Programmierung – Nachklausurtutorium

Laryssa Horn, Tim Engelhardt

21 März 2018

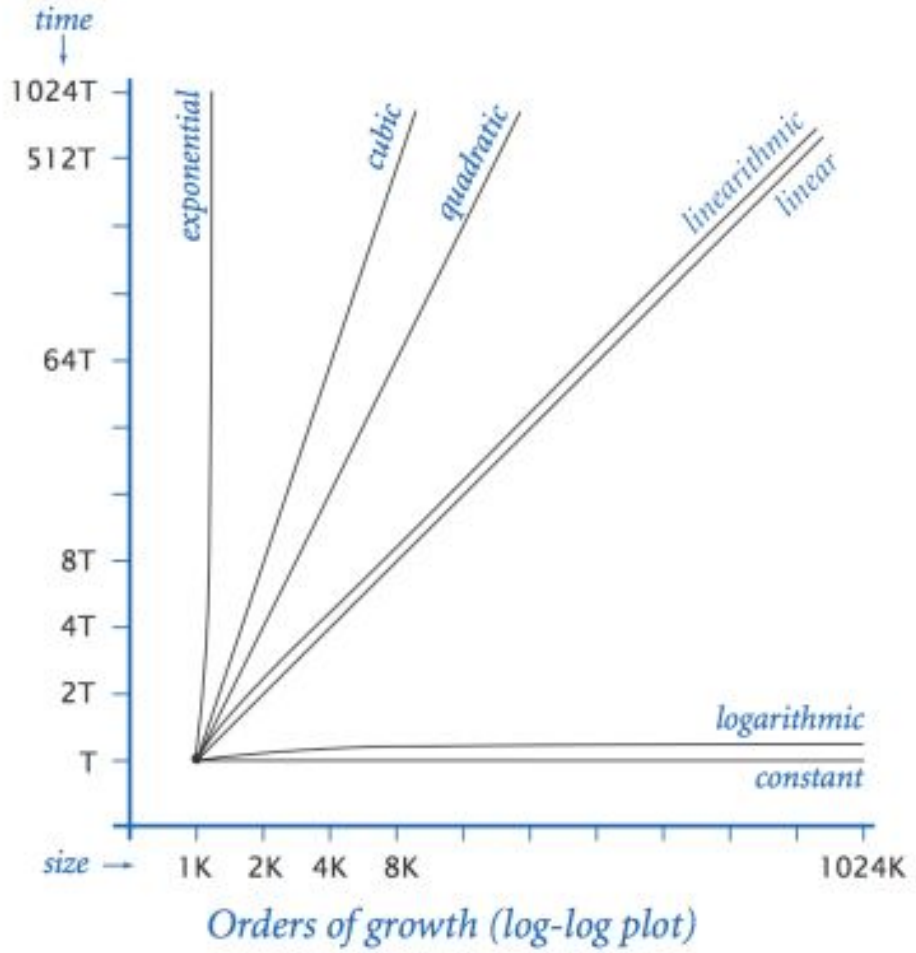
Laufzeitanalyse erste Definitionen

- Laufzeit T(N) : Beschreibt die Zeitdauer, in der ein Programm ausgeführt wird bei N Eingaben
- Laufzeit wird jedoch nicht in einer Zeiteinheit beschrieben -> man sucht die obere Schranke (Maximum) an auszuführenden Operationen
- Verwendet wird hierfür die Landau-Notation

Laufzeitanalyse

- Möglichkeit um die Effizienz eines Programms zu ermitteln
- Basis für Programmvergleiche

Versch. Laufzeiten im Überblick



Orders of growth (log-log plot)

order of growth: description	order of growth: function	factor for doubling hypothesis
constant	1	1
logarithmic	$\log N$	1
linear	N	2
linearithmic	$N \log N$	2
quadratic	N^2	4
cubic	N^3	8
exponential	2^N	2^N

Commonly encountered growth functions

Quelle: Vorlesung Professor Harmeling

Laufzeitanalyse Hinweis

- Konstanten (Ausnahme bei T(N) = 1) sind nicht relevant, man geht von dem am größten wachsenden N aus!

