

# Algorithms and Data Structures

## A15. Sorting: Overview & Outlook

Gabriele Röger

University of Basel

March 21, 2024

# Algorithms and Data Structures

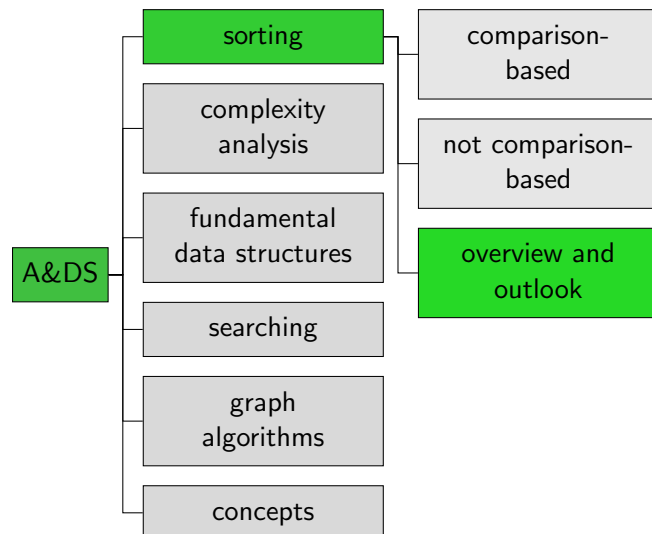
March 21, 2024 — A15. Sorting: Overview & Outlook

A15.1 Overview

A15.2 Outlook

A15.3 Quiz

## Content of the Course



A15.1 Overview

## Comparison-based Sorting: Overview

Algorithm	Running time $O(\cdot)$	Memory $O(\cdot)$	stable
	best/avg./worst	best/avg./worst	
Selection sort	$n^2$	1	no
Insertion sort	$n/n^2/n^2$	1	yes
Merge sort	$n \log n$	$n$	yes
Quicksort	$n \log n/n \log n/n^2$	$\log n/\log n/n$	no
Heapsort	$n \log n$	1	no

Very nice [visualization of the algorithms](https://www.toptal.com/developers/sorting-algorithms/) at  
<https://www.toptal.com/developers/sorting-algorithms/>

## Comparison-based Algorithms: Comments

- ▶ **Insertion sort** is **very fast on short sequences** and can be used to improve merge sort or quicksort for short ranges.
- ▶ **Quicksort** has a very short (= fast) inner loop. With randomization, the worst case almost never happens.
- ▶ **Merge sort** has the advantage of being **stable**. The merge step is also relevant for external sorting. Gets for example often used for data base applications.
- ▶ **Heapsort** is in practise slightly slower than merge sort, but interesting because it is an **in-place** approach. e.g. for embedded systems.
- ▶ Equal asymptotic running time does not mean that algorithms take equally long (different hidden constants in  $O(\cdot)$ ). Heapsort needs twice as many comparisons as merge sort.

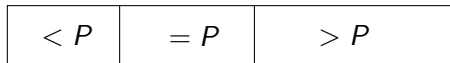
## A15.2 Outlook

## Partially Sorted Data

- ▶ Often some subsequences of the input are already sorted (so-called runs).
- ▶ Insertion sort directly benefits from this.
- ▶ For some other approaches, there are variants that exploit runs, e.g. **natural merge sort**.

## Many Equivalent Keys

- ▶ Quite common in practical applications.  
e.g. *sorting students by place of residence*
- ▶ There are special variants for some algorithms.
- ▶ For example, 3-way partitioning in quicksort



## Sorting Complex Objects

- ▶ Most of the time, we do not want to sort numbers but **complex objects**.
- ▶ It would be extremely expensive to move them in memory for every swap.
- ▶ **Instead:** Sort elements that only consist of the key and a pointer/reference to the actual object.

## Not So Correct Algorithms

### INEFFECTIVE SORTS

```
DEFINE HALFHEARTEDMERGESORT(LIST):
  IF LENGTH(LIST) < 2:
    RETURN LIST
  PIVOT = INT(LENGTH(LIST) / 2)
  A = HALFHEARTEDMERGESORT(LIST[:PIVOT])
  B = HALFHEARTEDMERGESORT(LIST[PIVOT:])
  // UMMMMMM
  RETURN [A, B] // HERE. SORRY.
```

```
DEFINE FASTBOGOSORT(LIST):
  // AN OPTIMIZED BOGOSORT
  // RUNS IN O(N*LOGN)
  FOR N FROM 1 TO LOG(LENGTH(LIST)):
    SHUFFLE(LIST):
  IF ISSORTED(LIST):
    RETURN LIST
  RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
```

```
DEFINE JOBBINTERVIEWQUICKSORT(LIST):
  OK SO YOU CHOOSE A PIVOT
  // HERE
```

```
DEFINE PANICSORT(LIST):
  IF ISSORTED(LIST):
```

full comic at <https://xkcd.com/1185/>  
(CC BY-NC 2.5)

## Solve other Problems by Sorting

### $k$ -smallest element

- ▶ For example, identifying the median ( $k = \lfloor n/2 \rfloor$ ).
- ▶ Use quicksort but only perform the recursive call for the relevant range ( $\rightarrow$  quickselect).

### Duplicates

- ▶ How many different keys are there? Which value is most common? Are there duplicate keys?
- ▶ Can be solved directly with quadratic algorithms.
- ▶ Or – more clever – sort first and then use a single scan.

# A15.3 Quiz

# Quiz



kahoot.it