Laufzeitabschätzung von Algorithmen

Michał Boroń

25. September 2020

Wie kann man das tun?

CPU: 100 MHz
10^8 Operationen

CPU: 2 GHz
$2 \cdot 10^9$ Operationen

Insertion-Sort

```
Data: Liste A = ⟨a_1, a_2, ..., a_n⟩ der Länge n
for j ← 2 to A.length do
    key ← A[j]
    i ← j − 1
    while i > 0 and A[i] > key do
        A[i + 1] ← A[i]
        i ← i − 1
    end
    A[i + 1] = key
end
```

Algorithm 1: Insertion-Sort

Insertion-Sort

Data: Liste $A = \langle a_1, a_2, ..., a_n \rangle$ der Länge n

for $j \leftarrow 2$ *to* $A.length$ **do**

  $key \leftarrow A[j]$

  $i \leftarrow j - 1$

  while $i > 0$ *and* $A[i] > key$ **do**

    $A[i+1] \leftarrow A[i]$

    $i \leftarrow i - 1$

  end

  $A[i+1] = key$

end

Algorithm 2: Insertion-Sort

Wie viele Operationen müssen wir ausführen?

Insertion-Sort

Data: Liste $A = \langle a_1, a_2, ..., a_n \rangle$ der Länge n

for $j \leftarrow 2$ *to* $A.length$ **do**	
$\quad key \leftarrow A[j]$	$n-1$
$\quad i \leftarrow j-1$	$n-1$
$\quad$ **while** $i > 0$ *and* $A[i] > key$ **do**	
$\quad\quad A[i+1] \leftarrow A[i]$	$\sum_{j=2}^{n} t_j$
$\quad\quad i \leftarrow i-1$	$\sum_{j=2}^{n} t_j$
$\quad$ **end**	
$\quad A[i+1] = key$	$n-1$
end	

Insertion-Sort

Die Summe der Operation von Insertion-Sort:

$$T(n) = n - 1 + n - 1 + \sum_{j=2}^{n} (t_j) + \sum_{j=2}^{n} (t_j) + n - 1$$

Insertion-Sort

Die Summe der Operation von Insertion-Sort:

$$T(n) = c_1(n-1) + c_2(n-1) + c_3(\sum_{j=2}^{n}(t_j)) + c_4(\sum_{j=2}^{n}(t_j))$$
$$+ c_5(n-1$$

Wir sollten eigentlich die Kosten der jeweiligen Operation dazu berechnen...

Insertion-Sort

Die Summe der Operation von Insertion-Sort:

$$T(n) = n - 1 + n - 1 + \sum_{j=2}^{n} (t_j) + \sum_{j=2}^{n} (t_j) + n - 1$$

Insertion-Sort

Die Summe der Operation von Insertion-Sort (**worst-case**):

$$T(n) = n - 1 + n - 1 + \sum_{j=2}^{n} (t_j) + \sum_{j=2}^{n} (t_j) + n - 1$$

$$T(n) = n - 1 + n - 1 + \frac{n(n-1)}{2} + \frac{n(n-1)}{2} + n - 1$$

Insertion-Sort

Die Summe der Operation von Insertion-Sort (**worst-case**):

$$T(n) = n - 1 + n - 1 + \sum_{j=2}^{n} (t_j) + \sum_{j=2}^{n} (t_j) + n - 1$$

$$T(n) = n - 1 + n - 1 + \frac{n(n-1)}{2} + \frac{n(n-1)}{2} + n - 1$$

$$T(n) = n(n-1) + 3n - 3$$

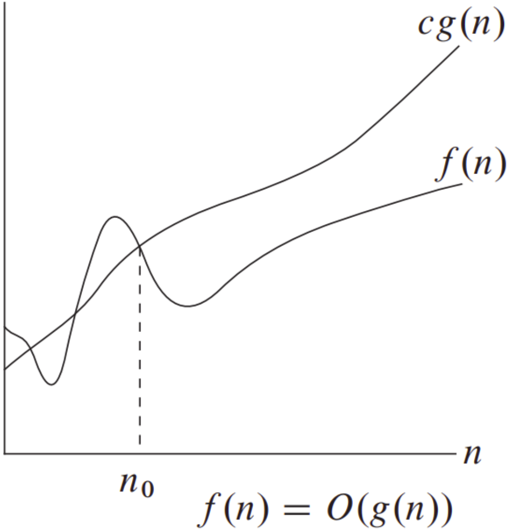
$$T(n) = n^2 + 2n - 3$$

Insertion-Sort

Die Summe der Operation von Insertion-Sort (**worst-case**):

$$
\begin{aligned}
T(n) &= n - 1 + n - 1 + \sum_{j=2}^{n} (t_j) + \sum_{j=2}^{n} (t_j) + n - 1 \\
T(n) &= n - 1 + n - 1 + \frac{n(n-1)}{2} + \frac{n(n-1)}{2} + n - 1 \\
T(n) &= n(n-1) + 3n - 3 \\
T(n) &= n^2 + 2n - 3 \\
T(n) &= an^2 + bn + c
\end{aligned}
$$

$\mathcal{O}$-Notation (deut. *Landau-Symbol*, eng. *Big O Notation*)

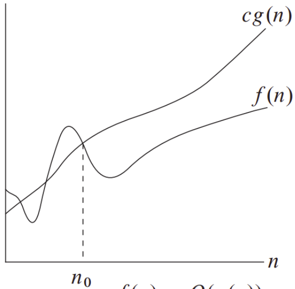


$\mathcal{O}$-Notation

Definition

$\mathcal{O}(g(n)) = \{f(n) : \exists c, n_0 : 0 \leqslant f(n) \leqslant cg(n), \forall n \geqslant n_0\}$

Somit ist $\mathcal{O}(g(n))$ eine **asymptotische obere Schranke**.



