Einführung in die IT

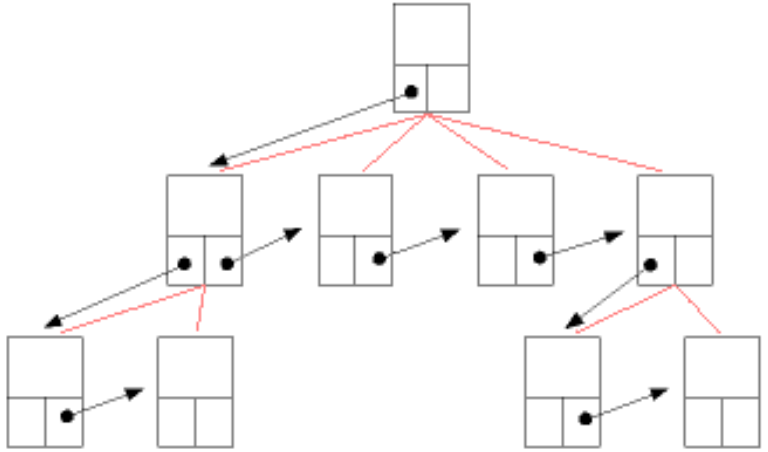
Teil3:
Bäume
Suchen und Sortieren

Bernd Schöner

DHBW Mannheim

Bäume (Trees)/1

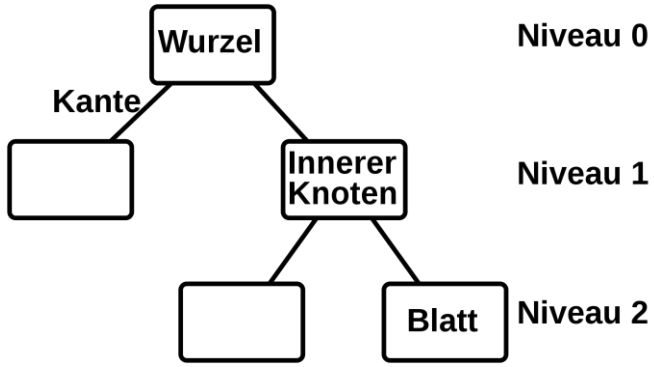
- Ein **Baum** eine Datenstruktur und ein abstrakter Datentyp, mit dem sich hierarchische Strukturen und Beziehungen zwischen Daten abbilden lassen. Sie spielen in der Informatik eine besondere Rolle.
- Ein allgemeiner Baum ist eine Datenstruktur, die aus einer Anzahl von Knoten besteht, die so durch Kanten verbunden sind, dass keine Kreise auftreten.
- Der oberste Knoten des Baumes wird als Wurzel bezeichnet. Ausgehend von der Wurzel (Root) können mehrere gleichartige Objekte miteinander verkettet werden, sodass die lineare Struktur der Liste aufgebrochen wird und eine Verzweigung stattfindet.
- Viele Spezialisierungen, (z.B. Binärbäume, Heaps)
- Der Vorteil von Bäumen gegenüber linearen Strukturen ist der effiziente Zugriff. So erfolgt beispielsweise eine Suche nur in logarithmischer Zeit gegenüber linearer Zeit bei.
- Datensätze entsprechen einer logischen Struktur, die bei der Softwareentwicklung (z. B. im konzeptionellen Schema der Datenmodellierung) festgelegt wurde.



https://commons.wikimedia.org/wiki/File:Allgemeiner-baum.png

Bäume (Trees)/2

- Die Elemente im Baum nennt man Knoten.
- Jeder Knoten speichert ausgehend von einem ersten Knoten, der Wurzel, eine Liste von Verweisen auf die untergeordneten Knoten.
- Diese Verweise heißen Kanten. Die Kanten verbinden die Knoten, welche üblicherweise Informationen tragen, mit ihren Nachfolgern.
- Es ist dann üblich, von Kindern und von einem Elternteil zu sprechen.
- Hat ein Knoten selbst keine Kinder, nennt man ihn ein Blatt.
- Die Tiefe eines Knotens gibt an, wie viele Kanten er von der Wurzel entfernt ist. Die Wurzel hat die Tiefe 0.
- Die Knoten mit derselben Tiefe bilden zusammen ein Niveau.
- Die Höhe eines Baumes ist dann die maximale Tiefe eines Knotens.
- Implementierung als Arrays oder verkettete Listen. Flexibler sind verkettete Listen.



