

### von Neumann-Architektur

### Speicherorte

```

int main() { // P1m 120
  int buchstaben = 82; // P1m Texter

  void write1() {
    global buchstaben, output;
    global buchstaben, output;
  }

  void loop1() {
    int i = 0;
    while (i < buchstaben) {
      digitalize(buchstaben, i);
      i++;
    }
  }
}

```

### Funktionsstapel

```

double pythagoras(double a, double b) {
  double a_sq = a * a;
  double b_sq = b * b;
  return sqrt(a_sq + b_sq);
}

void loop() {
  double c = pythagoras(4, 8);
}

```



Nächste Woche: Keine Vorlesung, keine Übungen – aber Sprechstunden, und eine große Aufgabe

# Themen heute

- Algorithmen
- Suchen und Sortieren
- Komplexität
- Töne!



## Algorithmus

- eindeutige *Handlungsvorschrift* zur Lösung eines Problems
- besteht aus endlich vielen, wohldefinierten *Einzelschritten*
- werden typischerweise in *Computerprogrammen* implementiert

Quelle: Wikipedia

## Erster Algorithmus

*“Wenn CD aber AB nicht misst, und man nimmt bei AB, CD abwechselnd immer das kleinere vom größeren weg, dann muss (schließlich) eine Zahl übrig bleiben, die die vorangehende misst.”*

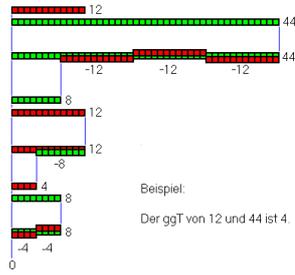
(Euklid, Die Elemente, 300 v. Chr.)

- Berechnet den *größten gemeinsamen Teiler*
- Größte natürliche Zahl, durch die sich AB und CD ohne Rest teilen lassen

Der älteste bekannte Algorithmus (als Rechenvorschrift) geht auf Euklid (3. Jahrhundert v. Chr.) von Alexandria zurück; in seinem Werk “Die Elemente” fasst er das mathematische Wissen seiner Zeit zusammen.

# Euklidscher Algorithmus

“Wenn CD aber AB nicht misst, und man nimmt bei AB, CD abwechselnd immer das kleinere vom größeren weg, dann muss (schließlich) eine Zahl übrig bleiben, die die vorangehende misst.”  
(Euklid, Die Elemente)



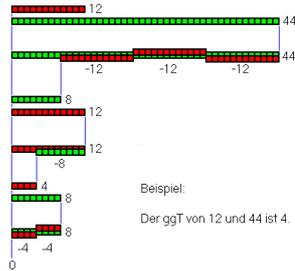
Der älteste bekannte Algorithmus (als Rechenvorschrift) geht auf Euklid (3. Jahrhundert v. Chr.) von Alexandria zurück; in seinem Werk “Die Elemente” fasst er das mathematische Wissen seiner Zeit zusammen.

# Euklidscher Algorithmus

```
int ggt(int a, int b) {
    if (a == 0)
        return b;

    while (b != 0) {
        if (a > b)
            a = a - b;
        else
            b = b - a;
    }

    return a;
}
```



Der älteste bekannte Algorithmus (als Rechenvorschrift) geht auf Euklid (3. Jahrhundert v. Chr.) von Alexandria zurück; in seinem Werk “Die Elemente” fasst er das mathematische Wissen seiner Zeit zusammen.

أبو جعفر محمد بن موسى الخوارزمي

Muhammad ibn Musa al-Chwarizmi, \* ~780 • †835–850



Der Begriff “Algorithmus” geht auf den iranischen Universalgelehrten **Abu Dscha'far Muhammad ibn Musa al-Chwarizmi** zurück. Al-Chwarizmi gilt als einer der bedeutendsten Mathematiker, da er sich – anders als etwa Diophant von Alexandria – nicht mit Zahlentheorie, sondern Algebra als elementarer Untersuchungsform beschäftigte. Auch leistete er bedeutende Beiträge als Geograph und Kartograph, dies auch durch Übersetzungen aus dem Sanskrit und dem Griechischen. (Wikipedia)

# الكتاب المختصر في حساب الجبر والمقابلة

Das kurzgefasste Buch über die  
Rechenverfahren durch Ergänzen und  
Ausgleichen



Im Jahr 830 schloss er die Arbeit an dem Buch **Kitāb al-muchtasar fi hisab al-dschabr wa-l-muqabala** („Rechnen durch Ergänzung und Ausgleich“) ab. Es ist eine Zusammenstellung von Regeln und Beispielen. Sein systematisch-logisches Vorgehen führte zu einer neuen Form von Verständnis für die Lösung linearer und quadratischer Gleichungen, und machte sie für Laien weitaus nachvollziehbarer. Der Begriff „**Algebra**“ wurde aus dem Titel dieses Werkes (**al-dschabr**) abgeleitet. (Wikipedia)

