
Java 1

Mosbach - Java2

VL 01

Praxis:: Die VM

Arbeit mit der VM

- Backups anlegen

- VM kopieren

- Upload in die cloud

- Email an sich selbst

- Drag&Drop ins Betriebssystem

- Upload ins Betriebssystem

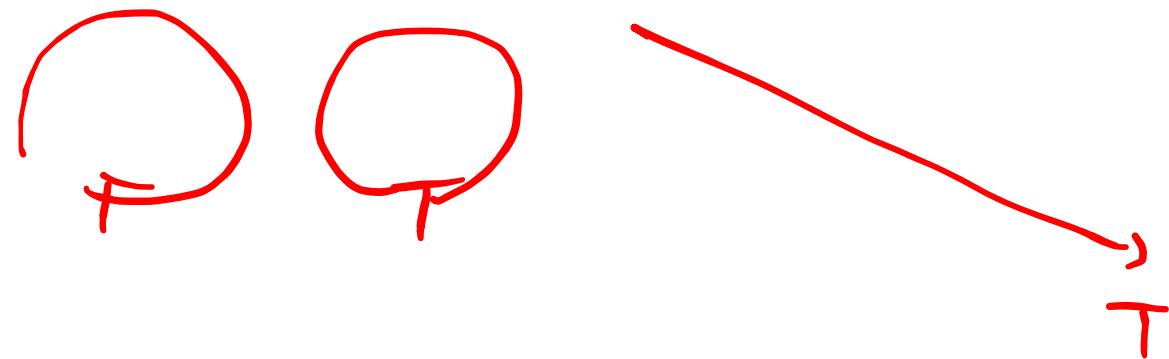
- C_DRIVE enable

- cd /transfer

- sudo mount -t vboxsf C_DRIVE transfer/

Automatisches Testen

Tests und Qualitätssicherung

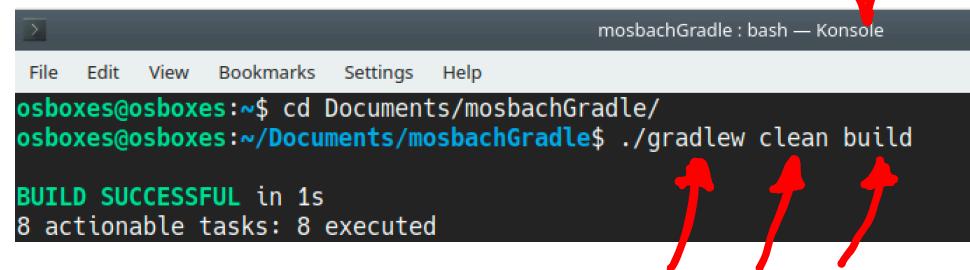


- Was ist ein Test?

✓ *Summe, Liste von Testfällen*
Testfall: Eingabe, Ausgabepage

- Wie starte ich einen Test?

~~javac~~



```
mosbachGradle : bash — Konsole
File Edit View Bookmarks Settings Help
osboxes@osboxes:~/Documents/mosbachGradle/
osboxes@osboxes:~/Documents/mosbachGradle$ ./gradlew clean build
BUILD SUCCESSFUL in 1s
8 actionable tasks: 8 executed
```

Three red arrows point from the handwritten note "javac" to the terminal window, highlighting the command being run.

- Wie teste ich?

IMMER! Ständig!

Gradle (Maven) Based Deployment

- Ziel: vollständig automatisches Verfahren zum
 - **Bauen**,
 - **Testen**,
 - **Installieren** und
 - **Starten** von Software
- engl. continuous integration and delivery
- Gradle ist eines der populärsten Systeme
- selbst in Groovy (dynamisches Java) geschrieben

A red hand-drawn underline is drawn under the word "Groovy" in the last bullet point.

Wozu? Was spricht gegen ein Bauen, Testen und Installieren von Hand?