$f(n) = O(g(n))$

$\mathcal{O}$-Notation – Insertion-Sort (worst-case)

$$T(n) = an^2 + bn + c$$

Die Laufzeit von Insertion-Sort ist deshalb $\mathcal{O}(n^2)$.

$\mathcal{O}$-Notation

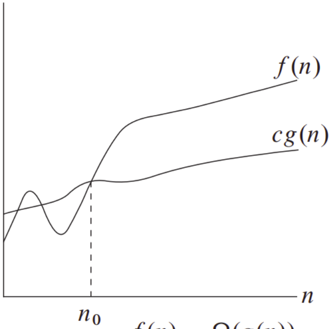
- $f \in O(g(n))$ – f *wächst nicht wesentlich schneller als* g
- Vorfaktoren fallen weg: $O(2n) \rightarrow O(n)$
- Konstanten fallen weg: $O(2n + 4) \rightarrow O(n)$
  - Achtung: wichtige Konstanten oder Faktoren sollen gekennzeichnet werden: $O(100n + 4) \rightarrow O(c \cdot n)$
- Unterscheidung: worst-case und average-case
  - Insertion-Sort (average-case): $O(n)$
  - Insertion-Sort (best-case): $O(n)$
- Wenn wir die *O-Notation* für worst-case anwenden, haben wir eine obere Schranke für **jede mögliche Eingabe**.

Ω-Notation

Definition

$\Omega(g(n)) = \{f(n) : \exists c, n_0 : 0 \leqslant cg(n) \leqslant f(n), \forall n \geqslant n_0\}$

Somit ist $\Omega(g(n))$ eine **asymptotische untere Schranke**.
Die Laufzeit von Insertion-Sort ist $\Omega(n)$.



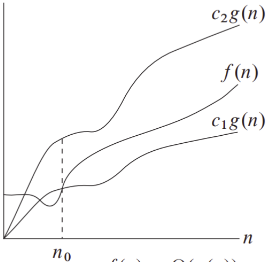
$f(n) = \Omega(g(n))$

Θ-Notation

Definition

$\Theta(g(n)) = \{f(n) : \exists c, n_0 : 0 \leqslant c_1 g(n) \leqslant f(n) \leqslant c_2 g(n), \forall n \geqslant n_0\}$

$\Theta(g(n))$ beschreibt, dass $f(n)$ zwsichen Funktion $c_1 g(n)$ und $c_2 g(n)$ eingekeilt ist.
Die Komplexität von Insertion-Sort ist $\Theta(n^2)$ (worst-case).
In der Literatur wird manchmal mit der *O-Notation* eigentlich die Θ-*Notation* gemeint.



$f(n) = \Theta(g(n))$

Insertion-Sort

	worst-case	average-case	best-case
O			
Ω			
Θ			

Laufzeit in Sekunden

Für: 1s $=$ 100.000.000 Operationen $= 10^8$ Operationen

n	10	1000	10^6
$O(1)$	$< 0,001$	$< 0,001$	$< 0,001$
$O(\log n)$	$< 0,001$	$< 0,001$	$< 0,001$
$O(n)$	$< 0,001$	$< 0,001$	$0,01$
$O(n \log n)$	$< 0,001$	$< 0,001$	$0,03$
$O(n^2)$	$< 0,001$	$0,01$	10000 ($> 2,5$ Std.)
$O(n^3)$	$< 0,001$	10	10^{10} (> 317 Jahre)
$O(2^n)$	$< 0,001$	$> 10^{293}$	∞
$O(n!)$	$0,03$	$> 10^{2559}$	∞

Beispiel 1

Data: eine natürliche Zahl n

$result \leftarrow 0$

for $i \leftarrow 1$ *to* n **do**

  while $i > 1$ **do**

    $result \leftarrow result + i$

    $i \leftarrow i/2$

  end

end

return *result*

Beispiel 2

Data: Liste $A = \langle a_1, a_2, ..., a_n \rangle$ der Länge n
$result \leftarrow 1$
for $i \leftarrow 1$ *to* n **do**
  for $j \leftarrow i$ *to* n **do**
    $sum \leftarrow 0$
    for $k \leftarrow i$ *to* $j + 1$ **do**
      $sum \leftarrow sum + A[k]$
    end
    $result \leftarrow max(sum, result)$
  end
end
return *result*

Aufgabe 1

Aufgabe 1

Gegeben ist eine sortierte Liste A mit n Elementen, $A_i \in \mathbb{Z}$. Gesucht wird ein Element x in a_n. Ausgegeben werden soll der Index von x in A oder -1, falls es dieses Element in der Liste nicht gibt.
Bestimmt und begründet die Laufzeit der binären Suche (in Abhängigkeit von n):

- worst-case (nur $\mathcal{O}$)
- best-case (nur $\mathcal{O}$)

Aufgabe 1

```
Data: Liste A = ⟨a_1, a_2, ..., a_n⟩ der Länge n, eine ganze Zahl x
L ← 1
R ← n
while L ⩽ R do
    m ← ⌊(L+R)/2⌋
    if A[m] < x then
        L ← m + 1
    end
    else if A[m] > x then
        R ← m − 1
    end
    else
        return m
    end
end
return −1
```

Literatur

- Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2009). *Introduction to Algorithms*, Third Edition. The MIT Press. Kapitel 2 und 3.