https://commons.wikimedia.org/w/index.php?curid=90617685

Bäume, Beispiel verkettete Speicherung

- Spalte 1: Anfangsadressen
- Spalte 2: Nutzdaten
- Spalte 3: linker Nachfolger
- Spalte 4: rechter Nachfolger
- Endknoten haben keine Nachfolger: 0

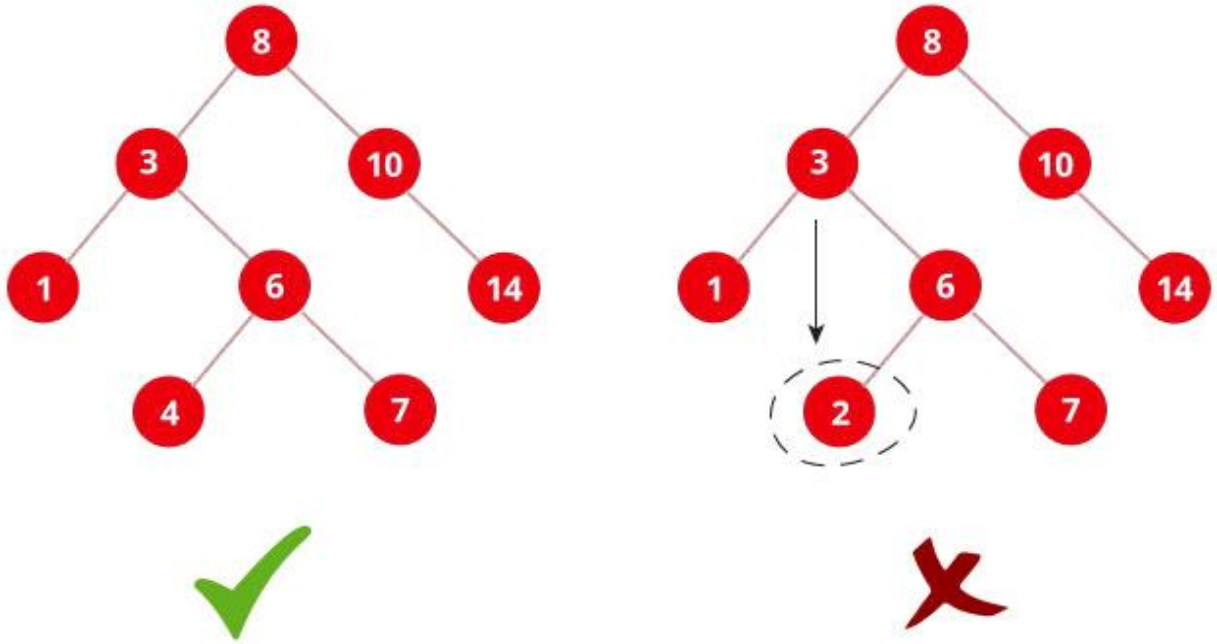
Adresse	Info	links	rechts
10 (W)	*	11	12
11	+	13	14
12	/	15	16
13	2	0	0
14	y	0	0
15	+	17	18
16	-	19	20
17	3	0	0
18	x	0	0
19	4	0	0
20	a	0	0

Malen Sie den Baum!

Binärbäume

- Binärbaum: Jeder Knoten hat max. zwei Kinder, also 0, 1 oder 2 Nachfolger.
- Vollständiger Binärbaum: Jeder Knoten hat exakt zwei Kinder, d.h. alle Blätter haben die gleiche Tiefe.
- Es gibt höchstens 2v Knoten auf derselben Ebene mit Niveauzahl v.
- Suchbaum: Für jeden Knoten k gilt, dass alle Schlüssel im linken Teilbaum von k kleiner als der Schlüssel von k sind, bzw. alle Schlüssel im rechten Teilbaum von k größer oder gleich dem Schlüssel von k sind.

