$

Speicherkomplexität

Definition

Bestimmung des Speicherbedarfs einer Funktion in Abhängigkeit der Eingabe

Es gilt: $snens \in O(n)$

```
1 function rangeSum(n) {
2   let c = 0;
3   for (let i = 0; i <= n; i++) {
4     c += i;
5   }
6   return c;
7 }
```

Zeitkomplexität: $O(n)$
Speicherkomplexität: $O(1)$

Speicherkomplexität

Definition

Bestimmung des Speicherbedarfs einer Funktion in Abhängigkeit der Eingabe

Es gilt: $snens \in O(n)$

```
function rangeSum(n) {
  let c = 0;
  for (let i = 0; i <= n; i++) {
    c += i;
  }
  return c;
}
```

Zeitkomplexität: $O(n)$
Speicherkomplexität: $O(1)$

Definition

Bestimmung des Speicherbedarfs einer Funktion in Abhängigkeit der Eingabe

Es gilt: $snens \in O(n)$

```
1 function rangeSum(n) {
2   let c = 0;
3   for (let i = 0; i <= n; i++) {
4     c += i;
5   }
6   return c;
7 }
```

Zeitkomplexität: $O(n)$

Speicherkomplexität: $O(1)$

Speicherkomplexität

Definition

Bestimmung des Speicherbedarfs einer Funktion in Abhängigkeit der Eingabe

Es gilt: $snens \in O(n)$

```
1 function rangeSum(n) {
2   let c = 0;
3   for (let i = 0; i <= n; i++) {
4     c += i;
5   }
6   return c;
7 }
```

Zeitkomplexität: $O(n)$
Speicherkomplexität: $O(1)$

$O(1)$

```
1  function o_one(object, key) {
2    return object[key]
3  }
```

$O(log(n))$

- Suche in einem Telefonbuch
- **git bisect**
- **anschaulich:** für eine doppelt so große Eingabe steigt der Aufwand konstant.

$O(log(n))$

- Suche in einem Telefonbuch
- **git bisect**
- **anschaulich:** für eine doppelt so große Eingabe steigt der Aufwand konstant.

$O(log(n))$

- Suche in einem Telefonbuch
- **git bisect**
- **anschaulich:** für eine doppelt so große Eingabe steigt der Aufwand konstant.

$O(n^2)$

```
1  function o_square(n) {
2    for (let i = 0; i < n; i++) {
3      for (let j = 0; j < n; j++) {
4        // do sth.
5      }
6    }
7  }
```

$O(n \cdot m)$

```
1  function o_n_m(lst1, lst2) {
2    let n = lst1.length
3    let m = lst2.length
4    for (let i = 0; i < n; i++) {
5      for (let j = 0; j < m; j++) {
6        console.log(`(${lst1[i]},${lst2[j]})`)
7      }
8    }
9  }
```

$O(x^n)$

vermutl. Lösung NP-schwerer Probleme*.

- Rucksackproblem
- Traveling Salesman

* Prüfung einer Lösung kann polynomiell-schnell geschehen, aber Finden einer Lösung vermutl. in exponentieller Zeit.

$O(x^n)$

vermutl. Lösung NP-schwerer Probleme*.

- Rucksackproblem
- Traveling Salesman

* Prüfung einer Lösung kann polynomiell-schnell geschehen, aber Finden einer Lösung vermutl. in exponentieller Zeit.

Wozu das alles?

- Simple Kodierung für Komplexität.
- Nutzen Bibliotheken gerne, um Laufzeit-/Speicherkomplexität ihrer API zu dokumentieren.
- kann u.U. auch in eigenen Projekten sinnvoll sein.
  - Folgefrage: gibt es Optimierungsmöglichkeiten

Wozu das alles?

- Simple Kodierung für Komplexität.
- Nutzen Bibliotheken gerne, um Laufzeit-/Speicherkomplexität ihrer API zu dokumentieren.
- kann u.U. auch in eigenen Projekten sinnvoll sein.
  - Folgefrage: gibt es Optimierungsmöglichkeiten

Wozu das alles?

- Simple Kodierung für Komplexität.
- Nutzen Bibliotheken gerne, um Laufzeit-/Speicherkomplexität ihrer API zu dokumentieren.
- kann u.U. auch in eigenen Projekten sinnvoll sein.
  - Folgefrage: gibt es Optimierungsmöglichkeiten

Wozu das alles?

- Simple Kodierung für Komplexität.
- Nutzen Bibliotheken gerne, um Laufzeit-/Speicherkomplexität ihrer API zu dokumentieren.
- kann u.U. auch in eigenen Projekten sinnvoll sein.
  - Folgefrage: gibt es Optimierungsmöglichkeiten

Ok tschau

Bedankt für eure Aufmerksamkeit!

Slides unter slides.sind.nicht-so.sexy

Gibts noch Fragen?

Ok tschau

Bedankt für eure Aufmerksamkeit!

Slides unter slides.sind.nicht-so.sexy

Gibts noch Fragen?

Ok tschau

Bedankt für eure Aufmerksamkeit!

Slides unter slides.sind.nicht-so.sexy

Gibts noch Fragen?