Gradle (Maven) Based Deployment

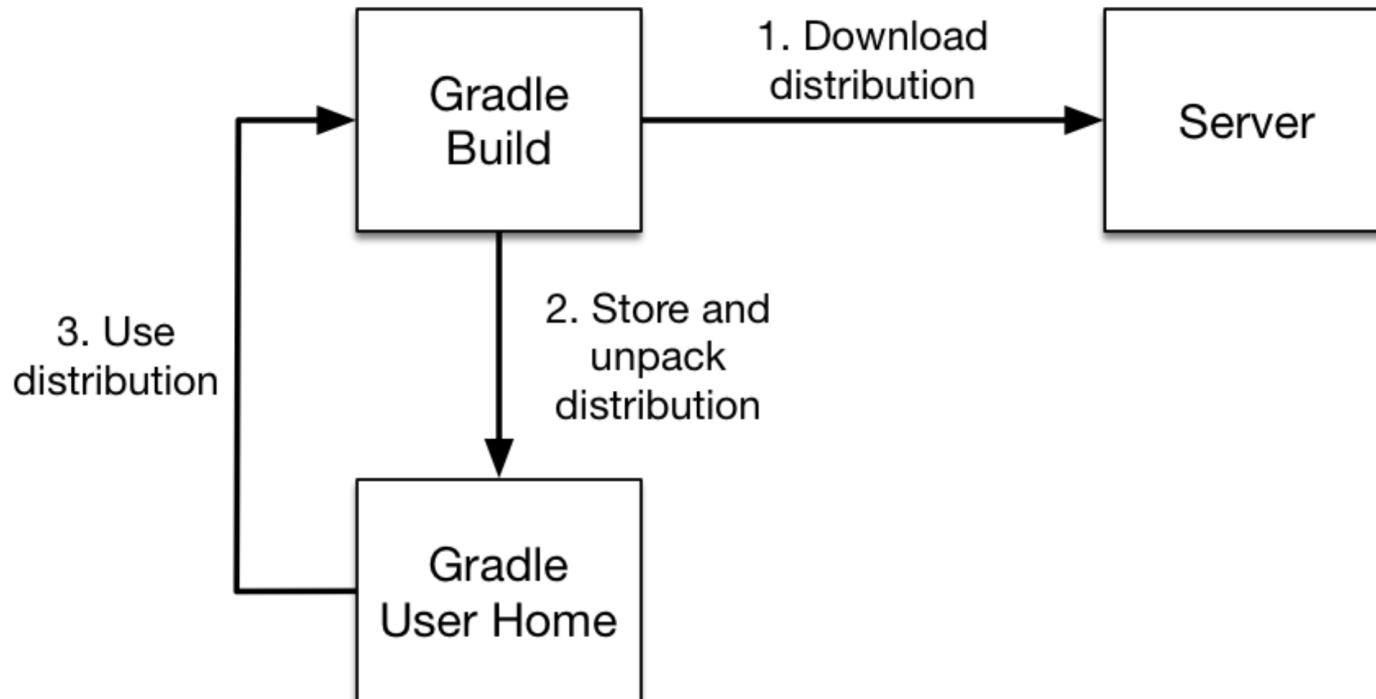


Figure 1. The Wrapper workflow

Quelle: gradle.org

Gradle Kommandos

- `./gradlew build`
 - Baut alle Java Dateien (ähnlich javac) und testet sie
- `./gradlew clean`
 - Löscht alle gebauten .class-Dateien
- `./gradlew run`
 - Startet das Java Hauptprogramm
- `./gradlew test`
 - Startet alle Test Programme

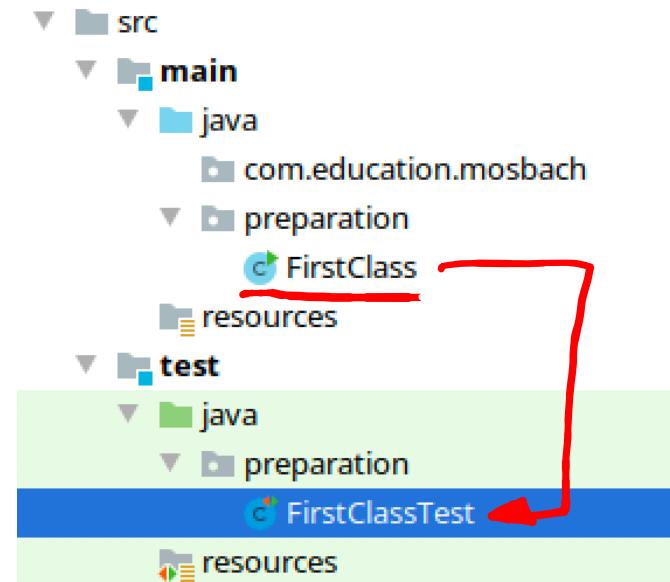


Wir testen mit

`./gradlew clean build`

Test-Regeln

- Keine Methode ohne entsprechende Test-Methode.



- Wann teste ich?
- Was teste ich?

} IMMER!

- Lege in gut benannte Pakete.
- Nutze englische Namen.
- Jede Klasse, die keine reine Datenklasse hat, bekommt eine Testklasse.
- Wähle ausreichende und interessante Testfälle.
- Alles Rumspielen heisst Dummy.

BMI Calculator

Sauber testen

Sortieren

1,7,9,4,5 ↗

1,8,9,4,5 ↗

5,4,3,2,1 ↗

[]

1,1,1,1,1,1,1



- Grenzfälle
- Pfadanalyse

Guter Testen

⇒ Finde viele extreme Werte

- Baue einen BMI Calculator.
- Diskutiere Getter/Setter.
- Diskutiere korrektes Testen.

Sortieren

- Array

fest Länge

$A = [1, 5, 4]$

$A[1]$

- ArrayList

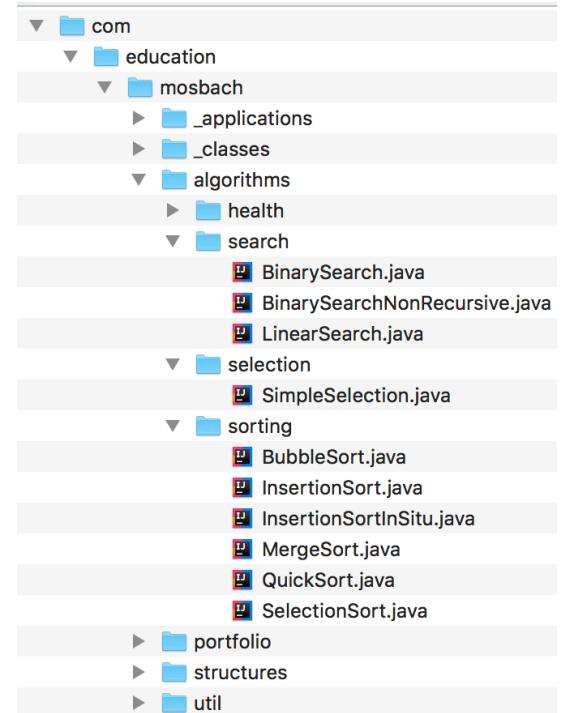
dynamische Länge

$\text{ArrayList } \underline{\text{al}} = \text{new ArrayList();}$

$\text{al.get}(1)$

Aufgaben

- BubbleSort
 - normal
 - verbessert
 - InsertionSort
 - normal
 - insitu
 - SelectionSort
 - MergeSort
 - QuickSort
- Stabilität
 - Wettkampf, Komplexität



$$A = [2, \underline{9}, 11, 7, 6, 8, 7, 5]$$

—
—

$$[2, 9, 4, 11, 6, 8, 7, 5]$$

$$[2, 9, 4, 6, \underline{11}, 8, 7, 5]$$

$$[2, 9, 4, 6, 8, \underline{11}, 7, 5]$$

$$[2, 9, 4, 6, 8, 7, \underline{11}, 5]$$

müssen bis ans
Ende

$$[2, 9, 4, 6, 8, 7, 5, 6]$$

A

1. Durchgang

[2, 9, 7, 6, 8, 7, 5, 11]

