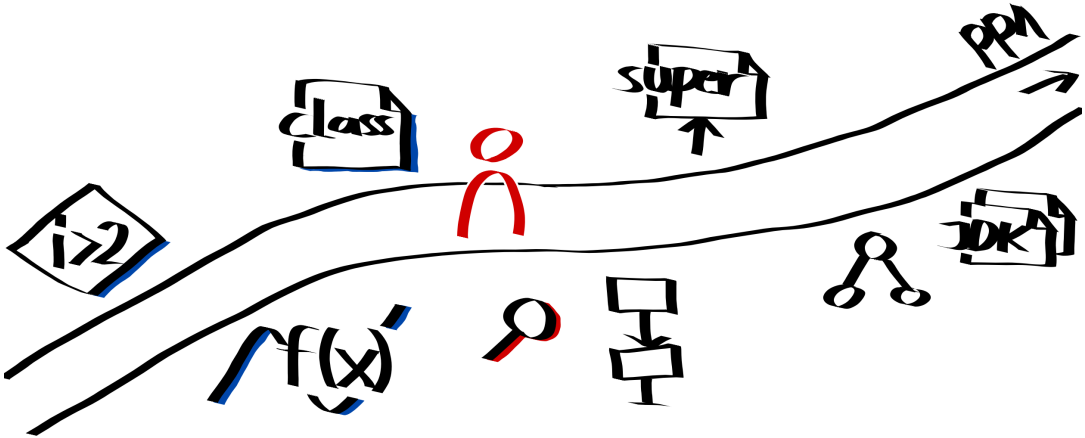
Kapitel 4: Suchen & Sortieren

VL 14: Lineare & Binäre Suche

Wintersemester 2023/24

Stand 21. November 2023, 17:33 Uhr

Wo stehen wir gerade?



Vorlesungsfeedback

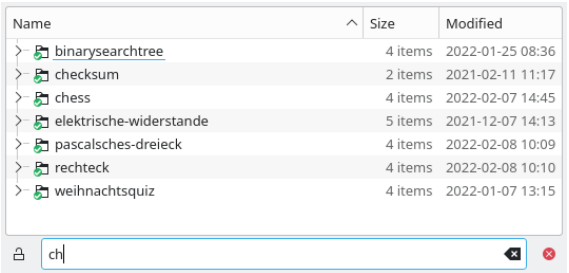
Wo begegnet uns das Thema „Suchen“?

Wo begegnet uns das Thema „Suchen“?

- Telefonbuch: Name gegeben, Telefonnummer gesucht
- E-Mails: Suchwort gegeben, E-Mails mit Wort gesucht
- Übungspunkte: Matrikelnummer gegeben, Summe der Punkte gesucht

→ allgemein: Datenmenge gegeben, bestimmter Eintrag (oder mehrere) zu finden

→ heute: Beschränkung auf Array mit Zahlen, Index einer bestimmten Zahl gesucht



Lineare Suche[1]

- Idee: Durchsuche Array ab erster Position und brich ab, sobald gesuchtes Element gefunden.

```java
private static int linearSearch(int[] haystack, int needle) {
    for(int currentIndex = 0; currentIndex < haystack.length; currentIndex++) {
        if(haystack[currentIndex] == needle) {
            return currentIndex;
        }
    }
    return -1;
}
```

Frage

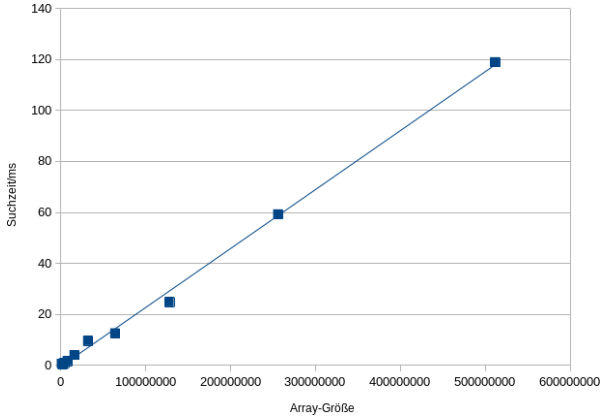
Hat dieser Ansatz ein Problem?