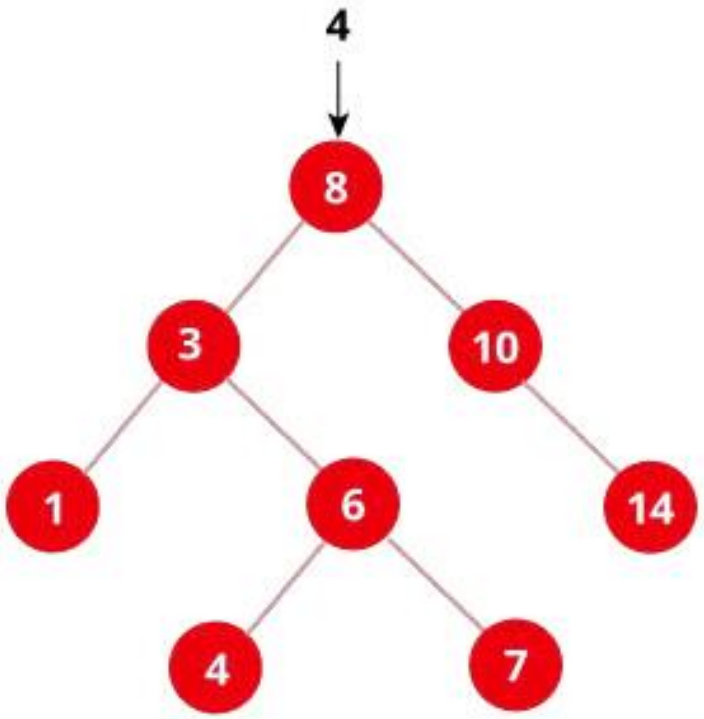
Der rechte Baum ist kein binärer Suchbaum, da sich rechts des Knotens 3 der Schlüssel 2 befindet.

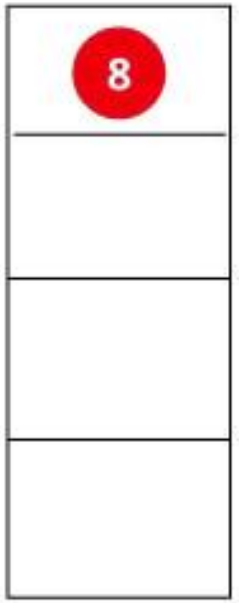


Quelle: programiz.com

Binärer Suchbaum

- Beispiel: Suche nach dem Schlüssel 4



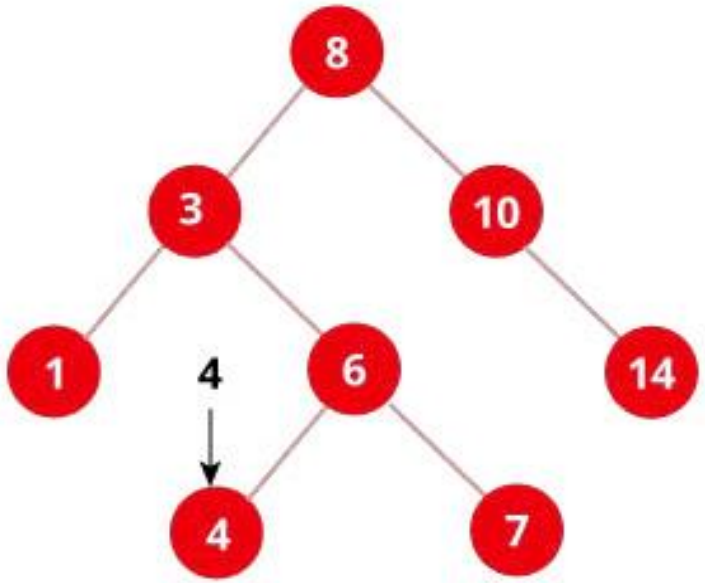


4 < 8 **!** Suche links fortsetzten.