[2, 4, 9, 6, 8, 7, 5, 11]

[2, 4, 6, 9, 8, 7, 5, 11]

[2, 4, 6, 8, 9, 3, 7, 5, 11]

[2, 4, 6, 8, 7, 9, 5, 11]

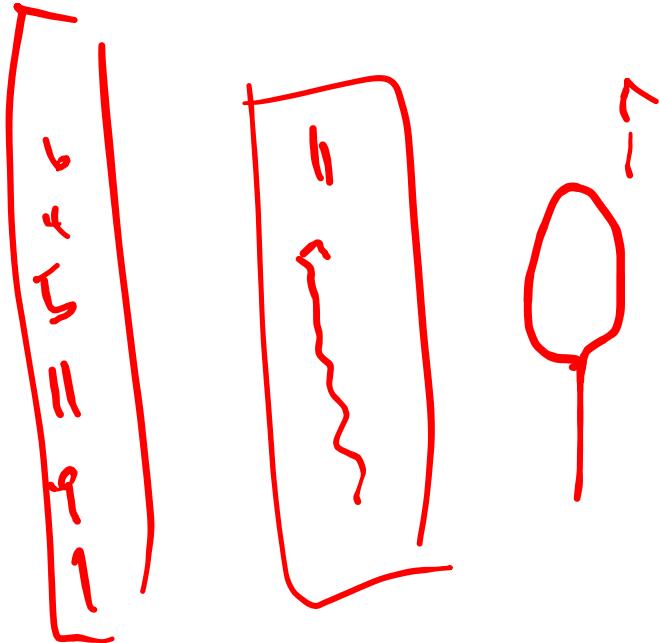
[2, 4, 6, 8, 7, 5, 9, 11]

↑↑

2. Durchgang

müssen nur
bis 1 vordende

noch weitere
Durchgänge

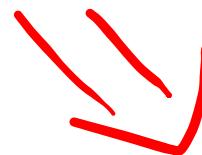


Bubbles

[1, 2, 3, 4, 5, 6]



Niemals getauscht!



Ende

Bulbladen improved:

- Pro Durchgang will bis ans Ende
- Abbruch, wenn nicht getauscht wurde

Lernziele

- Tests
- Konsistenter Zustand
- Paketierung
- Komplexität
- Sortierung: stabil
- BubbleSort
- InsertionSort
- SelectionSort
- QuickSort
- MergeSort

Wiederholung

Test (JUNIT)

↳ Liste von Ein-Ausgabekomb.

Testfall → Grenzfälle

viele

besondere Verh.

immer/ständig

alles, außer Datuklasse, Dummy El.

außer Setters/Getters

zur jeder Methode → Testmethode
gratikator clean build test

Sortieren

Bubble Sort

```
for _____ n-mal wiederholen  
  for _____ durch Array  
    : swap _____ Nachbarsort
```

Bubble Sort - Optimierungen:

- 1) nicht bis Ende
- 2) Wenn nicht getauscht, höre ab

Exceptions

Sortierung u. a.