## Erster für Rechner entwickelte Algorithmus

- Berechnet *Bernoulli-Zahlen*

$$B_0 = 1, B_1 = \pm 1/2, B_2 = 1/6, B_3 = 0, B_4 = -1/30, B_5 = 0, B_6 = 1/42, B_7 = 0, B_8 = -1/30.$$

- Folge rationaler Zahlen, die in der Mathematik in verschiedenen Zusammenhängen auftreten
- Beispiel: Summe über Potenzen

$$S_m(n) = \sum_{k=1}^n k^m = 1^m + 2^m + \dots + n^m.$$

$$S_m(n) = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k n^{m+1-k},$$

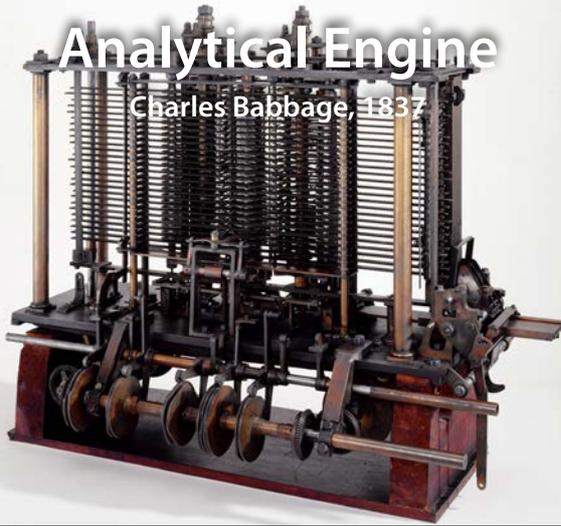
Eigentlich also: Das erste Programm



Augusta Ada Byron King, Countess of Lovelace, allgemein als Ada Lovelace bekannt, war eine britische Mathematikerin. Für einen nie fertiggestellten mechanischen Computer, die Analytical Engine, schrieb sie das erste Programm. **Aus diesem Grund wird sie als – noch vor dem ersten männlichen Kollegen – erste Programmiererin der Welt betrachtet.** Die Programmiersprache Ada[4] und die Lovelace Medal wurden nach ihr benannt.

## Analytical Engine

Charles Babbage, 1837



Die Analytical Engine (englisch für analytische Maschine) ist der Entwurf einer mechanischen Rechenmaschine für allgemeine Anwendungen. Sie stammt von dem britischen Mathematikprofessor Charles Babbage (1791–1871) und stellt einen wichtigen Schritt in der Geschichte des Computers dar.

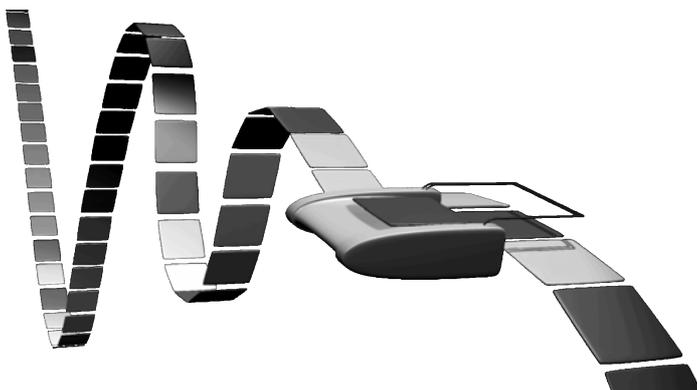
Es ist mittlerweile allgemein anerkannt, dass der Entwurf korrekt war und dass die Analytical Engine funktioniert hätte. Vergleichbare Computer für

## Alan Turing



1936 schließlich führte Turing die Begriffe des Algorithmus und der Berechenbarkeit fassbar, indem er mit seinem Modell die Begriffe des Algorithmus und der Berechenbarkeit als formale, mathematische Begriffe definierte.

## Turingmaschine



Im Allgemeinen wird davon ausgegangen, dass die Turing-Berechenbarkeit das intuitive Verständnis von Berechenbarkeit trifft – alles, was mit einer Turing-Maschine berechnet werden kann, kann überhaupt berechnet werden. (Wikipedia)

# Halteproblem

- Nicht alle Probleme können von Programmen gelöst werden
- Das *Halteproblem* etwa besagt, dass es kein Programm gibt, das für ein beliebiges gegebenes Programm  $P$  entscheidet, ob es ein Ergebnis liefern wird (*hält*) oder nicht.

# Collatz-Problem

(Wolfgang Collatz, 1937)

- Beginne mit einer natürlichen Zahl  $n$
- Ist  $n$  gerade, so nimm als nächstes  $n/2$
- Ist  $n$  ungerade, so nimm als nächstes  $3n+1$
- Wiederhole das Ganze

19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26,  
13, 40, 20, 10, 5, 16, 8, 4, 2, 1, ...

# Collatz-Problem

(Wolfgang Collatz, 1937)

- Anscheinend mündet jede so definierte Folge irgendwann in 4, 2, 1, ...
- Diese Eigenschaft ist unbewiesen

19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26,  
13, 40, 20, 10, 5, 16, 8, 4, 2, 1, ...

# Halteproblem

```
void collatz(int n) {  
  while (n != 1) {  
    if (n % 2 == 0)  
      n = n / 2;  
    else  
      n = 3 * n + 1;  
  }  
}
```

- Wird collatz() für jedes  $n$  halten (zurückkehren)?
- Lösung nur durch *Ausprobieren* (in unendlicher Zeit)

Es gibt keine Möglichkeit, für alle Programme vollautomatisch die Korrektheit zu beweisen

# Halteproblem

Um zu zeigen, dass ein echtes Programm seine Anforderungen erfüllt, müssen wir entweder

- von Hand mathematisches *Wissen und Annahmen* einsetzen, um es zu beweisen, (was sehr aufwändig ist), oder
- wir müssen es *testen* und hoffen, dass unsere Tests ausreichen.

