A&D Learning Goals

DISCLAIMER: This document serves as a supplementary study aid and is not an official course publication. It was prepared by Josia Heger (Teaching Assistant) and reflects his personal recommendations regarding study strategies and important concepts. There is no guarantee that the exam only covers topics listed here.

Mathematical Foundations

☐ Write **induction proofs** using the structure: base case, induction hypothesis, and induction step. You should also be able to prove more difficult statements, e.g. inequalities or the number of leaves on a specific tree. Understand when you need more than one base case and don't make off-by-one errors for the induction hypothesis.

☐ Know the **Gaussian Sum Formula**: $\sum_{k=1}^{n} k = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$

Maybe this is also useful: $\sum_{k=1}^{n} k^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$

Maybe this is also useful: $\sum_{k=1}^{n} k^3 = 1^3 + 2^3 + 3^3 + \cdots + n^3 = \left[\frac{n(n+1)}{2}\right]^2$

Asymptotic Growth

☐ Know the definition that f **grows asymptotically faster** that g if $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$.

☐ Given two functions, **compute the limit** and decide if one grows asymptotically faster than the other.

☐ Know the following **derivation rules**:

$(x^n)' = n \cdot x^{n-1}$

$(f + g)' = f' + g'$

$(e^x)' = e^x$

$(\ln(x))' = \frac{1}{x}$

$(f \cdot g)' = f'g + fg'$

$\left(\frac{f}{g}\right)' = \frac{f'g - fg'}{g^2}$

$(f(g(x)))' = f'(g(x)) \cdot g'(x)$ (least important)

☐ Know the following **power rules:**

$a^m \cdot a^n = a^{m+n}$

$\frac{a^m}{a^n} = a^{m-n}$

$(a^m)^n = a^{m \cdot n}$

$(a \cdot b)^n = a^n \cdot b^n$

$\left(\frac{a}{b}\right)^n = \frac{a^n}{b^n}$

☐ Know the following **log rules:**

$\log_a(x) = \frac{\log_b(x)}{\log_b(a)}$

$\log_a(x^r) = r \cdot \log_a(x)$

$\log_a(x \cdot y) = \log_a(x) + \log_a(y)$

☐ Know **L'Hopital** and when you can apply it

☐ Use **substitution** e.g. $y = \log(n)$ or $y = 1/n$ to simplify a limit

☐ Know the $e^{\ln}$ or $2^{\log_2}$ trick

☐ Know the **definitions of** O, Θ, Ω and how to use limits to show that e.g. $f \leq O(g)$

☐ Know the **sandwich theorem**

☐ Know how to lower/upper bound sums, e.g. $\sum_{i=1}^{n} i^3$

☐ Know that $1 \leq \log n \leq \sqrt{n} \leq n \leq n \log n \leq n^2 \leq 2^n \leq n! \leq n^n$

☐ Know when you can ignore constants (e.g. for $\log_c n$ or $\log n^c$) and when not (e.g. for c^n)

☐ **Count function calls** for a given pseudo code snippets as an exact sum (don't make off-by-one errors) and in O notation.

☐ Given a function in the form $T(n) \leq c + T(n/2)$ you should be able to **telescope** it to find an explicit formula for $T(n)$.

☐ Given the **master theorem** and a function in the form $T(n) \leq aT(n/2) + c \cdot n^b$ you should be able to decide what case of the master theorem it is and find explicit formula for $T(n)$.

Searching and Sorting

☐ Know what linear search does and the lower bound for searching

☐ Know the pseudocode of **binary search** and know its runtime. You should be able to program it in java.

For the following algorithms, you should be able to **simulate the algorithm step by step** on a given array on paper. You should know the **runtime** and how many **swaps/comparisons** it needs in **best/worst case** in O or Θ notation. You should also know the **invariants** (except for quick sort and merge sort) and what algorithms are in-place and if they have good locality.

☐ **Bubble Sort**

☐ **Selection Sort**

☐ **Insertion Sort**

☐ **Merge Sort** (If you want to learn to program a sorting algorithm, choose merge sort. However programming a sorting algorithm has never been asked at any exam... So I would recommend studying other things first)

☐ **Quick Sort**

☐ **Heap Sort**

☐ Know the precise definition of a **heap**, how to perform **ExtractMax** and how to insert values.

☐ Know how to prove an iterative algorithm with an **invariant**, i.e. show $INV(0)$ as base case, $INV(j) \implies INV(j+1)$ as induction step and $INV(n)$ as termination. You don't have to do this for recursive algorithms such as Merge Sort and Quick Sort.

☐ Know the **lower bound** for comparison based sorting, i.e. $\Omega(n \log n)$ and the proof with the decision tree.