Special Classes / Special Methods
Wrappers int - Integer, String

Collections, Interfaces, Abstract Classes

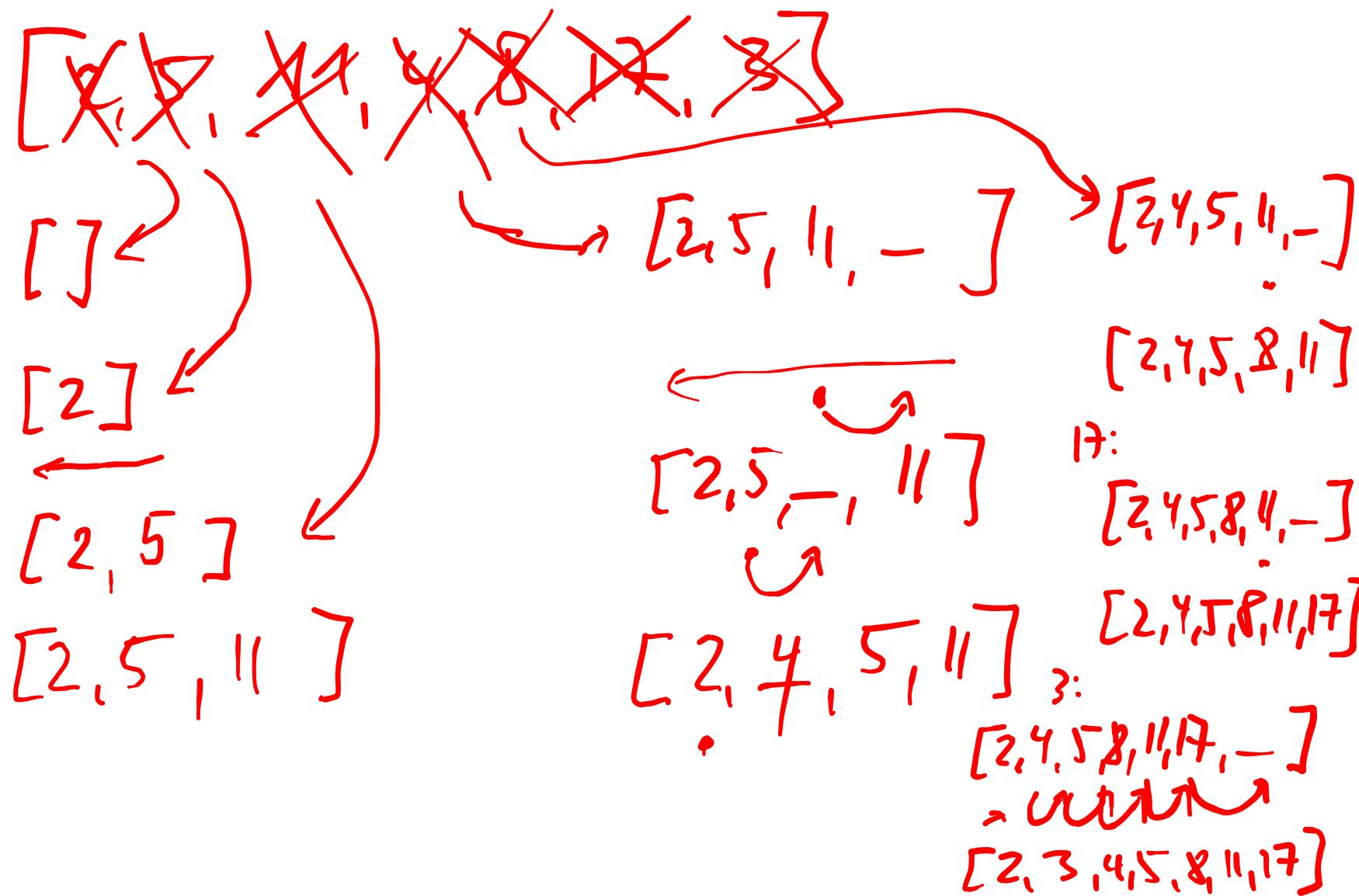
DAO, Manager

Nebenkonfiguration

Inversion

Sch

Sortieren durch Einfügen



Pre-mature optimiz.
is the root of all
evil,

- Knuth
- Dijkstra

Tag 2:

Warum Testen?

- Anforderungsprüfung
- Korrektheit
- Ldt., Spez., Ref.

Nebeneffekt: Ich lerne selbst.

Korrekturtests: - Grenzfälle finden

- Exotische Fälle



Wie? gradlew clean build test

Wann? lumen

Was? Gesamte Projekt

Aush. - Daten & Listen

- Dummy...

- ~~aus~~ Getter / Setter

Sortieren: BubbleSort

Idee ... -

Code for for
 if swap

InserionSort

Idee ...

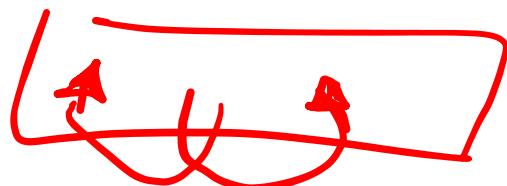


Code for/while ... remove
 while ... insert/add

Begriff: "in-situ"

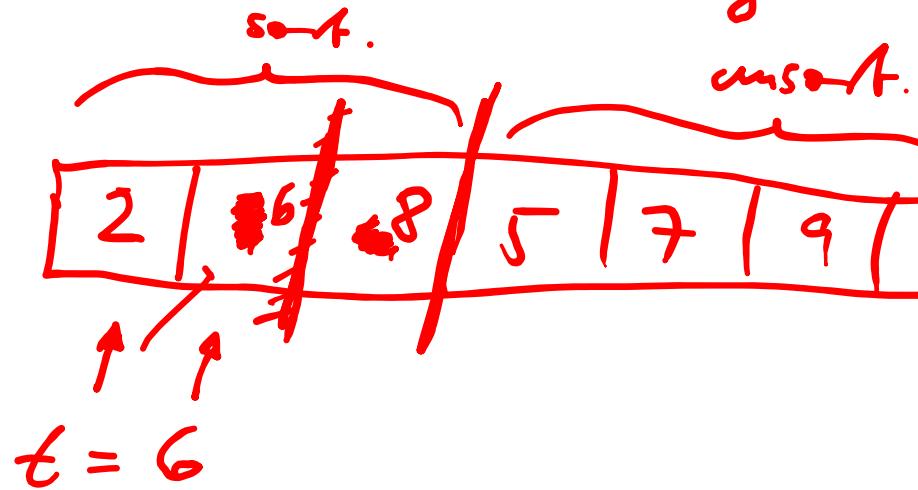
im Ausgangsplatz

im Array



wir brauchen einen extra Speicher

Auch unter Insertionsort fehlt in-situ.



Programmierung
später.

Selectionsort:

Idee: Suche n-mal das Minimum

$[6, 5, 9, 4, 3, 11, 8, 5]$

min: 6... 5..... 4... 3..... $\Rightarrow 3$



$[3, 5, 9, 4, 6, 11, 8, 5]$

min: 5.... 4..... $\Rightarrow 4$



$[3, 4, 9, 5, 6, 11, 8, 5]$

Code-Auspassung für Implementierung:

$[6, 5, 9, 4, 3, 11, 8, 5]$

minPos 0...1.....3...4..... $\Rightarrow 4$

actPos. 0
1

$[3, 5, 9, 4, 6, 11, 8, 5]$

$\Rightarrow 3$

minPos 1...2...3..... - ...

actPos 1

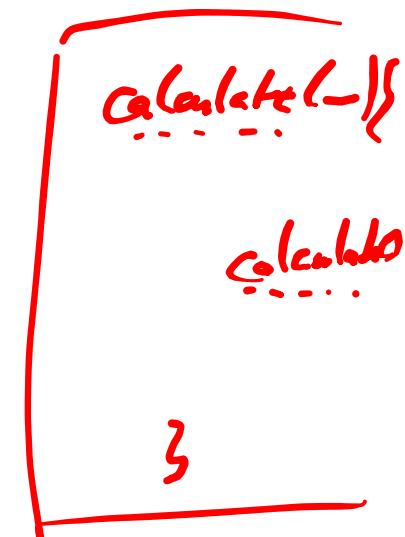
$[3, \underline{4}, 9, 5, 6, 11, 8, 5]$

Rekursive Algorithmen

Teile-Herrsche

Divide

Conquer



QuickSort

- & sortiere beim Teilen
- füge zusammen blind

MergeSort

- Teile blind

- sortiere blind zusammen - fügen

Quick Sort

$[6, 5, 12, 7, 9, 4, 1, 3, 5]$



Quicksort ($[5, 4, 3, 7, 8]$)

unsorted
size 5

pivot: 5

$[5, 4, 3, 7, 8]$

smaller: $[4, 3]$

larger: $[7, 8]$

$[3, 4, 5] \uparrow$ quicksort $[4, 3]$
- quicksort $[7, 8]$

Quicksort ($[4, 3]$)

