Zusammenfassung

Content

[1. Algorithms 2](#_Toc40867221)

[1.1 Terminology 2](#_Toc40867222)

[1.2 How to develop an algorithm 2](#_Toc40867223)

[1.3 What is an algorithm? 2](#_Toc40867224)

[1.4 What is a data structure? 2](#_Toc40867225)

[2. Sorting 3](#_Toc40867226)

[2.1 Bubble sort 3](#_Toc40867227)

[2.2 Selection sort 4](#_Toc40867228)

[2.3 Insertion sort 4](#_Toc40867229)

[3. Recursion 6](#_Toc40867230)

[3.1 Is recursion necessary? 6](#_Toc40867231)

[4. Algorithmic complexity 6](#_Toc40867232)

[4.1 Efficiency 6](#_Toc40867233)

[5. Correctness 6](#_Toc40867234)

[6. Asymptotic complexity 7](#_Toc40867235)

[6.1 Mathematics refresher 7](#_Toc40867236)

[6.2 Order of growth 8](#_Toc40867237)

[6.2.1 Big-O notation 8](#_Toc40867238)

[6.2.2 Big-Ω notation 8](#_Toc40867239)

[6.2.3 Big-Θ notation 8](#_Toc40867240)

[7. Special case analysis 9](#_Toc40867241)

[8. Divide and conquer 10](#_Toc40867242)

[8.1 Merge sort 10](#_Toc40867243)

# 1. Algorithms

Algorithms are about solving problems. They are procedures, recipes, or process descriptions.

## 1.1 Terminology

**Data structure**: Organization of data to solve problem at hand. Solutions depend on the choice of data structure.

**Algorithm**: Finite sequence of unambiguous instructions. It’s a step-by-step outline of a computational procedure. It is independent from the choice of programming language.

**Program**: Implementation of the given computational procedure.

## 1.2 How to develop an algorithm

1. **Specification**: precisely specify and understand the problem
2. **Design**: Specify the structure/ blocks of solution. Develop algorithms with the goals of correctness and efficiency
3. **Development**: Implement the algorithm in some language
4. **Testing**: Verify that the implementation meets the specification

## 1.3 What is an algorithm?

An algorithm is:

* Any well-defined computational procedure that
  + Takes some value(s) as input
  + Produces some value(s) as output
* A tool for solving a well-specified computational problem

## 1.4 What is a data structure?

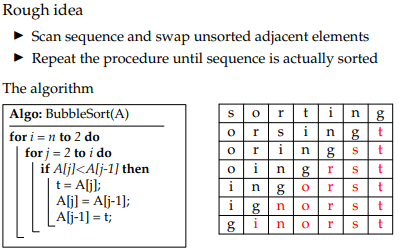
**Informal definition**: Way to store and organize data.

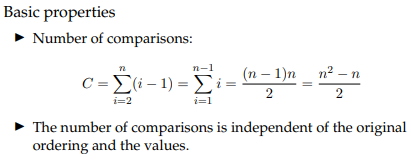
**Things to consider**: No single data structure works well for all purposes.

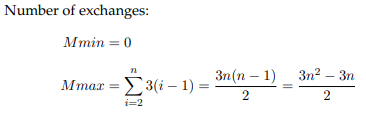
# 2. Sorting

Sorting is the most fundamental problem in the study of algorithms

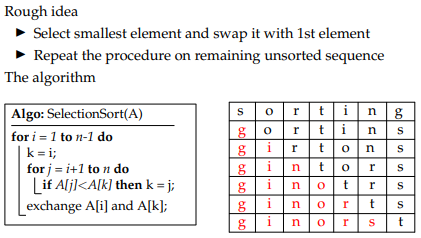
## 2.1 Bubble sort

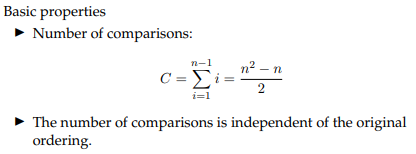


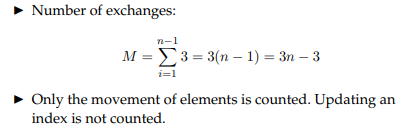




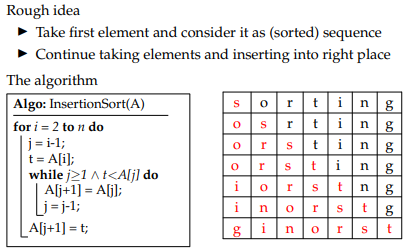
## 2.2 Selection sort

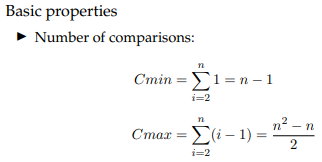


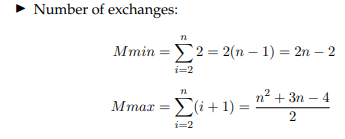




## 2.3 Insertion sort







# 3. Recursion

A recursive object contains itself as part of it. It is defined in terms of itself.