Quelle: programiz.com

Binärer Suchbaum

- Beispiel: Suche nach dem Schlüssel 4. Nach mehreren Schritten.



4 == 4 ! Suche beendet.

Quelle: programiz.com

Suchen und Sortieren

- Das Suchen und Sortieren von Daten sind zwei der grundlegendsten Verfahren der Informatik, besonders bei kaufmännischen Anwendungen.
- Suchen: Es geht dabei um das Auffinden bestimmter Informationen aus einer größeren Menge gespeicherter Daten, die einer oder mehreren Suchbedingungen genügen.
- Durchsuchen: jedes Element eines Datenbestandes soll unter Einhaltung einer bestimmten Reihenfolge genau einmal bearbeitet werden, etwa zur Auflistung aller Elemente oder zur Prüfung auf eine bestimmte Eigenschaft.
- Der Schlüssel soll die Datensätze möglichst eindeutig und kurz kennzeichnen.
- Z.B. das Web hat mehrere Billionen Seiten.
- Ordnung erleichtert das Suchen.
- Sortieren = nach einem Kriterium ordnen.
- Wichtige Ordnungen, Beispiele:
  - Zahlen ordnet man der Größe nach, etwa $1 < 5 < 19 < 45$
  - •Worte ordnet man alphabetisch, etwa Dieter < Michael < Stefan < Theo

Suchen

- Daten können alles Mögliche sein:
  - Zeichenketten, Zahlen, Bilder, …
  - Welche Telefonnummer(n) hat Bernd Schöner?
  - Welche Webseiten enthalten die Wörter „DHBW Mannheim“?
- Grundoperationen:
  - Zwei Objekte a und b vergleichen
  - Das Resultat ist
    - a < b, a ist kleiner als b, a steht vor b
    - a = b, a ist gleich b
    - a > b, a ist größer als b, a steht nach b
  - Wir messen Effizienz des Suchens in Anzahl von Vergleichen
  - Dies sagt auch tatsächliche Laufzeit gut vorher

- **Suchkategorien können sein:**
- *Elementare Suchverfahren*: Es werden nur Vergleichsoperationen zwischen Schlüsseln ausgeführt.
- *Schlüssel-Transformationen* (Hash-Verfahren): Aus dem Suchschlüssel wird mit arithmetischen Operationen direkt die Adresse von Datensätzen berechnet.
- *Suchen in Texten*: Suchen eines Musters in einer Zeichenkette.

Suche in geordneten Mengen

- Der einfachste Fall: Suchen in sortierten Folgen:
- Vereinfachende Annahmen:
  - Folge = Feld von numerischen Werten.
  - Auf jedes Element der Folge *F* kann über den Index *i* zugegriffen werden (mit *F[i]*). Erstes Element: *F[1] oder F[0]*, letztes Element bei n Werten: *F[n] oder F[n-1]*.
  - Für die Feldelemente sind die bekannten Vergleichsoperatoren =, < und > definiert.
  - Der für die Suche relevante Teil ist der numerische Wert, nach dem gesucht wird. Der gesuchte Wert ist der **Suchschlüssel**.

Sequentielle Suche

- Die Folge wird sequenziell durchlaufen, beginnend beim ersten Element.
  - In jedem Schritt wird das aktuelle Element mit dem Suchschlüssel verglichen.
  - Sobald das gesuchte Element gefunden wurde, kann die Suche beendet werden.
  - Wenn man am Ende der Folge ankommt ohne das Suchelement gefunden zu haben, wird als Ergebnis ein spezielles Element NO_KEY zurückgegeben.
- Sequenzielle Suche: Algorithmus

seqSearch (F, k)

Eingabe: Folge F der Länge n, Suchschlüssel k

Ausgabe: Position p des ersten Elements aus F, das gleich k ist, sonst NO_KEY

```
For i:= 1 to n
        If F[i] = k then return i;
        fi;
end
return NO_KEY
```

Aufwand Sequenzielle Suche

- Wichtigstes Kriterium für Beurteilung von Suchverfahren:

Aufwand.

