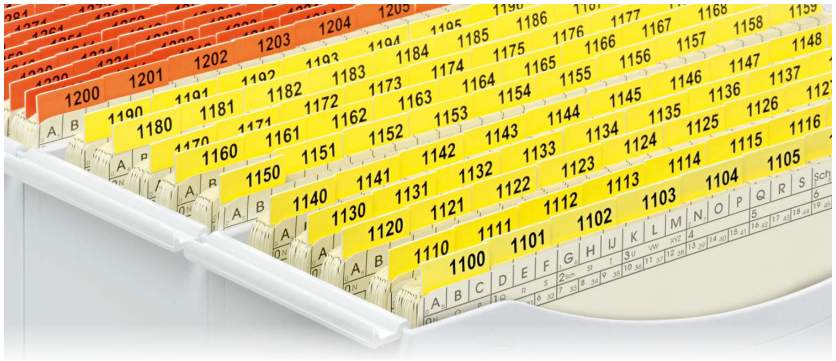
# Ideen und Konzepte der Informatik

# Suchen und Sortieren

„Ordnung ist das halbe Leben“



**Antonios Antoniadis**

**(Basierend auf Folien von Kurt Mehlhorn und Konstantinos Panagiotou)**

**6. November 2017**

# Suchen

- Was ist die Telefonnummer von meinem Lieblingsrestaurant?
- Wie schreibt sich das Wort „Gerechtigkeit“?
- Welche Webseiten enthalten die Wörter „Universität des Saarlandes“? Welche davon ist die wichtigste?

