Week 11
Revision

1. Introduction
2. Algorithms and their analysis
3. Graphs
4. Data structures
5. Conclusion

General remarks

You need to be able to perform the following:

- Reproducing all definitions.
- Performing all computations on (arbitrary) examples.
- Explaining all algorithms in (written) words (possibly using some mathematical formalism).
- Stating the time complexities in term of Θ.

Check all this by writing it all down!

Actually *understanding*
the definitions and algorithms helps a lot
(but sometimes "understanding" can become a trap —
in the sense that you need to get used to partial knowledge).

How to prepare

All information is on

Canvas

- Go through all lectures (perhaps you download the slides again — certain additions have been made).
- Go through the courseworks and their solutions.
- Go through the lab sessions and their solutions.
- And there are additional videos (all videos and recording available in the weekly "modules").

On the exam

You need to answer all questions.

- Give us a chance to give you marks — write down something!
- If in doubt about something, explain your doubts (in a few sentences), perhaps best in the second or third round, after having answered all questions.
- Examples are typically helpful; you can always write them down as part of your answer.

The general structure of the module

1. Weeks 1-4: Algorithms and their analysis
2. Weeks 5-7: Graphs — BFS and DFS.
3. Week 8: Hashing.
4. Week 9: Sorting finalised.

Sorting algorithms

We considered four sorting algorithms in detail:

Insertion-Sort, Merge-Sort, Heap-Sort and Quick-Sort.

These algorithms you should know.

- Know their run-times (with the special case of Quick-Sort).
- Know how to derive/explain these run-times.
- Explain them, and know their differences and similarities.

Order of growth

Get a working understanding of
O, Ω, Θ
(that is, know how to use them, and how to work with them).

This includes developing an **intuitive grasp** on the main types of growth rates:

$$\underbrace{1, \sin(n),}_{\texttt{constant}} \quad \underbrace{\log n, \lg(n),}_{\texttt{logarithmic}} \quad \underbrace{\sqrt{n}, \overbrace{n}^{\texttt{linear}}, n \log n, n^2, n^3,}_{\texttt{polynomial}} \quad \underbrace{2^n, n!}_{\texttt{exponential}}$$

- Know examples for the main complexities $1, \log n, n, n \log n, n^2$.
- You need also to know the outline of the definitions.
- And you need to know the various techniques for working with terms/expressions/formulas ("rewrite rules").
- For some given term like $t = 6n^2 + 88n$, work out its Theta-form: $t = \Theta(n^2)$.

Solving recurrences

Know the (simplified) master theorem!

- Especially know the three cases.
- And know various examples.

It's mostly practice.

General remark:
In the exam, there are no subtle questions.
If in doubt, and you have several approaches,
you might outline both of them.

At least state your assumptions explicitly.

Concepts

For graphs, various definitions and their relations are of central importance:

1. Know the definitions of “graphs” and “digraphs”.
2. Know the definition of a “dag”.
3. Know the definition of “(dis)connected graphs”, and of “connected components” of a graph.
4. Know the definition of a “tree” and “forest”.
5. Know what a “rooted tree” is:
    1. Many important notions are connected to it (“root, children, parent, leaf”).
    2. Know what “ordered rooted trees” are.
    3. Know what “binary trees” are.

Basic graph algorithms

You need to know BFS, DFS well.

This means:

1. Be able to explain them (including pseudo-code).
2. Be able to run them (on paper).
3. How are the trees resp. forests represented?
4. Know what $d[u]$ for BFS is.
5. Know what $d[u], f[u]$ for DFS is.

A basic application of BFS and DFS is the detection of **cycles** (understand this, for graphs and digraphs).

Our main application of DFS is **topological sorting**; and again, cycles (or their absence) are a main topic here, now with emphasise on the directed case (more complicated!).

Hash tables

Essential are the **ideas**:

- It's all about hash functions.
- Know what a **collision** is.
- Direct addressing is an important special case.
- Collisions can be solved rather efficiently by the *chaining method*.
- Know the two basic techniques for computing **hash values**:

  1. Transform objects into natural numbers via some *radix system*.
  2. Make natural numbers small (so that they can be used as indices of the hash table) via the *division method*.

Binary heaps

Again, essential are the **ideas**:

- **Binary heaps** are "complete" binary trees with the "heap property". Don't mix them up with binary *search* trees!
- We have an efficient array-based implementation.
- "Heapification" is a key procedure.
  1. With it we can build a binary heap efficiently.
  2. **Heap-Sort** is another application.
  3. And we get an efficient implementation of priority queues.

Run Heap-Sort on examples!

Quick-Sort

Again, understand the **ideas**:

- Pseudo-code for the main procedure should be easy.
- A different form of partitioning (different from Merge-Sort).
- Know the key points of the analysis ("good on average", "bad on worst-case").

Further remarks

- You should be able to develop simple divide-and-conquer algorithms, analysing them via recurrences, and determine an expression $T(n) = \Theta(?)$ for their run-time via the (always simplified) Master Theorem.
- And you should be able to compare such algorithms with the direct approaches which exist always for simple tasks (that's why they are "simple").