☐ Know that Bucket Sort exists and when it works

Data Structures

☐ Know the runtimes for inserting/deleting/... in an **array/linked list/double linked list**.
Be aware that the runtime of an algorithm depends on what data structure we use, e.g. binary search or heap sort have a worse runtime with linked lists.

☐ Know the precise definition of a **binary search tree** and how to perform search/insert.

☐ Know the precise definition of **2-3-Trees** and how to perform search/insert/delete. Don't use any other sources from the internet for 2-3-Trees, because the definitions are slightly different in most cases.

Dynamic Programming

For the following problems, you should know the problem description and possible sub-problems. Based on this you should be able to remember the base case, the recursion, the solution and the runtime. It's far more important that you understand the algorithms than that you know every detail by heart. However the DP problems on the exam will probably be similar to them, so it definitely helps if you understand them well. You should be able to explain them both on paper and program them in ~30 minutes java.

☐ **Fibonacci**

☐ **Maximum Subarray Sum**

☐ **Jump Game**

☐ **Longest Common Subsequence**

☐ **Edit Distance**

☐ **Subset Sum**

☐ **Knapsack** (Be aware that there is an approximation algorithm, but you don't have to describe it in detail.)

☐ **Longest Ascending Subsequence**

☐ Know the difference between **bottom-up (iterative) and top-down (recursive)**. You should be comfortable implementing both versions. Understand why we use memoization.

☐ Know basic strategies for finding a subproblem, e.g. only look at the first i elements

☐ Describe backtracking in words for a given DP algorithm.

☐ Explain what pseudo polynomial means and in two sentences what the P vs NP problem is.

Graph Theory

☐ Know the precise **definitions** of graph, node, edge, walk, path, closed walk, cycle, length, degree, directed/undirected, weighted, (closed) Eulerian walk, Hamilton path/cycle, connected component, tree, leaf, bipartite, DAG, adjacent, incident, in-degree, out-degree, sink, source

☐ Know how to prove statements about graphs. Use proof strategies from Discrete Mathematics such as $A \to B \equiv \neg B \to \neg A$, proof by contradiction, disproving with a counterexample...

☐ Represent graphs as **adjacency matrix/list** and understand the different runtimes

☐ **Handshake lemma** for both directed/undirected graphs

☐ Know when there exists a (closed) **Eulerian walk**

☐ Understand the complexity difference between finding a closed Eulerian walk vs. a Hamilton cycle.

☐ Know what a **Topological Sorting** is and when there exists one. Know that the reversed post order is a topological sorting if there are no directed cycles.

☐ Given a graph on paper, perform DFS step by step and find **pre/post numbers and classify edges** and draw the **search tree**. Understand what it means if there is a back edge. Know the runtime.

☐ Given a graph on paper, perform BFS step by step and find **enter/leave numbers** and the **distances** to s and draw the **search tree**. Understand the concept of level sets. Know the runtime.

☐ **Program DFS and BFS**. This should be a very easy task for you and shouldn't take more than ~5 minutes. Practice this until you are very comfortable with it.

☐ Know what we mean by shortest paths in weighted graphs, and why we need the assumption that there are no negative cycles.

☐ Given a graph on paper, perform **Dijkstra** step by step and find the shortest path tree and the distances to s. Know that Dijkstra assumes that there are no negative edge weights. Know the runtime. You should understand the pseudocode, but you don't have to program it. (This has never been asked in any exam)

☐ Know the pseudocode of **Bellman Ford** and the invariant "l-genaue Schranken" and improve bounds procedure. Know how to detect negative cycles with Bellman Ford. Know the runtime. Be aware that the order in which we iterate over the edges in one improve bounds iteration is not determined.

☐ Know the precise definitions of, spanning edges, spanning tree, **minimum spanning tree**

☐ Know that the unique minimum weight edge crossing a cut is a **critical edge** (=sichere Kante).

☐ Given a graph on paper, perform **Boruvka** step by step and draw the MST. Know the runtime and that it only works with unique edge weights. You never want to program it.

☐ Given a graph on paper, perform **Prim** step by step and draw the MST. Know the runtime. You never want to program it.

☐ Given a graph on paper, perform **Kruskal** step by step and draw the MST. Know the runtime. Ideally you know how to **program Kruskal** using the **union find** data structure. You can use a library function to sort the edges by weight. It's not crucial, but if you want a 5+ you should know how to do it in ~25 minutes.

☐ Know the subproblem, base case, recursion, solution and runtime of **Floyd-Warshall**. Again being able to program it is not crucial, but if you want a 5.5+ you should know how to do it in ~20 minutes.

☐ Understand the general idea of **Johnson** with the height function and describe how it works in a few sentences. You don't have to program it.

☐ Know some **creative tricks** for graph exercises, e.g. Graph layering, adding more nodes, creating a super node, flipping edges, inverting the edge weights, sorting edges, applying some function to the edge weights... Check out my notes of week 12 for some concrete examples.