[1] Sequentielle Suche

Lineare Laufzeit

- durchschnittliche Suchdauer steigt linear mit Array-Größe n
- schlechtester Fall: n Vergleiche

$\rightarrow$ egal, wie schnell Computer ist: doppelte Array-Größe $\Rightarrow$ doppelte Suchdauer

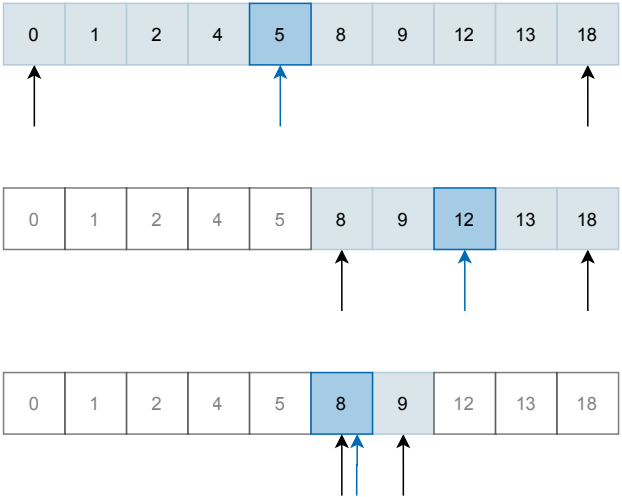


Binäre Suche

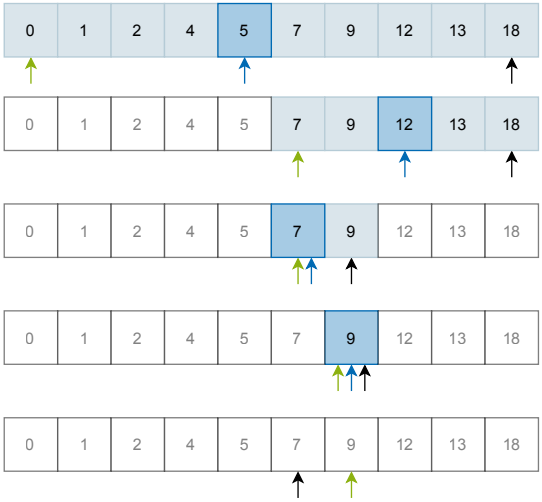
- Idee: schnellere Suche möglich, wenn Array sortiert (vgl. alphabetisches Verzeichnis)
  - starte in Mitte, suche dann in linker oder rechter Hälfte weiter

```java
private static int binarySearch(int[] haystack, int needle) {
    int leftIndex = 0;
    int rightIndex = haystack.length - 1;
    while(leftIndex <= rightIndex) {
        int currentIndex = (leftIndex + rightIndex) / 2;
        int currentElement = haystack[currentIndex];
        if(currentElement == needle) {
            return currentIndex;
        } else if(currentElement < needle) {
            leftIndex = currentIndex + 1;
        } else {
            rightIndex = currentIndex - 1;
        }
    }
    return -1;
}
```

Beispiel: Suche nach der Zahl 8



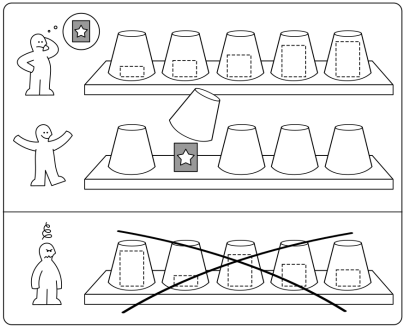
Beispiel: erfolglose Suche nach der Zahl 8



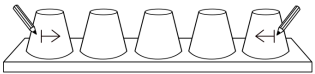
BINÄRY SEARCH

IDEA

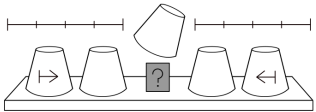
hhu
Heinrich Heine
Universität Düsseldorf