Bsp:

```
int count = 0;
for (int i = 0; i < N; i++)
   if (a[i] == 0) count++;
```

Sollte a[i] immer gleich 0 sein:

- Variablen Deklaration und Initialisierung (T(N) jeweils 1)
- for – Schleife (T(N) = N)
- if – Anweisung (T(N) = N)

-> 2 + N + N = 2N+2 = N

Mini – Quiz

Welche Funktion beschreibt die Laufzeit des folgenden Codes am besten?

```
for(int j = 1; j < n; j++)
  for(int i = 1; i < 10; i++)
    StdOut.println(i*j);
```

- $T(n) = 1$
- $T(n) = \log n$
- $T(n) = n$
- $T(n) = n \log n$
- $T(n) = n^2$

Mini – Quiz 2

Welche Funktion beschreibt die Laufzeit des folgenden Codes am besten?

```
public String something(String s){
  int N = s.length;
  String s2 = "";
  for(int i = 0; i < N; i++){
    s2 = s.charAt(i) + s2;
  }
  return s2;
}
```

- $T(n) = 1$
- $T(n) = \log n$
- $T(n) = n$
- $T(n) = n \log n$
- $T(n) = n^2$

Weitere Beispiele

```
while (N > 1) {
   N = N / 2;
   ...
}
```

$\lg N$ (*logarithmic*)

$\lg N = \log_2 N$

```
for (int i = 0; i < N; i++)
   ...
```

N (*linear*)

```
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      ...
```

N^2 (*quadratic*)

```
public static void g(int N) {
   if (N == 0) return;
   g(N/2);
   g(N/2);
   for (int i = 0; i < N; i++)
      ...
}
```

$N \lg N$ (*linearithmic*)

```
public static void f(int N) {
   if (N == 0) return;
   f(N-1);
   f(N-1);
   ...
}
```

2^N (*exponential*)

Quelle: Vorlesung 2017/18 Prof. Harmeling

Suchverfahren

Sequentielle/lineare Suche

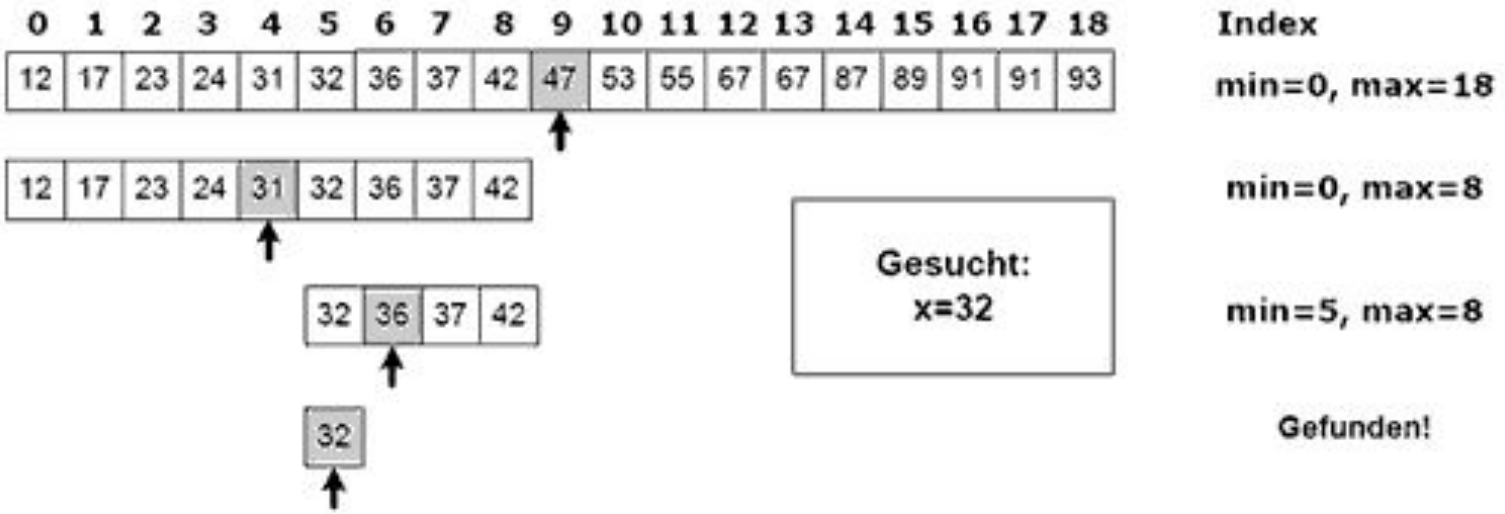
- Vergleiche jeden Wert mit dem gesuchten Wert

```
public static int search(String key, String[] a) {
   int N = a.length;
   for (int i = 0; i < a.length; i++)
      if (a[i].compareTo(key) == 0)
         return i;
   return -1;

}
```

Quelle: Vorlesung 2017/18 Prof. Harmeling

Binäre Suche



- Vergleiche das Element in der „Mitte“ der Datenstruktur (hier Array) mit dem gesuchten Wert, wiederhole diesen Vorgang rekursiv in der linken/ rechten Hälfte bis der Wert gefunden wurde
- Voraussetzung: Datenstruktur ist sortiert

Quelle:
http://hki.uni-koeln.de/archive/hki2016/wisem-2011/basisinformationstechnologie-i/programmiersprachen-iii-algorithmen-und-datenstrukturen/binaere-suche.html