unsorted
size 2

pivot 4

smaller $[3]$ Q:

larger $[]$

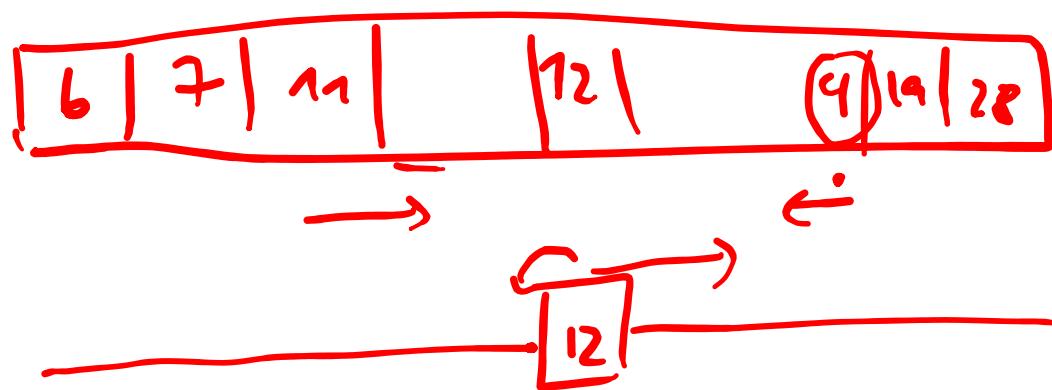
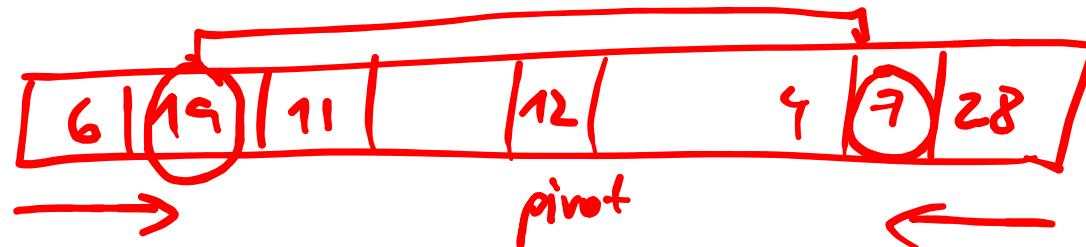
$[3] +$ quicksort $[3]$