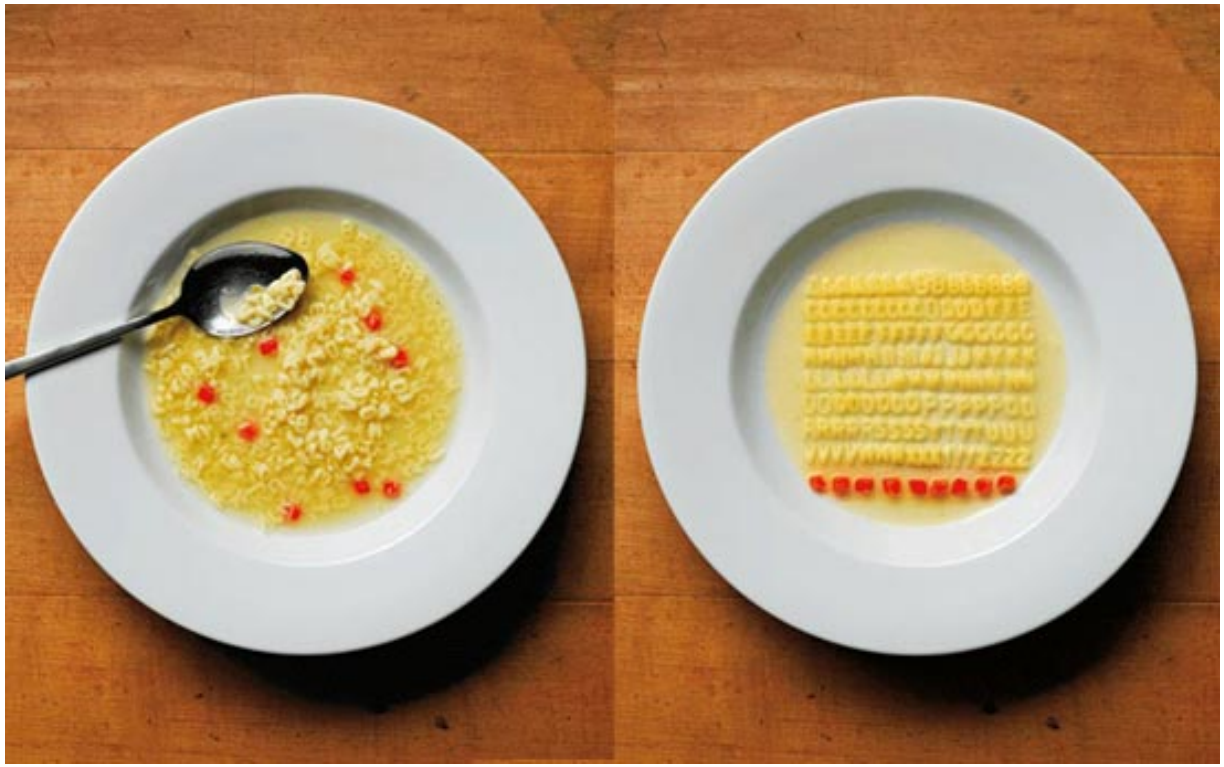
# Suchen

- Was ist die Telefonnummer von meinem Lieblingsrestaurant?
- Wie schreibt sich das Wort „Gerechtigkeit“?
- Welche Webseiten enthalten die Wörter „Universität des Saarlandes“? Welche davon ist die wichtigste?

Ordnung vereinfacht das Suchen!

# Beispiele

Gibt es ein „X“ in der Buchstabensuppe?



In welchem Teller würden Sie lieber suchen?

- **Suchen** und **Sortieren** sind zwei Hauptanwendungen von Computern.

# Thema heute:

- **Suchen:** Versuchen, etwas zu finden. Entweder findet man die Information, oder man kann sagen, dass diese nicht vorhanden ist.
- Ordnung erleichtert das Suchen.
- **Sortieren:** Nach einem Kriterium ordnen
- Datenstrukturen: Suchen in dynamischen Mengen (Mengen, die sich zeitlich verändern).
- Laufzeit (Komplexität) von Algorithmen.

# Übliche Ordnungen

- Zahlen kann man z.B. nach Wert sortieren:

$$2 < 3 < 23 < 42 < 17863.$$

- Wörter z.B. alphabetisch:

$$\text{Antonios} < \text{Christian} < \text{Christoph}$$

# Suchen

- **Daten:** können alles Mögliche sein, z.B. Bilder, Songs, E-Mails, Text.

# Suchen

- **Daten:** können alles Mögliche sein, z.B. Bilder, Songs, E-Mails, Text.
- Ein Beispiel: Name + Telefonnummer:

# Suchen

- **Daten:** können alles Mögliche sein, z.B. Bilder, Songs, E-Mails, Text.
- Ein Beispiel: Name + Telefonnummer:
  Stellen Sie sich vor, dass Sie einen Karteikasten haben, und auf jeder Karte steht ein Name und die dazugehörige Telefonnummer.

# Suchen

- **Daten:** können alles Mögliche sein, z.B. Bilder, Songs, E-Mails, Text.
- Ein Beispiel: Name + Telefonnummer:
  Stellen Sie sich vor, dass Sie einen Karteikasten haben, und auf jeder Karte steht ein Name und die dazugehörige Telefonnummer.
  **Wie viele** Karten müssen Sie sich anschauen um die Telefonnumer zu einem bestimmten Namen zu finden? Im schlimmsten Fall? Im Durchschnitt?

# Grundoperation

Zwei Objekte $a$ und $b$ nach irgendeiner Ordnung vergleichen.
Drei mögliche Fälle:

- $a < b$, $a$ ist kleiner als $b$.
- $a = b$, $a$ und $b$ sind gleich.
- $a > b$, $a$ ist größer als $b$.

# Grundoperation

Zwei Objekte $a$ und $b$ nach irgendeiner Ordnung vergleichen.
Drei mögliche Fälle:

- $a < b$, $a$ ist kleiner als $b$.
- $a = b$, $a$ und $b$ sind gleich.
- $a > b$, $a$ ist größer als $b$.

Wir messen Effizienz in **Anzahl von Vergleichen.** Das ist einfacher, und sagt auch die tatsächliche Laufzeit gut vorher!