# Algorithmen

Berechnen

- Fibonacci
- GgT
- Collatz

Suchen

- Linear
- Binär

Sortieren

- Einfügen
- Mischen

# Algorithmen

## Berechnen

- Fibonacci
- GgT
- Collatz

## Suchen

- Linear
- Binär

## Sortieren

- Einfügen
- Mischen

# Felder

*a*

0	1	2	3	4	5	6	7	8	9	10

```
int a[11]; // 11 Elemente
```

*Typ der Elemente*      *Name des Feldes*      *Größe des Feldes*

# Suchen

*a*

0	1	2	3	4	5	6	7	8	9	10
10	-10	7	2	2	-4	-7	-10	1	4	5

```
// Wenn x in a[0..size], gib index zurück,  
// so dass x == a[index];  
// sonst < 0  
int find(int a[], int size, int x) {  
    // was passiert hier?  
}
```

# Demo

## Suchen

```
// Wenn x in a[0..size], gib index zurück,  
// so dass x == a[index];  
// sonst < 0  
int find(int a[], int size, int x) {  
    for (int i = 0; i < size; i++) {  
        if (x == a[i])  
            return i;  
    }  
    return -1;  
}
```

## Suchen

*a*

0	1	2	3	4	5	6	7	8	9	10
10	-10	7	2	2	-4	-7	-10	1	4	5

*i*

```
int index = find(a, 11, -4);
```

# Suchen

*a*

0	1	2	3	4	5	6	7	8	9	10
10	-10	7	2	2	-4	-7	-10	1	4	5

*i*

```
int index = find(a, 11, -4);
```

# Suchen

*a*

0	1	2	3	4	5	6	7	8	9	10
10	-10	7	2	2	-4	-7	-10	1	4	5

*i*

```
int index = find(a, 11, -4);
```

# Suchen

*a*

0	1	2	3	4	5	6	7	8	9	10
10	-10	7	2	2	-4	-7	-10	1	4	5

*i*

```
int index = find(a, 11, -4);
```

## Suchen

*a*

0	1	2	3	4	5	6	7	8	9	10
10	-10	7	2	2	-4	-7	-10	1	4	5

*i*

```
int index = find(a, 11, -4);
```

## Suchen

*a*

0	1	2	3	4	5	6	7	8	9	10
10	-10	7	2	2	-4	-7	-10	1	4	5

*i*

```
int index = find(a, 11, -4);
```

## Komplexität

```
// Wenn x in a[0..size], gib index zurück,  
// so dass x == a[index];  
// sonst < 0  
int find(int a[], int size, int x) {  
    for (int i = 0; i < size; i++) {  
        if (x == a[i])  
            return i;  
    }  
    return -1;  
}
```

- Wie viele Vergleiche braucht find()?

**Antwort:** Bei einem Feld der Größe size im Schnitt size/2 Vergleiche.

# Komplexität

Bei  $n$  Elementen:

- Lineare Suche:  $n/2$  Vergleiche (*linear*)
- Was tun bei Millionen von Datensätzen?

Wenn ich bei jeder Suche erst einmal **alle** Daten durchforsten muss, dauert das sehr lange.

## Suchen

$a$

0	1	2	3	4	5	6	7	8	9	10
10	-10	7	2	2	-4	-7	-10	1	4	5

## Suchen

$a$

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4	1	2	2	4	5	7	10

```
// Wenn x in a[0..size], gib index zurück,  
// so dass x == a[index];  
// sonst < 0  
int sorted_find(int a[], int size, int x) {  
    // Kann ich jetzt schneller suchen?  
}
```

# Suchen

*a*

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4	1	2	2	4	5	7	10

*i*

```
int index = sorted_find(a, 11, -4);
```

# Suchen

*a*

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4	1	2	2	4	5	7	10

*i*

```
int index = sorted_find(a, 11, -4);
```

# Suchen

*a*

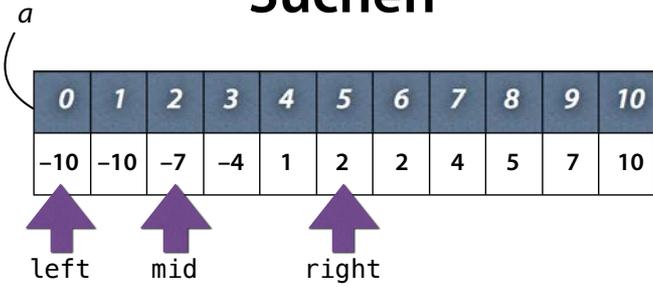
0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4	1	2	2	4	5	7	10

*i* ✓

```
int index = sorted_find(a, 11, -4);
```



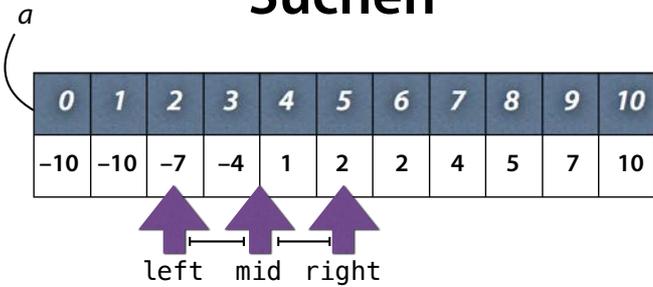
## Suchen



```
int index = sorted_find(a, 11, -4);
```

Die Mitte ist hier kein ganzzahliger Wert; wir runden einfach ab.