$[7] +$ quicksort $[]$

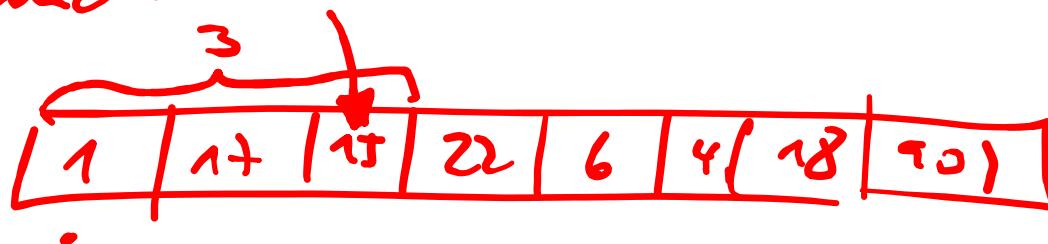
$[3], [4], []$

rethr

Quicksort in-situ Variante



Pivot schlecht

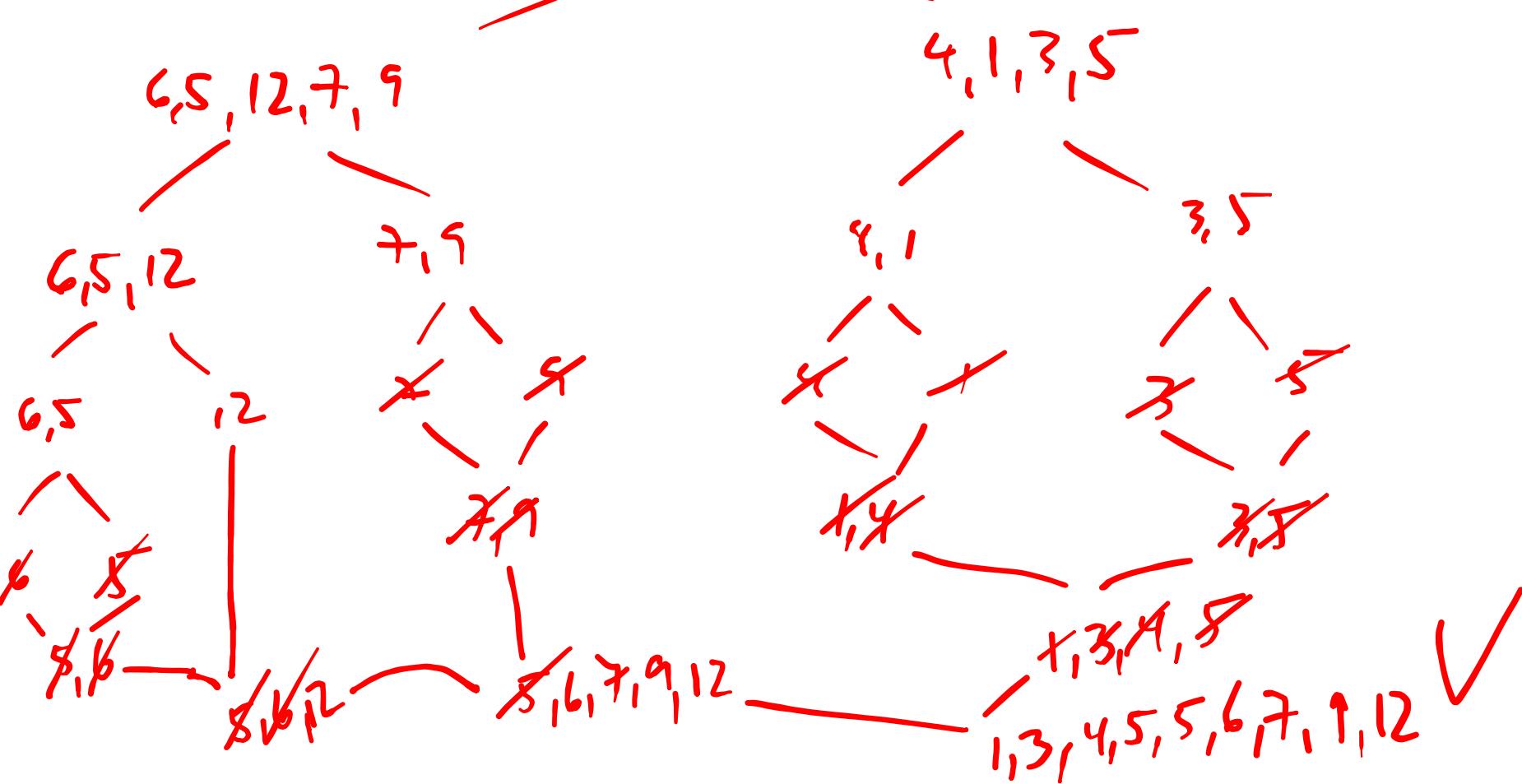


[1]



Merge sort

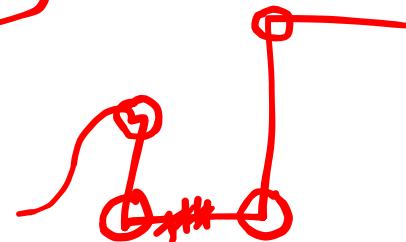
$\left[6, 5, \cancel{8}, \overset{72}{7}, 9, 4, 1, 3, 5 \right]$



Day 4

Testen: Wann? Korrektheit
 (test)
 Wie? frässler clea build
 Wann? ständig

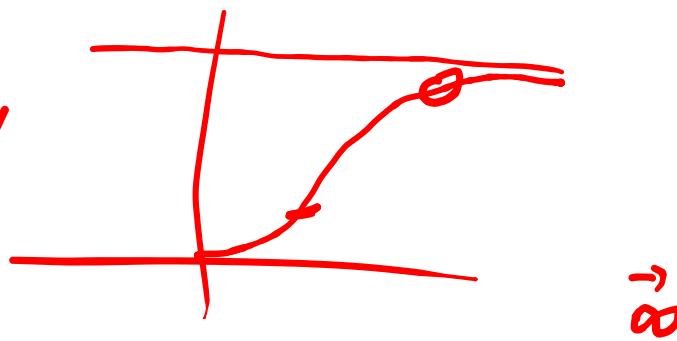
Gödel Øst.



Turing UK

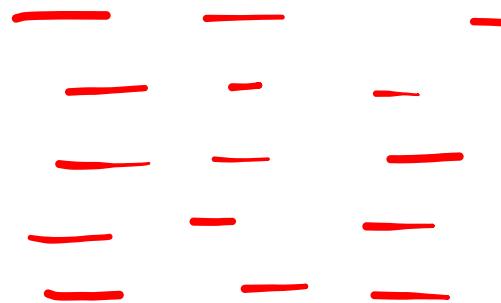
. 100% Korrekt

• $\approx 98\%$



Testmethode:

Fälle



Einlesen

Aufruf

Asset < ^{Vorläid}
Berichte

- Data Collections

- DatenContainer

feste Länge

Array

variable Länge

ArrayList

Sortieren:

Bubblesort: wird die "Liste" wird
Nachbarn verglichen

n-mal /

for

for

if-schp

Optimierungen

"in-situ" / "in-place"

Insortionsort : unsortiert langsam leeren
Sortierte langsam füllen
for/while
while >
anfügen