# Zurück zum Karteikasten

- Falls Name vorhanden ist, müssen wir uns im Schnitt nur die Hälfte der Karten anschauen. Im schlimmsten Fall allerdings müssen wir uns alle anschauen.
- Falls Name nicht vorhanden ist, müssen wir alle Karten durchgehen.

**Anzahl der angeschauten Karten** = **Anzahl der Vergleiche**

# Das Internet als riesiger Karteikasten

- Mehrere **Billionen** Seiten (1 Billion = 1.000.000.000.000)
- Angenommen wir können pro Sekunde 1.000.000.000 viele Seiten anschauen (was sehr **optimistisch** ist). Dann brauchen wir 1.000 Sekunden (mehr als 16 Minuten) um eine Seite zu finden.
- Wie kann Google das schneller machen?

# Ein sortierter Karteikasten

- Wir sortieren Karteikarten nach Name (wie z.B. im Telefonbuch).
- Wir suchen nach einem Namen X.
- Wir ziehen eine zufällige Karte, darauf steht Name Y.
- Entweder X kommt vor Y, X kommt nach Y oder X$=$Y in der alphabetischen Ordnung.

# Ein sortierter Karteikasten

- Wir sortieren Karteikarten nach Name (wie z.B. im Telefonbuch).
- Wir suchen nach einem Namen X.
- Wir ziehen eine zufällige Karte, darauf steht Name Y.
- Entweder X kommt vor Y, X kommt nach Y oder X$=$Y in der alphabetischen Ordnung.
- Wie hilft uns das weiter? Welche Karte sollten wir als erstes auswählen? Wie geht es weiter?

# Binärsuche

- Gegeben: Liste mit $N$ Elementen, aufsteigend sortiert - also Nachfolger größer als Vorgänger.
- Frage: Enthält die Liste ein Element $x$?

# Binärsuche

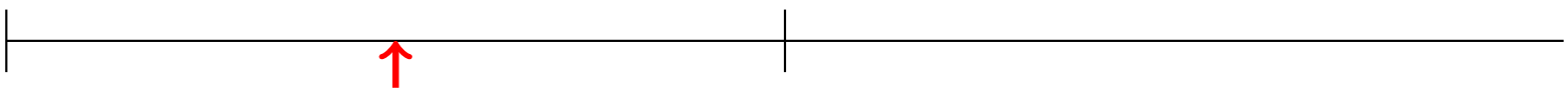
- Gegeben: Liste mit $N$ Elementen, aufsteigend sortiert - also Nachfolger größer als Vorgänger.
- Frage: Enthält die Liste ein Element $x$?

Algorithmus:

# Binärsuche

- Gegeben: Liste mit $N$ Elementen, aufsteigend sortiert - also Nachfolger größer als Vorgänger.
- Frage: Enthält die Liste ein Element $x$?

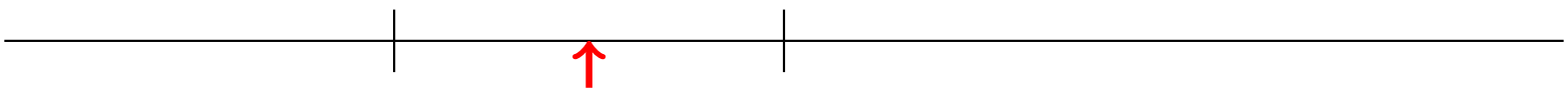
Algorithmus:



# Binärsuche

- Gegeben: Liste mit $N$ Elementen, aufsteigend sortiert - also Nachfolger größer als Vorgänger.
- Frage: Enthält die Liste ein Element $x$?

Algorithmus:



# Binärsuche rekursiv

- Liste/Karteikasten: $L(1), L(2), \ldots L(N)$, $N$ Karten.