Vergleich Laufzeiten

- Sequentielle Suche: Im Worst-Case T(N)=N
  (jedes Element muss überprüft werden)

- Binäre Suche: T(N) = logN

(halbiere N bis zuletzt ein Element übrig bleibt)

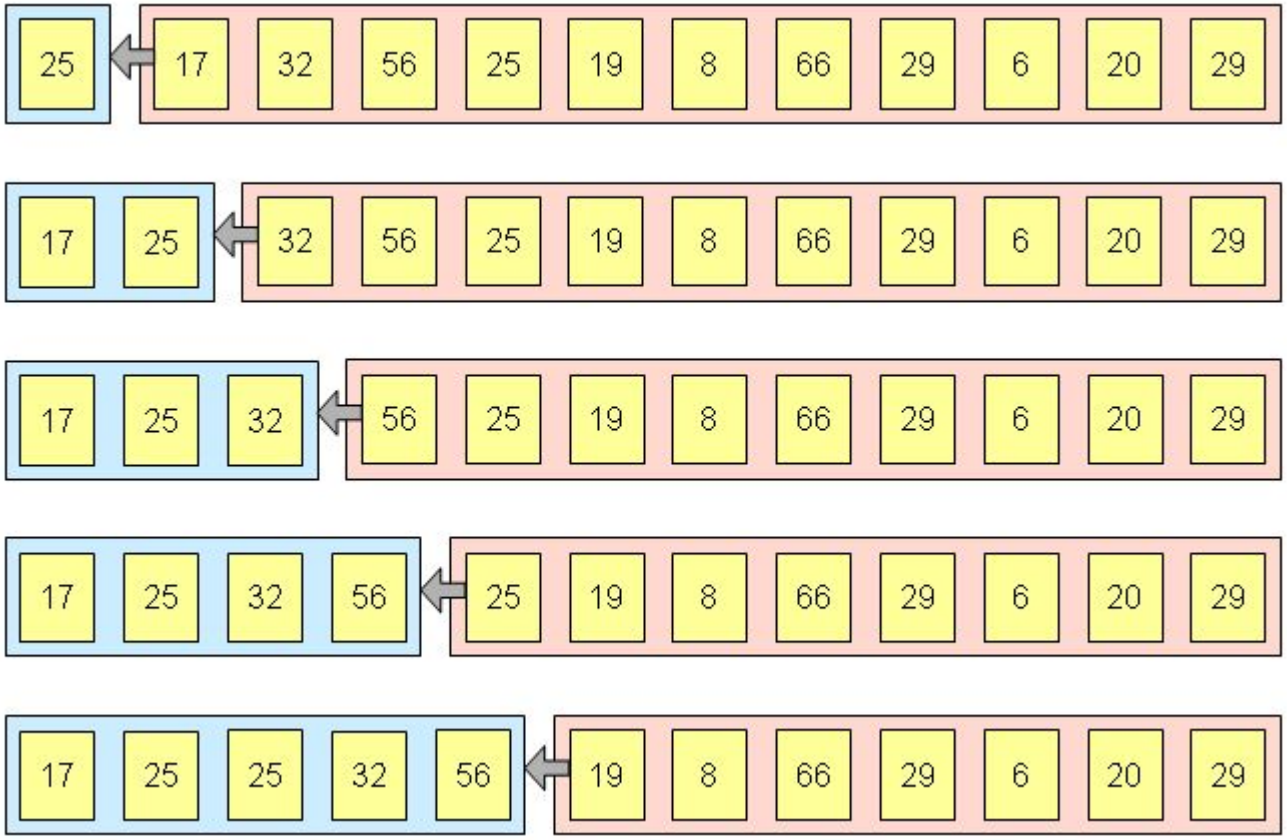
$$
\begin{aligned}
1 \\
2 &\to 1 \\
4 &\to 2 \to 1 \\
8 &\to 4 \to 2 \to 1 \\
16 &\to 8 \to 4 \to 2 \to 1 \\
32 &\to 16 \to 8 \to 4 \to 2 \to 1 \\
64 &\to 32 \to 16 \to 8 \to 4 \to 2 \to 1 \\
128 &\to 64 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1 \\
256 &\to 128 \to 64 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1 \\
512 &\to 256 \to 128 \to 64 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1 \\
1024 &\to 512 \to 256 \to 128 \to 64 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1
\end{aligned}
$$

Quelle: Vorlesung 2017/18 Prof. Harmeling

Sortierverfahren

Insertionsort

Vergleiche jedes Element mit dem vorherigen bis es an richtiger Stelle steht



Laufzeit

Welche Laufzeit hat das Verfahren Insertionsort im besten/schlechtesten Fall?