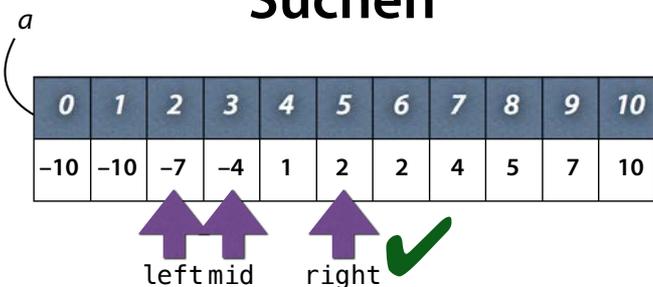
## Suchen



```
int index = sorted_find(a, 11, -4);
```

Jetzt sind wir zu klein; also suchen wir in der rechten Hälfte

## Suchen



```
int index = sorted_find(a, 11, -4);
```

- Wie viele Vergleiche braucht `sorted_find()`?

Nach Abrunden haben wir das gesuchte Element gefunden.

# Demo

## Binäre Suche

```
int sorted_find(int a[], int size, int x) {
    int left = 0;
    int right = size - 1;

    while (left <= right) {
        int mid = left + ((right - left) / 2);
        if (x == a[mid])
            return mid;
        else if (x < a[mid])
            right = mid - 1;
        else // (x > a[mid])
            left = mid + 1;
    }
    return -1;
}
```

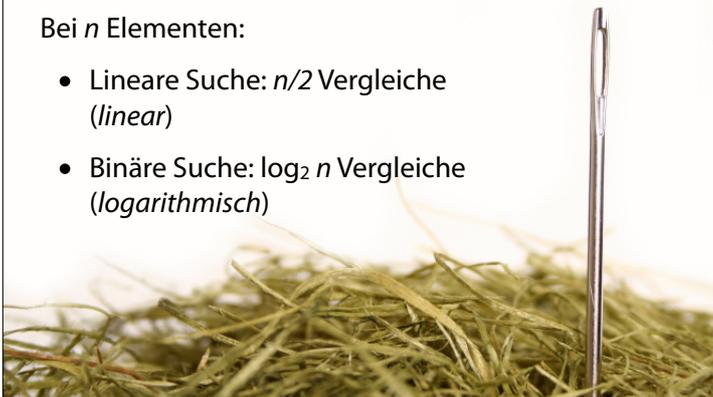
Warum heißt es hier:  $\text{mid} = \text{left} + ((\text{right} - \text{left}) / 2)$  und nicht (einfacher)  $\text{mid} = (\text{left} + \text{right}) / 2$ ? Weil es bei der Berechnung von  $\text{left} + \text{right}$  bei großen Feldern zu einem **Überlauf** kommen kann.

## Komplexität im Vergleich

Bei  $n$  Elementen:

- Lineare Suche:  $n/2$  Vergleiche  
(*linear*)
- Binäre Suche:  $\log_2 n$  Vergleiche  
(*logarithmisch*)

Es gibt noch quadratisch ( $n^2$ ),  
kubisch ( $n^3$ ), polynomiell ( $n^k$ ) und  
exponentiell ( $m^n$ )





Effiziente Suche in großen Datenbeständen ist mittlerweile der Hauptzweck der meisten installierten Server, die Dienste über das Internet bereitstellen. Google alleine betreibt etwa 1 Million solcher Server.

## Algorithmen

### Berechnen

- Fibonacci
- GgT
- Collatz

### Suchen

- Linear
- Binär

### Sortieren

- Einfügen
- Mischen

## Algorithmen

### Berechnen

- Fibonacci
- GgT
- Collatz

### Suchen

- L
- Binär

### Sortieren

- Einfügen
- Mischen

## Sortieren

*a*

0	1	2	3	4	5	6	7	8	9	10
10	-10	7	2	2	-4	-7	-10	1	4	5

## Sortieren

*a*

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4	1	2	2	4	5	7	10

Wie sortiere ich ein Feld?