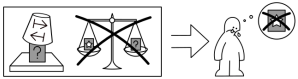
1



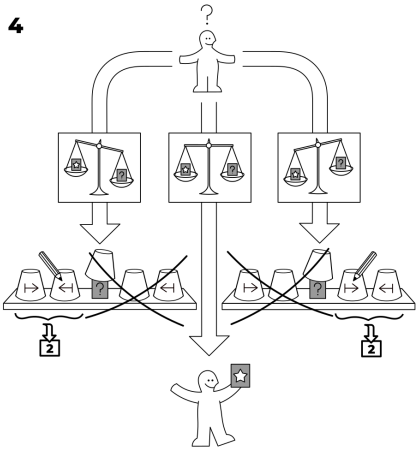
2



3



4



Was war noch gleich der Logarithmus?

Heinrich Heine Universität Düsseldorf

Definition

Der **Logarithmus** von a zur Basis b ist die Zahl, mit der b potenziert werden muss, um a zu erhalten.

$$log_b(a) = x \iff b^x = a$$

Beispiel:

$$log_2(8) =$$

Was war noch gleich der Logarithmus?

Heinrich Heine
Universität Düsseldorf

Definition

Der **Logarithmus** von a zur Basis b ist die Zahl, mit der b potenziert werden muss, um a zu erhalten.

$$log_b(a) = x \iff b^x = a$$

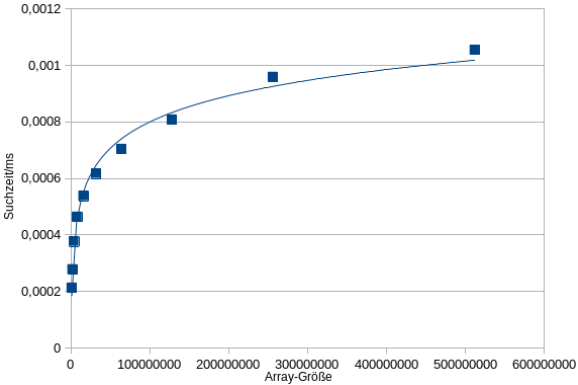
Beispiel:

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

Ist das jetzt schneller?

- durchschnittliche Suchdauer steigt logarithmisch mit Array-Größe n
- schlechtester Fall: $\log_2(n) + 1$ Vergleiche

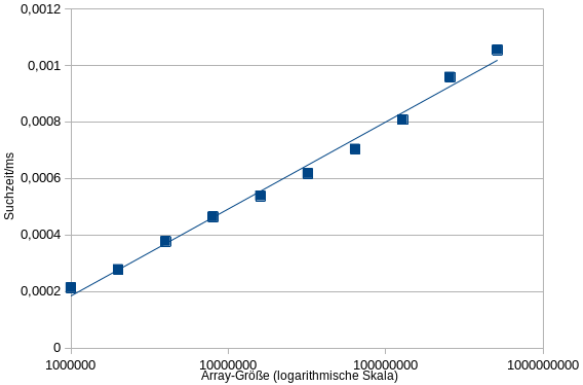
$\rightarrow$ egal, wie schnell Computer ist: Verdopplung der Array-Größe $\Rightarrow$ Suchdauer um Konstante größer



Ist das jetzt schneller?

- durchschnittliche Suchdauer steigt logarithmisch mit Array-Größe n
- schlechtester Fall: $\log_2(n) + 1$ Vergleiche

$\rightarrow$ egal, wie schnell Computer ist: Verdopplung der Array-Größe $\Rightarrow$ Suchdauer um Konstante größer



Woher kommt der Logarithmus?

- Nach jedem Vergleich Suchraum halbiert
- Wie oft kann man n halbieren, bis 1 erreicht?

Woher kommt der Logarithmus?