Selectionsort : aktuell Pos, Produkt Minim
in Rest
tauscht
for
for (i)
swap swap

Quicksort: Teile u. Herrsche

Teile: Soldat

Herrsche: Blind zusammenfügen

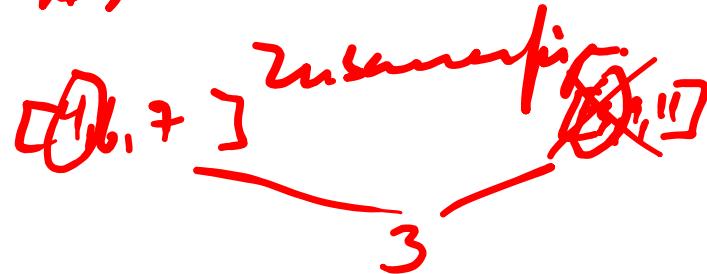


Mergesort Teile u. Herrsche

Teile: Blind

Herrsche: ~~Kopf~~ Soldaten beitreten

Vergleichen



Analyse der
Zeitkomplexität

3 Fälle

worst Case

Average Case

best Case

Bubblesort

for $i = 1 \dots n$ {

 for $j = 0 \dots n-1$ {

 if

 swap

 }

$T(n)$

$$n \cdot c_1$$

$$n \cdot n \cdot c_2$$

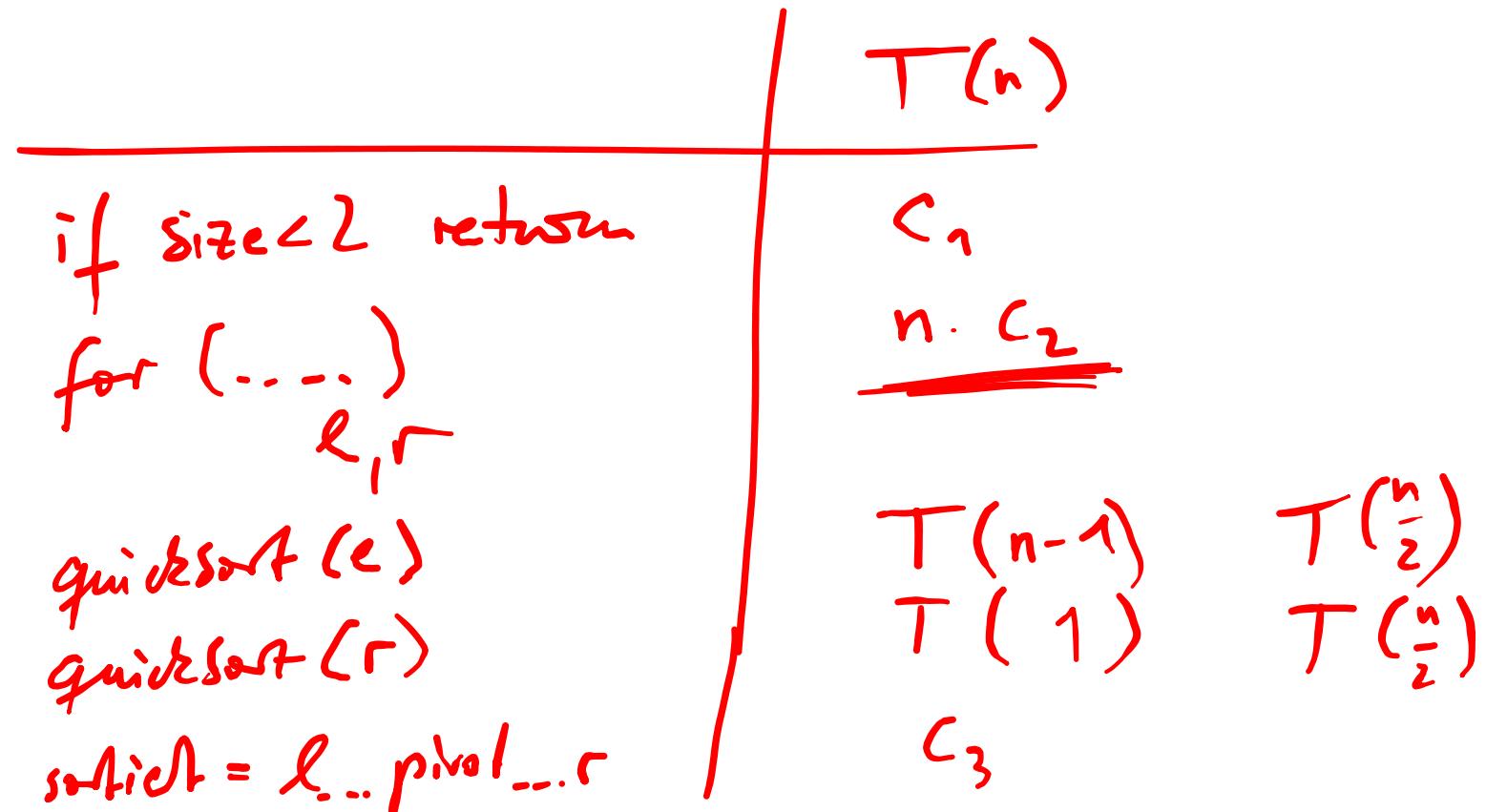
$$n \cdot n \cdot 1 \cdot c_3$$

$$n \cdot n \cdot 1 \cdot c_4$$

$\sum :$

$$\begin{aligned} T(n) &= nc_1 + n^2 \cdot c_2 + n^2 c_3 + n^2 c_4 = d_1 n^2 + d_2 n \\ &= O(n^2) \end{aligned}$$