  - Was ist der Aufwand bzw. wie wird Aufwand berechnet?
  - Notwendige Schritte mit den Vergleichen = Anzahl der Schleifendurchläufe.
  - Der Aufwand wird (sinnvoller Weise) nicht absolut betrachtet, sondern in Abhängigkeit von der Länge **n** der Folge.

	Anzahl Vergleiche
Bester Fall	1
Schlechtester Fall	n
Durchschnitt (erfolgreiche Suche)	n/2
Durchschnitt (erfolglose Suche)	n

Binäre Suche

Voraussetzung: Die Elemente in dem Feld sind geordnet.

Beschreibung des Algorithmus:

1. Wähle den mittleren Eintrag und prüfe, ob gesuchter Wert in der ersten oder in der zweiten Hälfte der Folge ist.

2. Fahre analog Schritt 1. Mit der Hälfte fort, in der sich der Eintrag befindet.

Realisierungsvarianten:

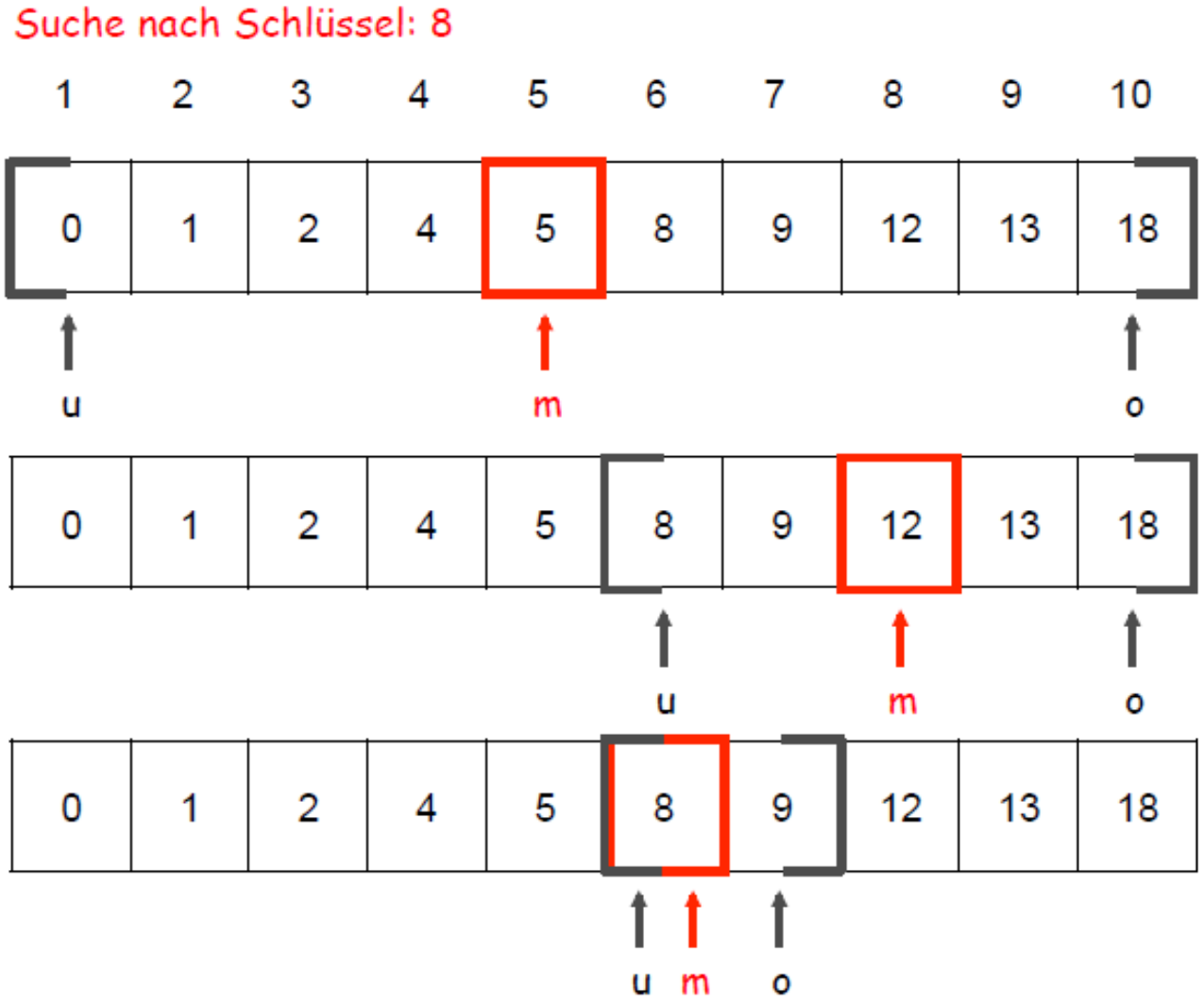
Iterativ oder rekursiv.