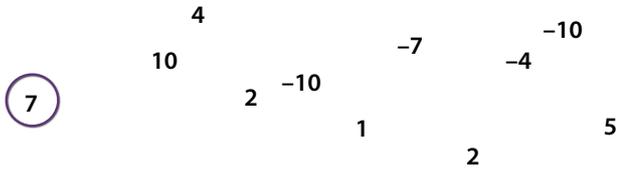
## Sortieren durch Einfügen

*a*

0	1	2	3	4	5	6	7	8	9	10
10	-10	7	2	2	-4	-7	-10	1	4	5

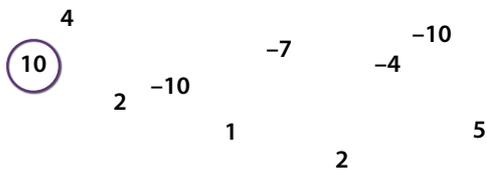
# Sortieren durch Einfügen

0	1	2	3	4	5	6	7	8	9	10



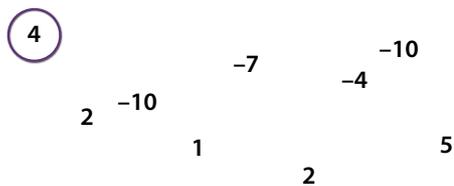
# Sortieren durch Einfügen

0	1	2	3	4	5	6	7	8	9	10
7										



# Sortieren durch Einfügen

0	1	2	3	4	5	6	7	8	9	10
7	10									



# Sortieren durch Einfügen

0	1	2	3	4	5	6	7	8	9	10
7	10									

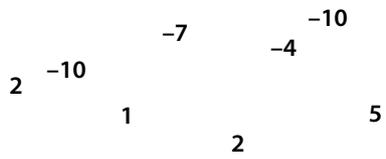
4



# Sortieren durch Einfügen

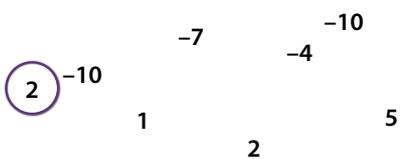
0	1	2	3	4	5	6	7	8	9	10
	7	10								

4



# Sortieren durch Einfügen

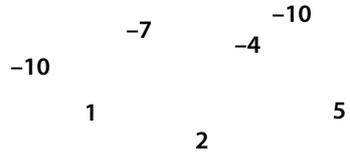
0	1	2	3	4	5	6	7	8	9	10
4	7	10								



# Sortieren durch Einfügen

a

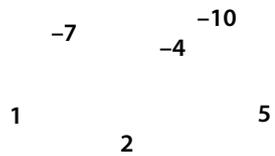
0	1	2	3	4	5	6	7	8	9	10
2	4	7	10							



# Sortieren durch Einfügen

a

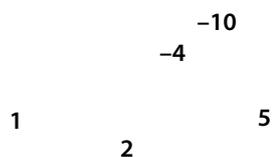
0	1	2	3	4	5	6	7	8	9	10
-10	2	4	7	10						



# Sortieren durch Einfügen

a

0	1	2	3	4	5	6	7	8	9	10
-10	-7	2	4	7	10					



# Sortieren durch Einfügen

a

0	1	2	3	4	5	6	7	8	9	10
-10	-7	1	2	4	7	10				

-10  
-4  
2 5

# Sortieren durch Einfügen

a

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	1	2	4	7	10			

-4  
2 5

# Sortieren durch Einfügen

a

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4	1	2	4	7	10		

2 5

## Sortieren durch Einfügen

*a*

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4	1	2	4	5	7	10	

2

## Sortieren durch Einfügen

*a*

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4	1	2	2	4	5	7	10

## Sortieren vor Ort

*a*

0	1	2	3	4	5	6	7	8	9	10
10	-10	7	2	2	-4	-7	-10	1	4	5

- Wir möchten *innerhalb des Feldes* sortieren
- Annahme: Feld  $a[0 \dots i-1]$  ist bereits sortiert
- Wir betrachten das Element  $a[i] \dots$
- ...und fügen es in das sortierte Feld ein

## Sortieren vor Ort

*a*

0	1	2	3	4	5	6	7	8	9	10
10	-10	7	2	2	-4	-7	-10	1	4	5

—  
bereits  
sortiert

## Sortieren vor Ort

*a*

0	1	2	3	4	5	6	7	8	9	10
-10	10	7	2	2	-4	-7	-10	1	4	5

—  
bereits  
sortiert

## Sortieren vor Ort

*a*

0	1	2	3	4	5	6	7	8	9	10
-10	7	10	2	2	-4	-7	-10	1	4	5

—  
bereits  
sortiert

## Sortieren vor Ort

*a*

0	1	2	3	4	5	6	7	8	9	10
-10	7	2	10	2	-4	-7	-10	1	4	5

bereits  
sortiert

## Sortieren vor Ort

*a*

0	1	2	3	4	5	6	7	8	9	10
-10	2	7	10	2	-4	-7	-10	1	4	5

bereits  
sortiert

- Zum Einfügen *tauschen* wir so lange, bis wir die richtige Position erreicht haben

## Sortieren vor Ort

*a*

0	1	2	3	4	5	6	7	8	9	10
-10	2	7	2	10	-4	-7	-10	1	4	5

bereits  
sortiert

- Zum Einfügen *tauschen* wir so lange, bis wir die richtige Position erreicht haben

## Sortieren vor Ort

*a*

0	1	2	3	4	5	6	7	8	9	10
-10	2	2	7	10	-4	-7	-10	1	4	5

bereits  
sortiert

- Zum Einfügen *tauschen* wir so lange, bis wir die richtige Position erreicht haben

## Sortieren vor Ort

*a*

0	1	2	3	4	5	6	7	8	9	10
-10	2	2	7	-4	10	-7	-10	1	4	5

bereits  
sortiert

- Zum Einfügen *tauschen* wir so lange, bis wir die richtige Position erreicht haben

## Sortieren vor Ort

*a*

0	1	2	3	4	5	6	7	8	9	10
-10	2	2	-4	7	10	-7	-10	1	4	5

bereits  
sortiert

- Zum Einfügen *tauschen* wir so lange, bis wir die richtige Position erreicht haben

## Sortieren vor Ort

*a*

0	1	2	3	4	5	6	7	8	9	10
-10	2	-4	2	7	10	-7	-10	1	4	5

bereits  
sortiert

- Zum Einfügen *tauschen* wir so lange, bis wir die richtige Position erreicht haben

## Sortieren vor Ort

*a*

0	1	2	3	4	5	6	7	8	9	10
-10	-4	2	2	7	10	-7	-10	1	4	5

bereits  
sortiert

- Zum Einfügen *tauschen* wir so lange, bis wir die richtige Position erreicht haben

*Demo*

# Sortieren durch Einfügen

```
void insertion_sort(int a[], int size) {  
    for (int i = 1; i < size; i++) {  
        int j = i;  
        while (j > 0 && a[j - 1] > a[j]) {  
            // Swap a[j] and a[j - 1]  
            int tmp = a[j];  
            a[j] = a[j - 1];  
            a[j - 1] = tmp;  
            j--;  
        }  
    }  
}
```

- Wie viele Vergleiche braucht insertion\_sort()?