- Nach jedem Vergleich Suchraum halbiert
- Wie oft kann man n halbieren, bis 1 erreicht?
  - $\rightarrow$ ungefähr $\log_2(n)$-mal

$$
\begin{aligned}
2 &\rightarrow 1 & \quad log_2(2) &= 1 \\
4 &\rightarrow 2 \rightarrow 1 & \quad log_2(4) &= 2 \\
8 &\rightarrow 4 \rightarrow 2 \rightarrow 1 & \quad log_2(8) &= 3 \\
16 &\rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 & \quad log_2(16) &= 4 \\
32 &\rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 & \quad log_2(32) &= 5 \\
64 &\rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 & \quad log_2(64) &= 6
\end{aligned}
$$

Lineare vs. Logarithmische Laufzeit[2]

hhu
Heinrich Heine
Universität Düsseldorf

Merke

Die Logarithmus-Funktion wächst immer langsamer als eine lineare Funktion.

⇒ Jeder Algorithmus mit logarithmischer Laufzeit ist *unabhängig von der verwendeten Hardware* ab einer bestimmten Problemgröße schneller als ein Algorithmus mit linearer Laufzeit, der dasselbe Problem löst.

- Praxis:
  - zuerst funktionierende Lösung
  - falls Laufzeit kritisch: optimieren

[2] Formal gibt man Laufzeiten in der $\mathcal{O}$-Notation an. Das ist aber Teil der Veranstaltung *Algorithmen und Datenstrukturen*. Lineare Suche hat eine mittlere Laufzeit von $\mathcal{O}(n)$, binäre Suche von $\mathcal{O}(\log n)$.

Hat die binäre Suche trotzdem ein Problem?

Hat die binäre Suche trotzdem ein Problem?

Heinrich Heine
Universität Düsseldorf

- Daten müssen erstmal sortiert werden

→ nächste Vorlesung

Aber kann Java nicht schon suchen?

Heinrich Heine
Universität Düsseldorf

[3] https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html#binarySearch-int:A-int-
[4] https://docs.oracle.com/javase/8/docs/api/java/util/List.html#indexOf-java.lang.Object-

Aber kann Java nicht schon suchen?

Heinrich Heine Universität Düsseldorf

Ja!

- `Arrays.binarySearch`[3] und `List.indexOf`[4]

```java
import java.util.Arrays;

public class ArraySearch {
    public static void main(String[] args) {
        int[] numbers = {1, 5, 7, 12, 42, 56};
        int index = Arrays.binarySearch(numbers, 7);
        System.out.println(index);
    }
}
```

```
% java ArraySearch
2
```

[3] https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html#binarySearch-int:A-int-
[4] https://docs.oracle.com/javase/8/docs/api/java/util/List.html#indexOf-java.lang.Object-

Warum haben wir das dann selbst gemacht?

Warum haben wir das dann selbst gemacht?

Heinrich Heine
Universität Düsseldorf

- wollen wissen, wie `Arrays.binarySearch` funktioniert
- grundlegendes Verständnis, um eigene Algorithmen für eigene Datenstrukturen entwickeln zu können
- Praxis: benutze immer das, was es schon gibt
  → weniger Fehler, Zeitersparnis

Merke

Für die meisten Standardaufgaben (z. B. Element finden) gibt es bereits fertige Implementierungen.

Lernziele

Heinrich Heine Universität Düsseldorf

Sie können am Ende der Woche ...

- lineare Suche **implementieren**.
- **erklären**, warum binäre schneller als lineare Suche ist.
- die Geschwindigkeit bekannter Algorithmen unabhängig von der Hardware **angeben**.
- **erklären**, warum meistens Standardimplementierungen eigenen Implementierungen vorgezogen werden sollten.

Wichtige Begriffe

Lineare Suche

lineare Laufzeit

Binäre Suche

logarithmische Laufzeit

Laufzeit

Übungsblatt

- Zulassung
- Rennsimulation
- Elektrische Widerstände

Anonyme Vorlesungs-Evaluation

- Einladung per E-Mail
- Zweck:
  - Qualitätssicherung über Semester hinweg
  - weitere Verbesserung für die nächsten Jahre
- bitte intensiv teilnehmen (unabhängig von meinen eigenen Feedbackumfragen)