Binäre Suche Algorithmus

binarySearch (F, k)
Eingabe: Folge F der Länge n, Suchschlüssel k
Ausgabe: Position p des ersten Elements aus F, das gleich k ist, sonst NO_KEY

```
u := 1; o := n;
while
        u <= o
        m := (u+o)/2;
        if F[m] = k
        then return m;   // gefunden!
        else if k < F[m] then
                o := m – 1; // suche in der unteren Hälfte weiter
                else
                u := m + 1; // suche in der oberen Hälfte weiter
                fi;
        fi;
end;
return NO_KEY
```

Binäre Suche Beispiel



Aufwand Binäre Suche

- Wichtigstes Kriterium für Beurteilung von Suchverfahren:

Aufwand.

- Was ist der Aufwand bzw. wie wird Aufwand berechnet?
- Notwendige Schritte mit den Vergleichen = Anzahl der Schleifendurchläufe.
- Der Aufwand wird (sinnvoller Weise) nicht absolut betrachtet, sondern in Abhängigkeit von der Länge **n** der Folge.

	Anzahl Vergleiche
Bester Fall	1
Schlechtester Fall	$\approx \log_2 n$
Durchschnitt (erfolgreiche Suche)	$\approx \log_2 n$
Durchschnitt (erfolglose Suche)	$\approx \log_2 n$

Binäre Suche vs. Sequenzielle Suche (im Mittel)

Anz. Elemente / Verfahren	10	100 (10^2)	1.000 (10^3)	10.000 (10^4)
sequenziell (n/2)	≈ 5	≈ 50	≈ 500	≈ 5000
binär ($\log_2 n$)	≈ 3.3	≈ 6.6	≈ 9.9	≈ 13.3

Sortieren/1

- Bei kaufmännisch-administrativen Anwendungen werden über 25 Prozent der Computerzeit für Sortiervorgänge benötigt. Daher wurde intensiv nach effizienten Sortieralgorithmen gesucht.
- Unter Sortieren versteht man das Anordnen einer Menge von Objekten in einer bestimmten Ordnung, d.h. Umordnung der Datensätze, dass eine klar definierte Ordnung der Schlüssel entsteht.
- Diese Ordnung kann z. B. bei numerischen Daten durch die größer/kleiner-Relation oder bei Texten durch die lexikografische Reihenfolge gegeben sein.

- Viele Anwendungen setzen – aus unterschiedlichen Gründen - sortierte Datenmengen voraus:
  - Kontoauszüge nach Kontonummer.
  - Prüfungsnoten nach Matrikelnummer.
- Das Suchen wird schneller.

Sortieren/2

- Jeder Mensch hat wohl eine Vorstellung von Sortierstrategien, z.B. beim Kartenspielen.
- Grundlegende Sortieralgorithmen sind auch einfach zu verstehen und mit wenigen Programmzeilen implementierbar. Erst eine detailliertere Beschäftigung mit der Materie zeigt, dass die Probleme im Detail stecken.
- Es existiert eine große Anzahl von Sortieralgorithmen unter denen man den geeignetsten auswählen muss.
  - Wie ist das Zeitverhalten eines bestimmten Verfahrens?
  - Die Sortiermethoden hängen stärker als die meisten anderen Algorithmen von der Struktur der zu sortierenden Daten ab.
  - Interne vs. externe Sortierung, d.h. passen die zu sortierenden Schlüssel alle in den Arbeitsspeicher oder nicht. Hauptspeicher mit wahlfreiem Zugriff oder externe Speichermedien, die langsamer sind, jedoch größere Kapazität haben – also auf Band- oder Plattenlaufwerke.
  - Arbeitsspeicherverbrauch, d.h. wird zusätzlicher Speicherplatz – außer dem Platz für die Schlüssel – benötigt.
  - Kleine Leistungssteigerungen spielen eine große Rolle, wegen der Häufigkeit von Sortierläufen.