# Binärsuche rekursiv

- Liste/Karteikasten: $L(1), L(2), \ldots L(N)$, $N$ Karten.

`Suche(Karteikasten,n)` bestimmt ob `x` im `Karteikasten` vokommt.

**if** (`Karteikasten` leer)
    drucke: `x nicht vorhanden`
**else**
    Sei $m$ der mittlere Index.
    **if** ($x = L(m)$)
        drucke: `x ist vorhanden`
    **else**
        **if** ($x < L(m)$)
            `Suche(Teilkasten der Elemente vor` $L(m)$ `,x)`
        **else**
            `Suche(Teilkasten der Elemente nach` $L(m)$ `,x)`

# Binärsuche Komplexität

## Anzahl der Vergleiche

- $N = 1$, 1 Vergleich,
- $N = 3$, 2 Vergleiche,
- $N = 7$, 3 Vergleiche (1+ Anzahl für $N = 3$),
- $N = 15$, 4 Vergleiche,
- $N = 31$, 5 Vergleiche.

Muster:

- $N = 15 = 2 \cdot 2 \cdot 2 \cdot 2 - 1 = 2^4 - 1$, 4 Vergleiche,
- $N = 31 = 2^5 - 1$, 5 Vergleiche,

# Binärsuche Komplexität

**Anzahl der Vergleiche**

- $N = 1$, 1 Vergleich,
- $N = 3$, 2 Vergleiche,
- $N = 7$, 3 Vergleiche (1+ Anzahl für $N = 3$),
- $N = 15$, 4 Vergleiche,
- $N = 31$, 5 Vergleiche.

Muster:

- $N = 15 = 2 \cdot 2 \cdot 2 \cdot 2 - 1 = 2^4 - 1$, 4 Vergleiche,
- $N = 31 = 2^5 - 1$, 5 Vergleiche,
- $N = 2^{40} - 1$, 40 Vergleiche!!! **Das ist rasend schnell!!!**

Allgemein: Anzahl der Vergleiche = Zweierlogarithmus von $N + 1$. ($\log_2(N + 1)$).

# Vergleich zur Lineare Suche

**Lineare Suche** = Nacheinander alle Karten anschauen, bis $x$ gefunden wird. Funktioniert auch bei unsortierten Daten.
**Binärsuche**: Funktioniert nur wenn man die Daten sortieren kann. (z.B. (Name + Telefonnummer), Webseiten?)

# Vergleich zur Lineare Suche

**Lineare Suche** = Nacheinander alle Karten anschauen, bis $x$ gefunden wird. Funktioniert auch bei unsortierten Daten.
**Binärsuche**: Funktioniert nur wenn man die Daten sortieren kann. (z.B. (Name + Telefonnummer), Webseiten?)

## Aufwand/Komplexität

- Lineare Suche: Anzahl Elemente.
- Binärsuche: Logarithmus der Anzahl der Elemente. Das ist rasend schnell!

# Aber wie Sortieren?

# Aber wie Sortieren?

Angenommen wir wollen zwei sortierte Karteikästen zu einem sortierten Karteikasten zusammenfügen...

# Aber wie Sortieren?

Angenommen wir wollen zwei sortierte Karteikästen zu einem sortierten Karteikasten zusammenfügen...

**Strategie:** Die beide ersten Elemente vergleichen und das kleinere Element zum neuen Kasten hinzufügen. Wiederholen. Wenn ein Kasten erschöpft ist, bewege alle Karten des anderen Kastens.

# Aber wie Sortieren?

Angenommen wir wollen zwei sortierte Karteikästen zu einem sortierten Karteikasten zusammenfügen...

**Strategie:** Die beide ersten Elemente vergleichen und das kleinere Element zum neuen Kasten hinzufügen. Wiederholen. Wenn ein Kasten erschöpft ist, bewege alle Karten des anderen Kastens.