# Komplexität im Vergleich

Bei  $n$  Elementen:

- Insertion sort:  $n^2$  Vergleiche  
(quadratisch)

Es gibt noch quadratisch ( $n^2$ ),  
kubisch ( $n^3$ ), polynomiell ( $n^k$ ) und  
exponentiell ( $m^n$ )

# Sortieren durch Mischen

0	1	2	3	4	5	6	7	8	9	10
10	-10	7	2	2	-4	-7	-10	1	4	5

## Sortieren durch Mischen

0	1	2	3	4	5
10	-10	7	2	2	-4

6	7	8	9	10
-7	-10	1	4	5

## Sortieren durch Mischen

0	1	2	3	4	5
10	-10	7	2	2	-4

6	7	8	9	10
-7	-10	1	4	5

## Sortieren durch Mischen

0	1	2	3	4	5
-10	-4	2	2	7	10

6	7	8	9	10
-10	-7	1	4	5

## Sortieren durch Mischen

0	1	2	3	4	5	6	7	8	9	10

0	1	2	3	4	5
-10	-4	2	2	7	10

6	7	8	9	10
-10	-7	1	4	5

## Sortieren durch Mischen

0	1	2	3	4	5	6	7	8	9	10
-10										

0	1	2	3	4	5
	-4	2	2	7	10

6	7	8	9	10
-10	-7	1	4	5

## Sortieren durch Mischen

0	1	2	3	4	5	6	7	8	9	10
-10	-10									

0	1	2	3	4	5
	-4	2	2	7	10

6	7	8	9	10
	-7	1	4	5

## Sortieren durch Mischen

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7								

0	1	2	3	4	5
	-4	2	2	7	10

6	7	8	9	10
		1	4	5

## Sortieren durch Mischen

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4							

0	1	2	3	4	5
		2	2	7	10

6	7	8	9	10
		1	4	5

## Sortieren durch Mischen

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4	1						

0	1	2	3	4	5
		2	2	7	10

6	7	8	9	10
			4	5

## Sortieren durch Mischen

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4	1	2					

0	1	2	3	4	5
			2	7	10

6	7	8	9	10
			4	5

## Sortieren durch Mischen

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4	1	2	2				

0	1	2	3	4	5
				7	10

6	7	8	9	10
			4	5

## Sortieren durch Mischen

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4	1	2	2	4			

0	1	2	3	4	5
				7	10

6	7	8	9	10
				5

## Sortieren durch Mischen

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4	1	2	2	4	5		

0	1	2	3	4	5
				7	10

6	7	8	9	10

## Sortieren durch Mischen

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4	1	2	2	4	5	7	

0	1	2	3	4	5
					10

6	7	8	9	10

## Sortieren durch Mischen

0	1	2	3	4	5	6	7	8	9	10
-10	-10	-7	-4	1	2	2	4	5	7	10

0	1	2	3	4	5

6	7	8	9	10

## Sortieren durch Mischen

0	1	2	3	4	5
10	-10	7	2	2	-4

6	7	8	9	10
-7	-10	1	4	5

## Sortieren durch Mischen

Wie sortiere ich die Teilfelder?

0	1	2	3	4	5
-10	-4	2	2	7	10

6	7	8	9	10
-10	-7	1	4	5

Antwort: Indem ich mergesort() **rekursiv** auf die Teilfelder anwende.

## Rekursives Sortieren

6 5 3 1 8 7 2 4

- Grundidee: mergesort() *rekursiv* auf die Teilfelder anwenden

# John von Neumann



Mergesort wurde von John von Neumann erfunden

*Demo*

## Aufruf

```
// Sort a[0..size], using b[0..size] as a buffer
void merge_sort(int a[], int b[], int size)
{
    partial_merge_sort(a, b, 0, size);
}
```

- Wir nutzen *b[]* als das Hilfsfeld
- Muss gleiche Größe *size* wie *a[]* haben

# Sortieren

```
// Sort a[begin..end], using b[begin..end] as buffer
void partial_merge_sort(int a[], int b[],
                        int begin, int end)
{
    if (end - begin < 2)
        return;

    // Split and sort
    int mid = begin + (end - begin) / 2;
    partial_merge_sort(a, b, begin, mid);
    partial_merge_sort(a, b, mid, end);

    // Merge and copy
    merge(a, b, begin, mid, end);
    copy(a, b, begin, end);
}
```

mid wird wie bei der binären Suche verwendet

# Mischen

```
// Merge a[begin..mid - 1] and a[mid..end] into b[begin..end]
void merge(int a[], int b[], int begin, int mid, int end) {
    int i_begin = begin;
    int i_mid = mid;
    for (int j = begin; j < end; j++)
        if (i_begin < mid &&
            (i_mid >= end || a[i_begin] <= a[i_mid]))
            b[j] = a[i_begin++];
        else
            b[j] = a[i_mid++];
}
```

(Bedingung Stück für Stück entwickeln)

- $P \ \&\& \ Q$  wertet  $Q$  nur aus, wenn  $P$  wahr ist
- $P \ || \ Q$  wertet  $Q$  nur aus, wenn  $P$  falsch ist
- Schützt hier Feldgrenzen vor Zugriff