- Anhand dieser Kriterien ist für ein gegebenes Sortierproblem eine geeignete Auswahl zu treffen. Es gilt folgender Satz: Kein Sortier-Verfahren kommt mit weniger als $n \log_2 n$ Vergleichen zwischen Schlüsseln aus.

Sortieren durch Auswahl (Selection sort)

- Das kleinste Element wird gesucht und mit dem ersten Element der Folge vertauscht.
  Die nächste zu betrachtende Folge beginnt ein Element weiter.

- **Beispiel:**

- Die zu sortierende Folge in der nebenstehenden Grafik enthält fünf Elemente.
- Beginnend bei 420 wird das kleinste Element gesucht. Es wird an der 4. Stelle gefunden.
  Die 35 wird mit der 420 auf 1. Stelle ausgetauscht (b).
- Jetzt wird beginnend ab dem 2. Element das kleinste Element gesucht. Es steht an 3.
  Stelle (97). Die 97 wird mit der 188 vertauscht (c).
- Beginnend ab 3. wird nun das kleinste Element gesucht (188). Da es auf der 3. Stelle
  bereits steht, geschieht nichts.
- Ab 4. wird gesucht. Das Vertauschen von 301 und 420 führt zur sortierten Folge.
- Es werden nie mehr als O(n) Vertauschungen erforderlich.
- Es werden $o(n^2)$ Vergleiche benötigt, unabhängig von der Eingabeverteilung..

Element	a	b	c	d	e
1	420	35	35	35	35
2	188	188	97	97	97
3	97	97	188	188	188
4	35	420	420	420	301
5	301	301	301	301	420

Sortieren -Quicksort

- Quicksort ist ein schneller, rekursiver, nicht-stabiler Sortieralgorithmus, der nach dem Prinzip Teile und Herrsche (lateinisch Divide et impera!, englisch divide and conquer) arbeitet.
- Er wurde ca. 1960 von C. Antony R. Hoare in seiner Grundform entwickelt und seitdem verbessert.
- Der Algorithmus hat den Vorteil, dass er über eine sehr kurze innere Schleife verfügt (was die Ausführungsgeschwindigkeit stark erhöht) und ohne zusätzlichen Speicherplatz auskommt (abgesehen von dem für die Rekursion zusätzlichen benötigten Platz auf dem Aufruf-Stack).
- Im Durchschnitt führt der Quicksort-Algorithmus $O(n \cdot \log(n))$ Vergleiche durch. Im schlechtesten Fall werden $O(n2)$ Vergleiche durchgeführt, was aber in der Praxis sehr selten vorkommt.
- Heute ist Quicksort für ein breites Spektrum von praktischen Anwendungen der bevorzugte Sortieralgorithmus, weil er schnell ist und, sofern Rekursion zur Verfügung steht, einfach zu implementieren ist.
- In vielen Standardbibliotheken ist er bereits vorhanden.

Sortieren – Quicksort Grundidee

- Zerlegung der zu sortierenden Folge in 2 Teile bzw. Teilfolgen.
- Alle Elemente der einen Folge sind kleiner (linke Teilliste) als ein Referenzelement – das sog. **Pivot-Element –** und alle Elemente der anderen Folge sind größer (rechte Teilliste) als das Referenzelement.
- Das Pivot-Element kann beliebig gewählt werden (das erste, das letzte, das mittere). Optimal ist dabei das Median-Element, das zwei gleich große Teillisten erzeugt.
- Anschließend muss man also noch jede Teilliste in sich sortieren, um die Sortierung zu vollenden. Quicksort wird jeweils auf der linken und auf der rechten Teilliste ausgeführt.
- Jede Teilliste wird dann wieder in zwei Teillisten aufgeteilt und Algorithmus angewandt, und so weiter. Diese Selbstaufrufe werden als Rekursion bezeichnet.
- Wenn eine Teilliste der Länge eins oder null auftritt, so ist diese bereits sortiert und es erfolgt der Abbruch der Rekursion.
- Da sich die Reihenfolge von gleichwertigen Elementen zueinander ändern kann, ist Quicksort im Allgemeinen nicht stabil.