## 3.1 Is recursion necessary?

Recursion is elegant and, in some cases, the best solution by far.

# 4. Algorithmic complexity

## 4.1 Efficiency

The analysis of algorithms is predicting the resources that the algorithm requires in terms of running time and memory usage. The efficiency of an algorithm depends on the input size.

There are various ways of determining the size of input data.

Often, counting the number of iterations of the core part is sufficient. In some cases the cost of a specific operation may dominate all other costs.

# 5. Correctness

An algorithm is correct if it terminates and produces the desired output for any legal input. Automatic proof of correctness is not possible.

**Partial correctness**: if an answer is returned it will be correct

**Total correctness**: additionally requires that the algorithm terminates

**Invariants**: assertions that are valid any time they are reached

One must show three things about loop invariants:

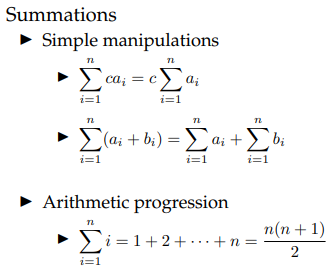
* **Initialization**: it is true before the first iteration
* **Maintenance**: if it is true before an iteration then it is true after that iteration
* **Termination**: when the loop terminates, it gives a useful property that helps to show that the algorithm is correct

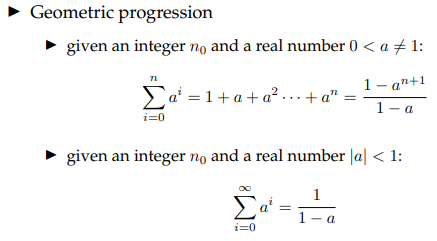
# 6. Asymptotic complexity

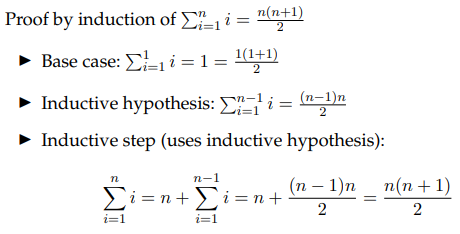
Asymptotic analysis simplifies the analysis of running time by getting rid of unnecessary details. The fundamental concern is how the running time increases with the size of the input in the limit.

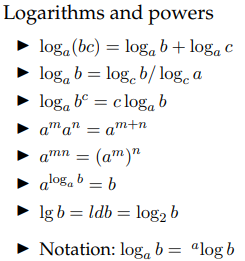
Asymptotically more efficient algorithms are best for all but small inputs.

## 6.1 Mathematics refresher



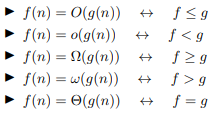








## 6.2 Order of growth



### 6.2.1 Big-O notation

Asymptotic upper bound, used for worst-case analysis.

### 6.2.2 Big-Ω notation

Asymptotic lower bound, used for best case analysis.

### 6.2.3 Big-Θ notation

Asymptotic tight bound.

# 7. Special case analysis

Basic approach:

* Checking correctness of a program
* Consider all possible extreme cases
* Make sure the solution works in these cases
* Problem is reduced to identifying all the special cases
* Related to problem specification (input/ output relation)

Special cases can be

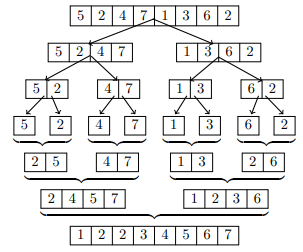
* Empty arrays,
* empty strings,
* negative numbers,
* Zero,
* entering and termination of functions,
* start and end of loop,
* first iteration of loop

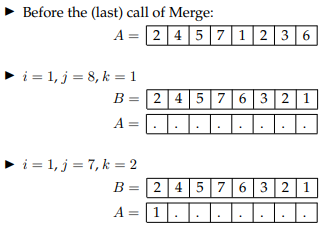
# 8. Divide and conquer

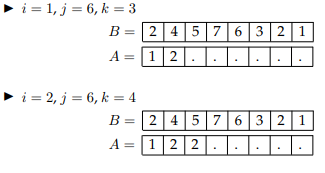
If a problem size is too big to solve trivially, then solve it with the divide and conquer- method:

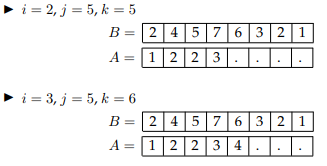
1. Divide problem into a number of subproblems
2. Conquer subproblems by solving them recursively
3. Combine solutions of the subproblem into solution for the original problem

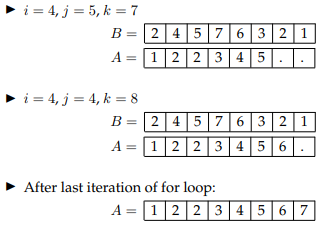
## 8.1 Merge sort











# 9. Recurrences