# Kopieren

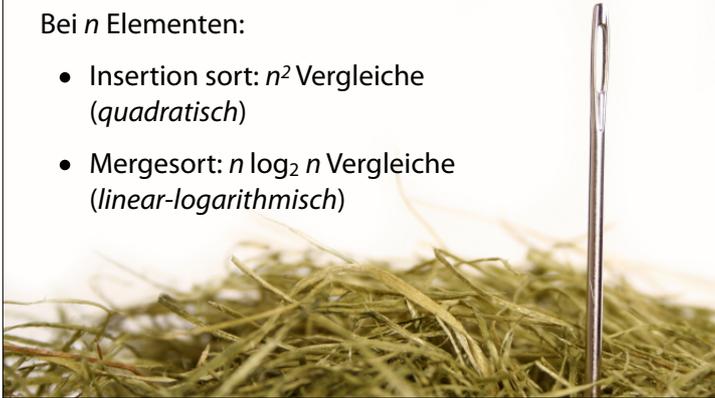
```
// Copy b[begin..end] into a[begin..end]
void copy(int a[], int b[], int begin, int end) {
    for (int k = begin; k < end; k++)
        a[k] = b[k];
}
```

- Wie viele Vergleiche braucht `merge_sort()`?

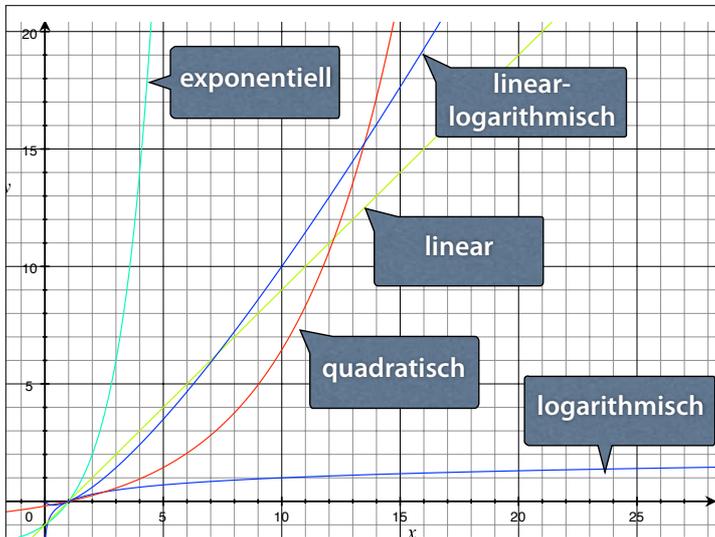
# Komplexität im Vergleich

Bei  $n$  Elementen:

- Insertion sort:  $n^2$  Vergleiche (*quadratisch*)
- Mergesort:  $n \log_2 n$  Vergleiche (*linear-logarithmisch*)



Es gibt noch quadratisch ( $n^2$ ), kubisch ( $n^3$ ), polynomiell ( $n^k$ ) und exponentiell ( $2^n$ )



Man vergleiche die unterschiedlichen Wachstumskurven

# Algorithmen

Berechnen

- Fibonacci
- GgT
- Collatz

Suchen

- L
- Binär

Sortieren

- Einfügen
- Mischen

# Algorithmen

## Berechnen

- Fibonacci
- GgT
- Collatz

## Suchen

- Linear
- Binär

## Sortieren

- Einfügen
- Mischen

# Summer

- Einfacher Piezo-Summer
- Kann Töne in beliebigen Frequenzen ausgeben



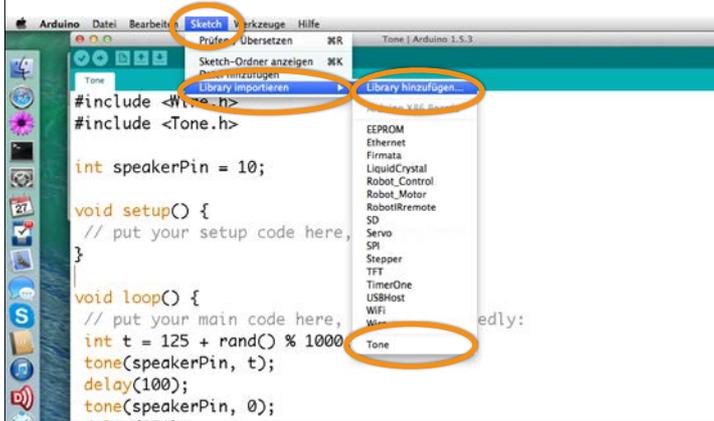
# Tone-Bibliothek



Tone.zip

Verfügbar auf der Kurs-Webseite

# Bibliothek hinzufügen



## Tone-Bibliothek

- Der Summer wird an + und einen *pin* angeschlossen
- Zu Beginn schreiben Sie

```
#include <Wire.h>
#include <Tone.h>
```
- Die Funktion `tone(pin, frequenz)` gibt den Ton *frequenz* aus (0: kein Ton)

*Demo*

# Zufallsmucke

```
#include <Wire.h>
#include <Tone.h>

int speakerPin = 10;

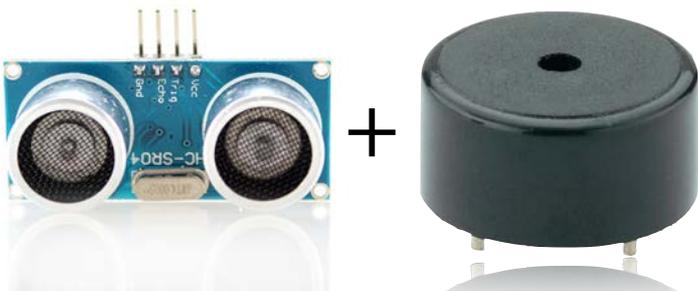
void setup() { }

void loop() {
  int t = 125 + rand() % 1000;
  tone(speakerPin, t);
  delay(100);
  tone(speakerPin, 0);
  delay(250);
}
```

# Theremin



# Theremin =



# Demo



The Big Bang Theory – Sheldon's Theremin – Staffel 4. Das gespielte Lied ist "Nobody knows the trouble I've seen".