Beispiel Stabilität

Name	Alter
Abel, Günther	65
Krämer, Willy	52
Stein, Erwin	48
Urschel, Karin	24
Winter, Gerd	52

Sortierkriterium

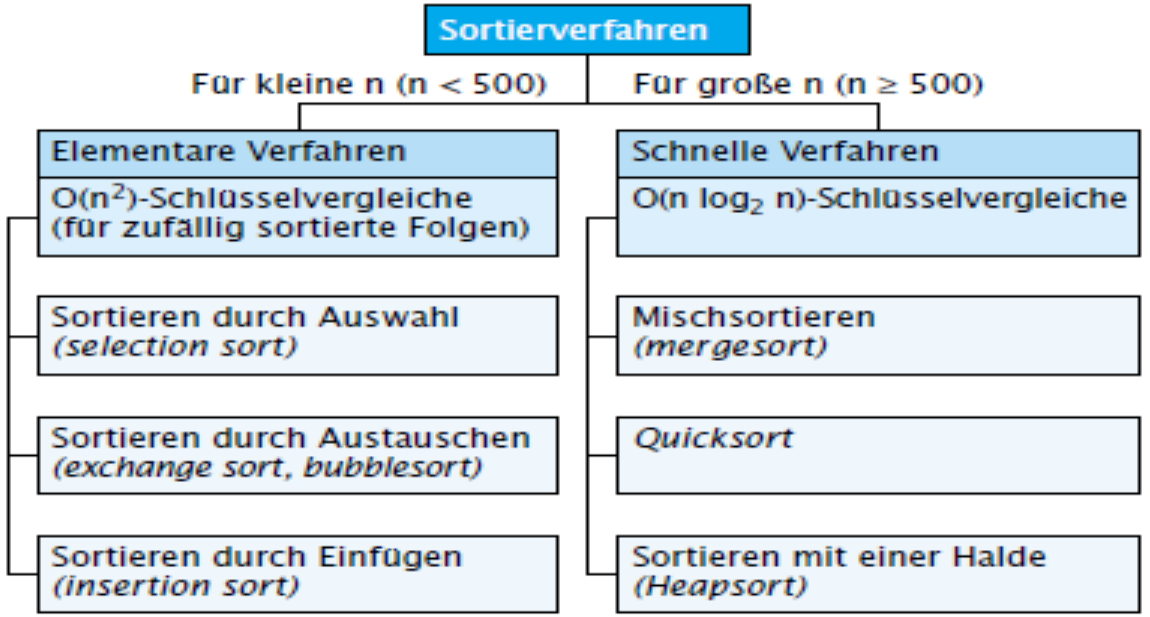
Sortierkriterium

Name	Alter
Urschel, Karin	24
Stein, Erwin	48
Krämer, Willy	52
Winter, Gerd	52
Abel, Günther	65

Die Reihenfolge bleibt unverändert.

Sortierverfahren – Klassifizierungen

- Es gibt verschiedene Klassifizierungen für Sortierverfahren, z.B.
  - Klassifiziert nach dem Arbeitsspeicherverbrauch
  - Klassifiziert nach der Stabilität
- Übersicht über verschiedene Sortierverfahren: https://de.wikipedia.org/wiki/Sortierverfahren
- Beispiel: Sortierverfahren, klassifiziert nach dem Zeitverhalten.



- Der Laufzeitaufwand ist i.w. durch die Anzahl der **Schlüsselvergleiche** und die Anzahl der **Vertauschungen** bestimmt.