- Im besten Fall: $T(N) = N$
- Im schlechtesten Fall: $T(N) = N^2$

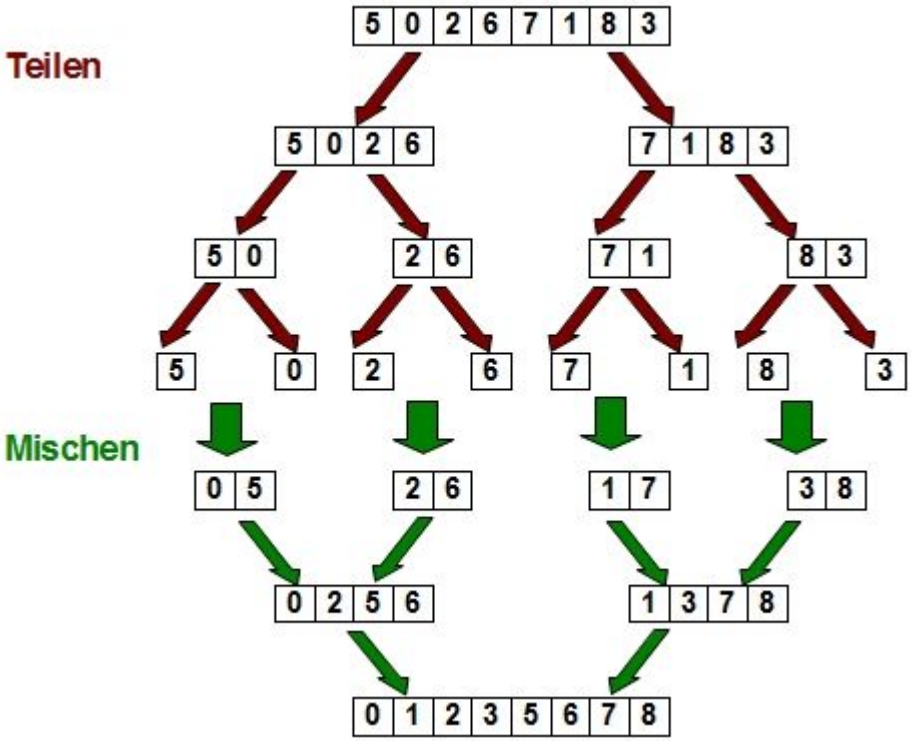
Laufzeit Quiz

Wann tritt die schlechteste Laufzeit im Insertionsort ein?

Mergesort

- Halbiere den Array jeweils in der Mitte
- Sortiere Rekursiv jeweils beide Hälften bis jeweils ein Teil-Array nur noch ein Element besitzt
- Führe alle Teilhälften wieder zusammen

Mergesort



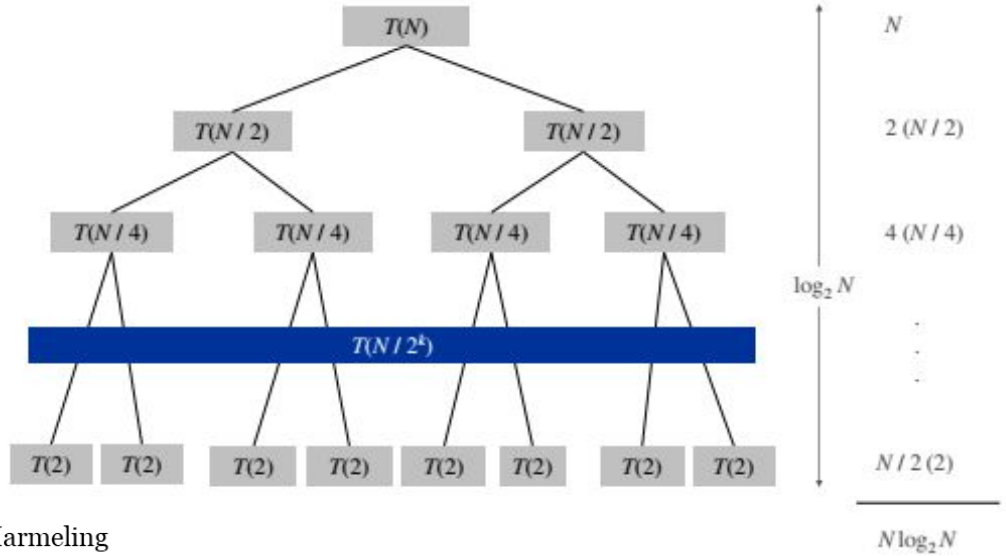
Quelle: http://www.theoretische-informatik.com/mergesort.php

Laufzeit Mergesort

Die Laufzeit beträgt T(N) = N LogN

Erklärung:

- Das Aufteilen des Array benötigt LogN Schritte
- Das Zusammenführen (Vergleich der Werte) benötigt N/2 Schritte pro Array



Quelle: Vorlesung 2017/18 Prof. Harmeling