Quick sort

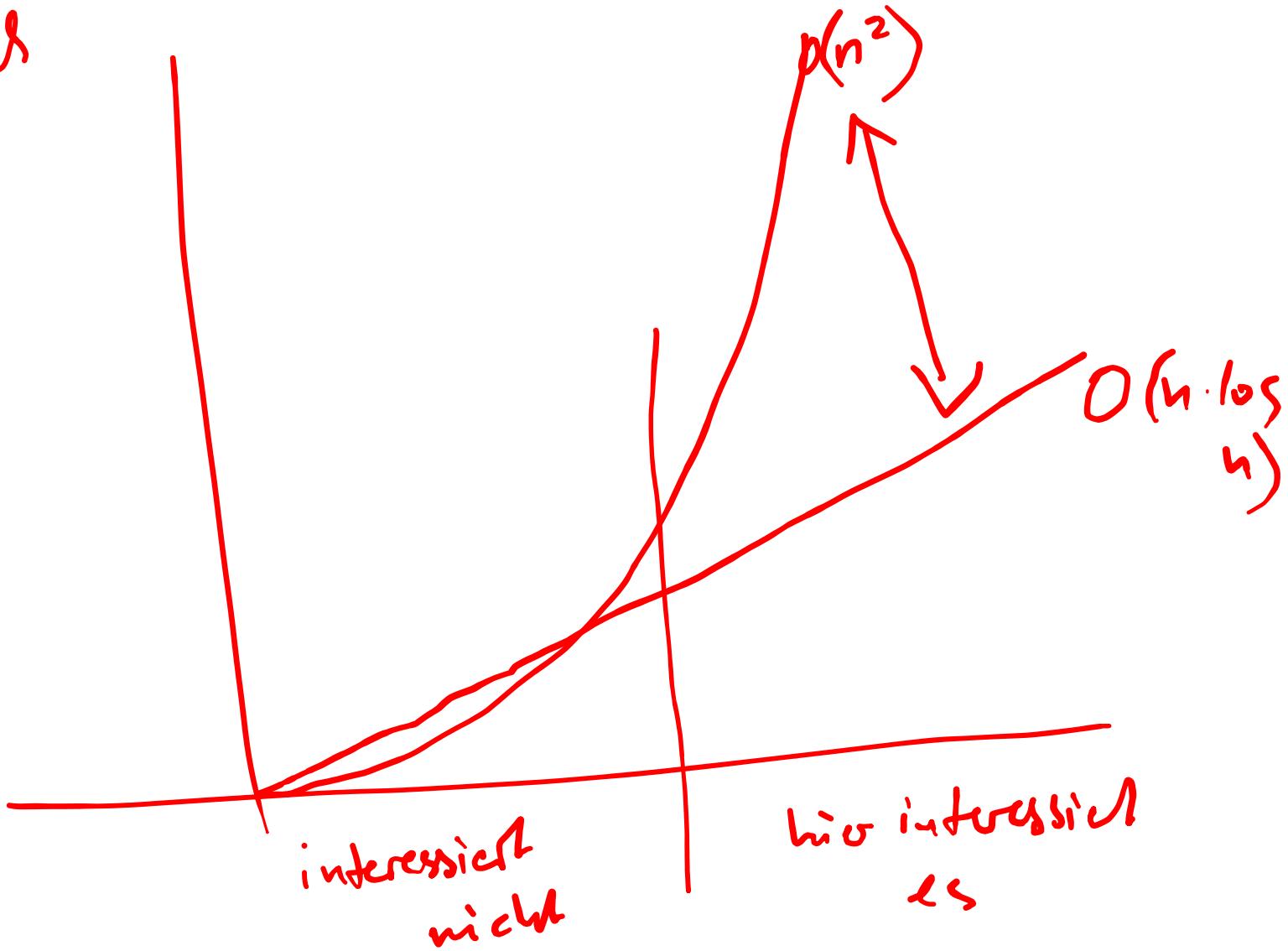


Master
Case (Best Conquerd)

$$O(n^2)$$

$$O(n \log n)$$

Vergleich



BS

$$\begin{array}{rcccl} & \times & \curvearrowright & 20\ 000 & \cancel{\text{Längen}} \\ (2 \times \beta) & \cancel{\times} & 80\ 000 & 64\ 000 & \end{array}$$

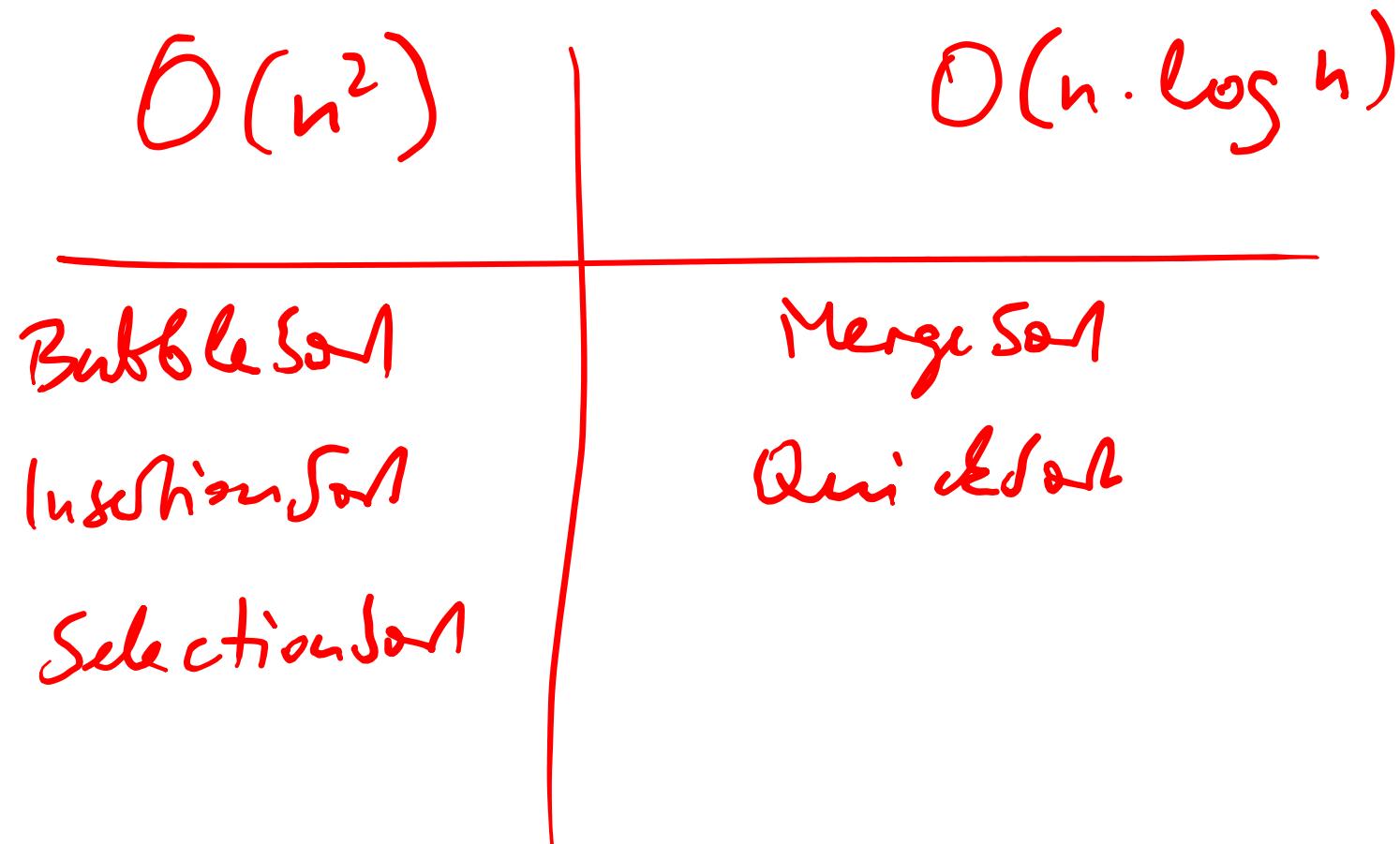
Verdopplung der Länge

\Rightarrow Verlängerung der
der Sektion

Quicksort

- $x \Rightarrow 200\text{ms}$
- $2x \Rightarrow 200\text{ms} + 30\text{ms}$
 - 1, ...

↳ $n-1$. irgendwas facht
sich



In-situ Algorithmen:

Sortieren geht nicht besser
als $n \cdot \log n$.

Nicht-in-situ-Sortierung

Bucket-Sort / Radix-Sort / Post-Sort
Sortier algo -