**Anzahl Vergleiche:** Wenn jeder Kasten $n$ Karten enthält dann brauchen wir höchstens $2n - 1$ viele Vergleiche.

# Mischen als Pseudocode

Seien $A[0]$ bis $A[n-1]$ und $B[0]$ bis $B[n-1]$ sortierte Folgen.
$C[0]$ bis $C[2n-1]$ für das Ergebnis bereit.

Setze $i$ und $j$ auf Null.

**Solange** $(i < n)$ oder $(j < n)$
  **Falls** $(i < n$ und $j < n$ und $A[i] < B[j])$ oder $(j = n)$
    bewege $A[i]$ nach $C[i+j]$ und erhöhe $i$ um eins,
  **Sonst**
    bewege $B[j]$ nach $C[i+j]$ und erhöhe $j$ um eins.

# Mischen als Pseudocode

Seien $A[0]$ bis $A[n-1]$ und $B[0]$ bis $B[n-1]$ sortierte Folgen.
$C[0]$ bis $C[2n-1]$ für das Ergebnis bereit.

Setze $i$ und $j$ auf Null.

**Solange** $(i < n)$ oder $(j < n)$
  **Falls** $(i < n$ und $j < n$ und $A[i] < B[j])$ oder $(j = n)$
    bewege $A[i]$ nach $C[i+j]$ und erhöhe $i$ um eins,
  **Sonst**
    bewege $B[j]$ nach $C[i+j]$ und erhöhe $j$ um eins.

Beispiel...

# Sortieren durch Mischen, allgemein

(i) Wir haben $n = 2^k$ viele Elemente und damit $n$ sortierte Folgen der Länge 1.

(ii) Paare je zwei Folgen und mische diese zu einer Folge der doppelten Länge.

(iii) Solange noch mehr als eine Folge, wiederhole.

Jede Ausführung von (ii) braucht höchstens $n$ Vergleiche.
Wie oft wird (ii) ausgeführt?

# Sortieren durch Mischen, allgemein

(i) Wir haben $n = 2^k$ viele Elemente und damit $n$ sortierte Folgen der Länge 1.

(ii) Paare je zwei Folgen und mische diese zu einer Folge der doppelten Länge.

(iii) Solange noch mehr als eine Folge, wiederhole.

Jede Ausführung von (ii) braucht höchstens $n$ Vergleiche.
Wie oft wird (ii) ausgeführt? $k$ Mal

# Sortieren durch Mischen, allgemein

(i) Wir haben $n = 2^k$ viele Elemente und damit $n$ sortierte Folgen der Länge 1.

(ii) Paare je zwei Folgen und mische diese zu einer Folge der doppelten Länge.

(iii) Solange noch mehr als eine Folge, wiederhole.

Jede Ausführung von (ii) braucht höchstens $n$ Vergleiche.
Wie oft wird (ii) ausgeführt? $k$ Mal
Also insgesamt höchstens $n \cdot k = n \cdot \log_2 n$ viele Vergleiche

# Sortieren durch Mischen, praktisch

- $n = 2^{22} \Rightarrow$ 1.09 Sekunden
- $n = 2^{25} \Rightarrow$ 9.94 Sekunden
- $n = 2^{29} \Rightarrow$ 183 Sekunden
- $n = 2^{30} \Rightarrow$ 1240 Sekunden

# Sortieren durch Mischen, praktisch

- $n = 2^{22} \Rightarrow$ 1.09 Sekunden
- $n = 2^{25} \Rightarrow$ 9.94 Sekunden
- $n = 2^{29} \Rightarrow$ 183 Sekunden
- $n = 2^{30} \Rightarrow$ 1240 Sekunden

$$\frac{\text{Anzahl Vergleiche bei } n = 2^{29}}{\text{Anzahl Vergleiche bei } n = 2^{25}} = \frac{2^{29} \cdot \log_2 2^{29}}{2^{25} \cdot \log_2 2^{25}} = 16 \cdot \frac{29}{25} = 18.56$$

$$\frac{\text{Eigentliche Laufzeit bei } n = 2^{29}}{\text{Eigentliche Laufzeit bei } n = 2^{25}} = \frac{183 \text{ Sekunden}}{9.94 \text{ Sekunden}} = 18.41$$

