Algorithms and Data Structures 2020

Hans Georg Schaathun

12. mai 2020

1 Introduction

Welcome to *Algorithms and Data Structures* 2020. The module is comprised of

Lecturer Professor Hans Georg Schaathun hasc@ntnu.no. Please ask if anything is unclear.

Language English is the primary language for the module, because it also serves as an elective for the (international) MSc programme. Norwegian may be used in groups where everybody is fluent.

Main Session (four hours). The main session will start and end with a plenary lecture. The middle part will be group work.

Auxiliary Session (two hours) consists of group or individual work supervised by teaching assistants.

Compulsory assignments must be handed in, but they are not graded. However, roughly half of the exam paper will be drawn from the compulsory projects, and thus it pays to do the assignments well.

Blackboard BlackBoard is used for announcement, and if we are not allowed to have on-campus sessions, BB Collaborate will be used for joint sessions as well.

Learning material is this we site, which is dynamic and is updated as we go along.

Textbook Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser: *Data Structures and Algorithms in Java*, 6th Edition International Student Version.

In principle, the syllabus is the entire book, except Chapter 1. Supplementary texts may appear during the term, to provide more depth to key concepts.

Additional Reading It is worth reading *Introduction to Algorithms*, Third Edition by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. It is heavier than Goodrich, Tamassia, and Goldwasser in more than one sense of the word.

1.1 Learning Outcomes

A module on Algorithms and Data Structures is supposed to make you better programmers. We do that by stepping away from actual programming, and try to view computer programs from the outside. We shall learn to solve problems, describe solutions, analyse properties of the program, and prove correctness without being distracted by the details of implementation.

For most students it may be a good idea to do some programming exercises in order to link what you learn to what you already knew. That will often help you learn the new material better.

This module, however, will be assessed based on abstract discussion of programmable solutions, rather than on actual programmed solutions.

The key skills and competencies to learn are

1. being able to communicate, discuss, and evaluate potential solutions and partial designs for computer programs.
2. being able to prove that an algorithm is correct.
3. being able to assess the runtime complexity of an an algorithm.
4. being able to choose suitable algorithms and data structures for a given information system. (This will also require you to be familiar with a range of standard algorithms and data structures.)
5. being able to construct variations of algorithms.

1.2 Compulsory Assignments

Målet med refleksjonsoppgåvene er å tenkja gjennom (reflektera over) gruppearbeidet for å få best mogleg utbyte.

1.3 Previous Exam Papers

Sorry. This is a new module, so there are no previous exam papers.

2 Foundations of Algorithms

Reading 1. *Goodrich & Tamassia: Chapter (1), 2, 4.*

Algorithm Theory works with **Language-Independent Models**. We do *not* want to be tied up in the details of particular programming languages. This is important. We choose a simpler language to be understandable by a wider audience, across different language traditions (C, Java, Ada, Python).

The first thing to do in this module is to learn the basics of this language. We shall try to map the concepts to Java jargon as we go along.

2.1 Key Concepts

Consider the sorting of a bridge hand. Thirteen cards to be sorted in increasing order.

We sort first by suit, so that

$$\spadesuit > \heartsuit > \diamondsuit > \clubsuit$$

and then within each suit so that

$$A > K > Q > Kn > 10 > 9 > 8 > 7 > 6 > 5 > 4 > 3 > 2.$$

2.1.1 Step 1. Specification of the Problem

Input. A hand of 13 cards.

Output. A hand of 13 cards sorted in increasing order.

2.1.2 Step 2. Specification of the Problem

Input. An array of 13 objects.

Output. An array of 13 objects sorted in increasing order.

The objects can be of any type (class), as long as we have a binary relation $\leq$, so that for any two objects x, y we can determine whether $x \leq y$ is true or false.

In Java this may be a Boolean function $x.\mathsf{isSmallerThanOrEqual}(\mathsf{y})$.

2.1.3 Step 3. Pseudo-Code

1. Counting
2. Computing Model
3. Complexity
4. Big-O
5. Recursion and loop

2.2 Projects

1. Sorting. How do *you* sort a deck of cards? Can you describe your approach in pseudo-code?
2. Change Making (G&T P-2.5)
   - 1, 5, 10, 20, 50, 100, 200, 500, 1000
   - 1, 2, 3, 5, 10, 15, 20, 50 kopek coins; 1, 3, 5 rubles coin and note, 10, 25, 50, 100 rubles notes.
3. Letter frequencies
   - How to model the text?
   - How to parse?
   - How to model the frequency table?

3 Data Structures

Reading 2. *Goodrich & Tamassia: Chapter (2), 3*

3.1 ADT (Abstract Data Type)

1. ADT - Interfaces **NB** Definition
2. API (versus ADT)
3. Static and Dynamic Definitions
    - Java Generics
    - Type parameters

1. List
2. Array
3. Implementation versus Interface. List vs. ArrayList
4. Stack
5. Queue

3.2 Java

- List (Abstract Class)
- LinkedList and ArrayList (Concrete Classes)
- ListIterator (Java) $\sim$ List (ADT)
- LinkedList also implements the Deque ADT

3.3 Projects

1. Compare Running Time of LinkedList and ArrayList in Java
2. Data Structure for Hall of Fame (Top 100)

3.4 Projects for later

- P-5.5 Capital Gain (FIFO)
- P-6.3 Text Editor
- C-6.20 Sorting Data Packets
- Skyline.
    1. Data structure
    2. Linear or Split and Conquer?

4 The Heap

Reading 3. *Goodrich & Tamassia: Chapter 8*

4.1 Projects

- P-8.8 Job Scheduler

5 Hash tables

Reading 4. *Goodrich & Tamassia: Chapter 9*

- The Map interface

6 Sorting

Reading 5. *Goodrich & Tamassia: Chapter 11*

6.1 Projects

- Sorting - Technical Variations
- Big-O

7 Text Processing

Reading 6. *Goodrich & Tamassia: Chapter 12*

7.1 Dynamic Programming

8 Greedy Algorithms and Huffmann

Reading 7. *Goodrich & Tamassia: Chapter 12*

9 Trees

Reading 8. *Goodrich & Tamassia: Chapter 7*

10 Search Trees

Reading 9. *Goodrich & Tamassia: Chapter 10*

11 Shortest Path

Reading 10. *Goodrich & Tamassia: Chapter 13*

11.1 Algorithms

Learn (1) proofs and (2) complexity for Dijkstra's Algorithm.

Graph Algorithm Problems

1. DFT and BFT
2. Topological Sort
3. Transitive Closure
4. Shortest Path
5. Spanning Tree

11.2 Projects

11.2.1 Network Broadcast

11.2.2 Google Maps

Practical judgement rather than algorithmic

1. Calculation
2. Caching

12 Memory

Reading 11. *Goodrich & Tamassia: Chapter 14*