[https://www.youtube.com/watch?v=i7v1R7\\_85IY&feature=kp](https://www.youtube.com/watch?v=i7v1R7_85IY&feature=kp)

## Algorithmus

- eindeutige *Handlungsvorschrift* zur Lösung eines Problems
- besteht aus endlich vielen, wohldefinierten Einzelschritten
- werden typischerweise in *Computerprogrammen* implementiert

## Algorithmen

### Berechnen

- Fibonacci
- GgT
- Collatz

### Suchen

- Linear
- Binär

### Sortieren

- Einfügen
- Mischen

## Sortieren vor Ort

0	1	2	3	4	5	6	7	8	9	10
10	-10	7	2	2	-4	-7	-10	1	4	5

- Wir möchten *innerhalb des Feldes* sortieren
- Annhme: Feld  $a[0..i-1]$  ist bereits sortiert
- Wir betrachten das Element  $a[i]$ ...
- ...und fügen es in das sortierte Feld ein

## Theremin =



# Handouts

## Halteproblem

```
void collatz(int n) {  
  while (n != 1) {  
    if (n % 2 == 0)  
      n = n / 2;  
    else  
      n = 3 * n + 1;  
  }  
}
```

- Wird collatz() für jedes  $n$  halten (zurückkehren)?
- Lösung nur durch *Ausprobieren* (in unendlicher Zeit)

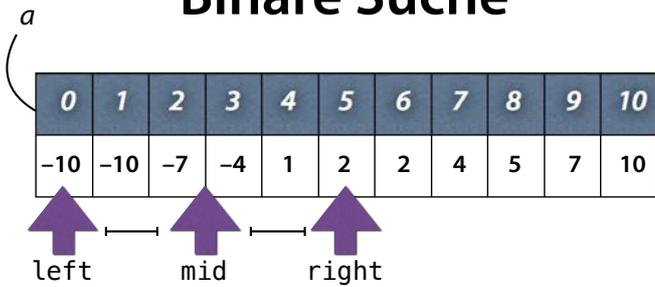
Es gibt keine Möglichkeit, für alle Programme vollautomatisch die Korrektheit zu beweisen

## Binäre Suche

```
int sorted_find(int a[], int size, int x) {  
  int left = 0;  
  int right = size - 1;  
  
  while (left <= right) {  
    int mid = left + ((right - left) / 2);  
    if (x == a[mid])  
      return mid;  
    else if (x < a[mid])  
      right = mid - 1;  
    else // (x > a[mid])  
      left = mid + 1;  
  }  
  return -1;  
}
```

Warum heißt es hier:  $\text{mid} = \text{left} + ((\text{right} - \text{left}) / 2)$  und nicht (einfacher)  $\text{mid} = (\text{left} + \text{right}) / 2$ ? Weil es bei der Berechnung von  $\text{left} + \text{right}$  bei großen Feldern zu einem **Überlauf** kommen kann.

## Binäre Suche



```
int index = sorted_find(a, 11, -4);
```

Wir suchen weiter in der linken Hälfte des Intervalls; dort ebenfalls in der Mitte

## Sortieren durch Einfügen

```
void insertion_sort(int a[], int size) {  
    for (int i = 1; i < size; i++) {  
        int j = i;  
        while (j > 0 && a[j - 1] > a[j]) {  
            // Swap a[j] and a[j - 1]  
            int tmp = a[j];  
            a[j] = a[j - 1];  
            a[j - 1] = tmp;  
            j--;  
        }  
    }  
}
```

## Sortieren durch Mischen

```
// Sort a[0..size], using b[0..size] as a buffer  
void merge_sort(int a[], int b[], int size)  
{  
    partial_merge_sort(a, b, 0, size);  
}
```

- Wir nutzen *b[]* als das Hilfsfeld
- Muss gleiche Größe *size* wie *a[]* haben

## Sortieren

```
// Sort a[begin..end], using b[begin..end] as buffer
void partial_merge_sort(int a[], int b[],
                        int begin, int end)
{
    if (end - begin < 2)
        return;

    // Split and sort
    int mid = begin + (end - begin) / 2;
    partial_merge_sort(a, b, begin, mid);
    partial_merge_sort(a, b, mid, end);

    // Merge and copy
    merge(a, b, begin, mid, end);
    copy(a, b, begin, end);
}
```

mid wird wie bei der binären Suche verwendet

## Mischen

```
// Merge a[begin..mid - 1] and a[mid..end] into b[begin..end]
void merge(int a[], int b[], int begin, int mid, int end) {
    int i_begin = begin;
    int i_mid = mid;
    for (int j = begin; j < end; j++)
        if (i_begin < mid &&
            (i_mid >= end || a[i_begin] <= a[i_mid]))
            b[j] = a[i_begin++];
        else
            b[j] = a[i_mid++];
}
```

- $P \ \&\& \ Q$  wertet  $Q$  nur aus, wenn  $P$  wahr ist
- $P \ || \ Q$  wertet  $Q$  nur aus, wenn  $P$  falsch ist
- Schützt hier Feldgrenzen vor Zugriff

(Bedingung Stück für Stück entwickeln)

## Kopieren

```
// Copy b[begin..end] into a[begin..end]
void copy(int a[], int b[], int begin, int end) {
    for (int k = begin; k < end; k++)
        a[k] = b[k];
}
```