# Sortieren durch Mischen, praktisch

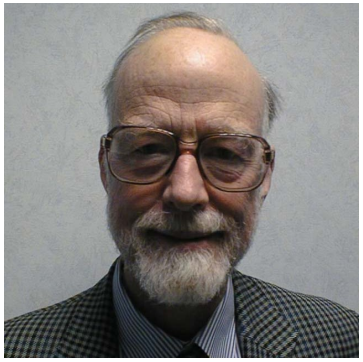
- $n = 2^{22} \Rightarrow$ 1.09 Sekunden
- $n = 2^{25} \Rightarrow$ 9.94 Sekunden
- $n = 2^{29} \Rightarrow$ 183 Sekunden
- $n = 2^{30} \Rightarrow$ 1240 Sekunden

$$\frac{\text{Anzahl Vergleiche bei } n = 2^{29}}{\text{Anzahl Vergleiche bei } n = 2^{25}} = \frac{2^{29} \cdot \log_2 2^{29}}{2^{25} \cdot \log_2 2^{25}} = 16 \cdot \frac{29}{25} = 18.56$$

$$\frac{\text{Eigentliche Laufzeit bei } n = 2^{29}}{\text{Eigentliche Laufzeit bei } n = 2^{25}} = \frac{183 \text{ Sekunden}}{9.94 \text{ Sekunden}} = 18.41$$

- Die Analyse sagt Laufzeitwachstum gut vorher.
- Letzte Zeile - Die Festplatte wurde benutzt.

# Quicksort



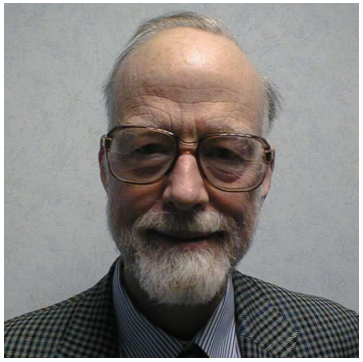
Tony Hoare, Turing Preis 1980

- $S$ : Menge zu sortieren
- Wähle ein Element $s$ in $S$ aus.
- Teile $S$ in:

$S_< =$ Elemente kleiner als $s$.
$S_> =$ Elemente größer als $s$.

- Gib aus: $Sort(S_<), s, Sort(S_>)$
- Rekursion endet wenn $S_<$ und $S_>$ leer sind.

# Quicksort



Tony Hoare, Turing Preis 1980

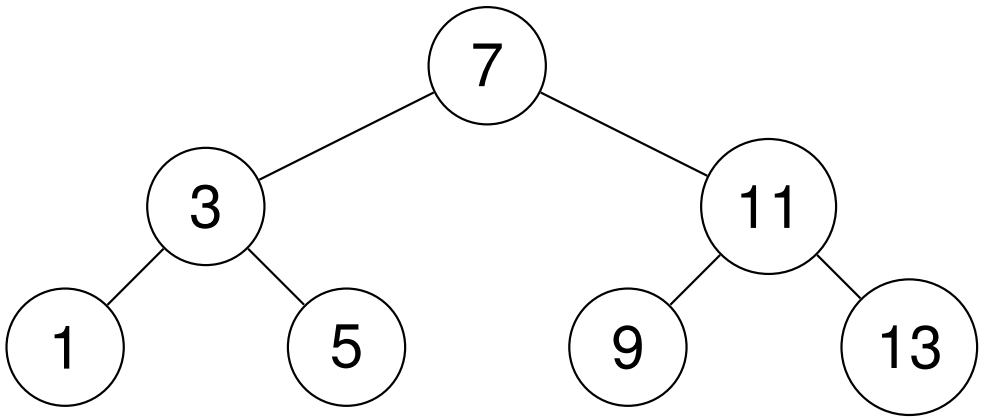
- $S$ : Menge zu sortieren
- Wähle ein Element $s$ in $S$ aus.
- Teile $S$ in:

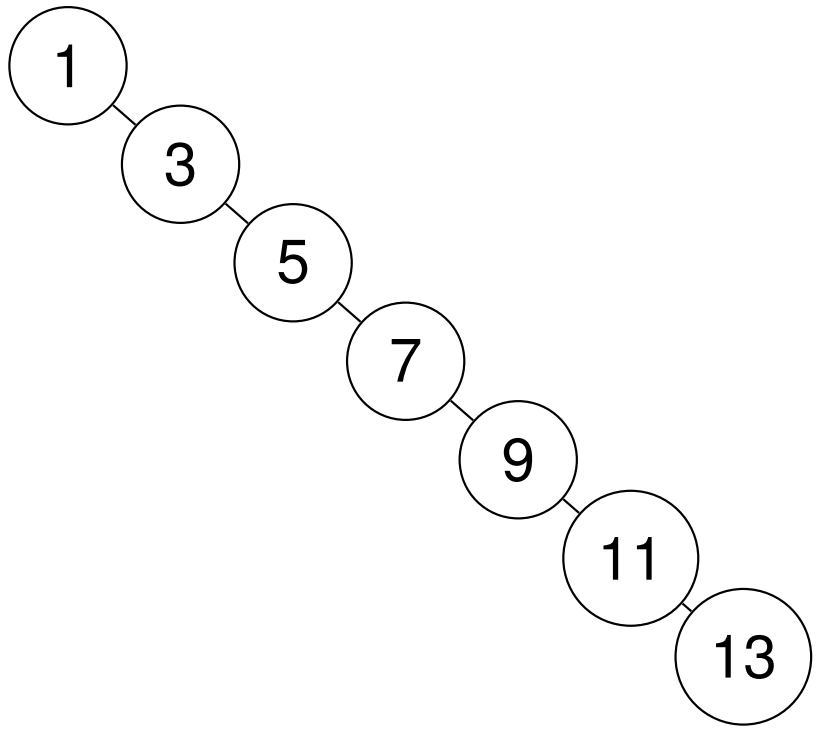
$S_< =$ Elemente kleiner als $s$.
$S_> =$ Elemente größer als $s$.

- Gib aus: $Sort(S_<), s, Sort(S_>)$
- Rekursion endet wenn $S_<$ und $S_>$ leer sind.

Beispielausführung $S = [5, 3, 12, 11, 23, 1, 4, 10, 9]$

# Glück oder Pech




Laufzeit $n \log n$, 0.1sec für $n = 10^6$

Laufzeit $n^2$, 500sec für $n = 10^6$

# Kann man Glück erzwingen?

- Vor 1980: Immer raffiniertere deterministische Algorithmen.
- Seit 1980: Wähle das Teilungselement zufällig. $\Rightarrow$ Randomisierter Algorithmus.

# Suchbäume

Tafelbeispiele:

- Elemente 2,7,3,9,4, Suche nach 9,
- Elemente 11,1,6, Suche nach 8.

*„Ich stelle fest, dass es zwei Wege gibt, ein Software-Design zu erstellen, entweder so einfach, dass es offensichtlich keine Schwächen hat, oder so kompliziert, dass es keine offensichtlichen Schwächen hat. Die erste Methode ist weitaus schwieriger.“*

- Tony Hoare, Dankesrede für den Turingpreis 1980

# Zusammenfassung

- Binärsuche: Rasend schnell, 40 Vergleiche für Suche in einer Billion Elemente!
- Sortieren ist billig: Eine Million Elemente in 0.1sec auf einem privaten Rechner.