ROTTERDAM UNIVERSITY OF APPLIED SCIENCES / CMI

Course Manual
Data Structure and Algorithms

INFDSA 01

ECTS: 12
Course facilitator: Saleem Anwar, Nezhinsky, A.E. (Alexander)

Description of the course

Course name:	Data Structure and Algorithms
Course code:	INFDSA 01
Study points and workload:	12 ECTS This course provides the student with 12 ECTS, which corresponds to a study load of 336 hours. The distribution of these hours over the teaching weeks is as follows: **Contact time** Lesson: 16 weeks Practicum: 16 weeks Combined 8 hours per week 128 hours **Self-study** Approximately 10 hours a week ~9.9 hours 198 hours **Assessment:** Exam and practical assessment 10 hours **Total**: **336 hours**
Learning goals:	After successfully completing this course, the student is able to: 1. **Understand** fundamental data structures (e.g., arrays, lists, stacks, queues, trees, hash tables, matrices) and their properties. 2. **Analyse** properties of fundamental data structures (e.g., arrays, lists, stacks, queues, trees, hash tables, matrices). 3. **Understand** core algorithmic techniques (e.g., searching, sorting, etc). 4. **Analyze** properties (time and memory) of core algorithmic techniques. 5. **Apply** appropriate data structures to design and produce solutions for a given context (e.g. software development, AI/ML). 6. **Apply** algorithmic thinking to solve computational problems for a given context (e.g. software development, AI/ML). 7. **Evaluate** correctness and efficiency of data structures and algorithms in a programming language for a given context (e.g. software development, AI/ML). All concepts and keywords learnt during the previous course (*Semester 1 & 2*) are assumed to be available and will be built upon.
Course facilitator:	Anwar, S. (Saleem); Nezhinsky, A.E. (Alexander)
Date:	27-1-2026

1 General Information

This course introduces students to fundamental **data structures** and **algorithms**, emphasizing how they influence software efficiency, scalability, and maintainability. Students will implement core data structures and algorithms in a specified programming language, analyse performance, and apply them to practical problems.
The course blends theory, coding labs, and case-based learning to simulate **real-world development challenges**, emphasizing clean architecture, abstraction, and algorithmic efficiency.

The objective of the course is to develop a programming solutions with efficiency (time and memory) in mind. A program by definition involves processing data, and to store that data efficiently, one must use appropriate data structures along with efficient algorithms. A "good" program is defined as one that minimizes memory usage and completes tasks in the least amount of time. Although execution time may vary based on the machine's processing power and available resources, an understanding of performance analysis—particularly complexity analysis—is essential.

2 Program

This course is offered in two tracks, both of which cover the fundamentals of data structures and algorithms. One track focuses on applying these concepts in a **general software development** context, while the other explores their use within a **machine learning** context.

The course includes weekly in-person sessions (see the lesson schedule on HINT). These sessions may consist of lectures, practicums, and practical case work. Students are expected to complete training assignments to practice and reinforce the concepts introduced in class. Practical work is designed to be collaborative and will take place both during scheduled class hours and at appropriate times outside of class.

The course topics are outlined below and may be adjusted as the course progresses. Any changes will be communicated to students in a timely manner. Detailed lesson materials are available in the course GitHub repository.

2.1 Topics (Track DSA in Software Development)

- Introduction to Complexity Analysis and Arrays recap
- Recursion recap, Higher Order Functions
- Searching
- Sorting
- Linked Lists
- Stack and Queue
- Hash Table
- Binary Search Tree
- Graphs and Graph Algorithms
- Dynamic Programming
- Miscellanies topics in Data Structures and Algorithms.

For a more detailed description of subjects see:
hogeschool/InfDSA: Data structures and algorithms

2.2 Topics (Track DSA in Foundations of Machine Learning)

- Basic Datastructures, complexity and recap
- Graphs and Trees
- Working with numpy and basic data exploration
- Data types and representation

- Unsupervised learning
- Supervised Learning
- Neural Networks Introduction
- Architecture and data for the NN
- Activation functions and forward propagation
- Back propagation
- Train and predict
- Decision Trees
- Extra topics on machine learning

For a more detailed description of subjects see:
https://github.com/hogeschool/Machine-Learning-Basics-public/blob/main/theory.md

2.3 Practical Case

The description of the case for **DSA in Software Development** Track can be found on:
hogeschool/InfDSA: Data structures and algorithms

The description of the case for **Machine Learning** Track can be found on:
https://github.com/hogeschool/Machine-Learning-Basics-public/blob/main/README.md

3 Attendance

Attendance is compulsory for all lectures and practicums, and attendance is recorded at every session. Students are required to attend **at least 80% of all scheduled sessions**.

Being late (arriving more than 10 minutes after the scheduled start time), counts as an absence. The course coordinator, manager or dean will determine whether your absence qualifies as an 'exceptional circumstance' (such as a prolonged illness or the funeral of an immediate family member).
Exceptions to the attendance requirement can only be made in consultation with the instructor(s). Work, a scheduled doctor's or dentist's appointment, a few days of flu, or problems with public transport, etc., do not qualify as exceptional circumstances. Public transport strikes are an exception.

If you do not meet the attendance requirement, you will receive a No Go/ND. You will not be able to retake the course. This means you will have to retake the course next academic year.

General Software Development Track:
This track runs for 16 weeks with 3 sessions per week, totaling 48 sessions (128 hours). Based on the 80% attendance requirement, students must attend **at least 38 of the 48 sessions**. A session may be either a theory lecture or a practicum.

Machine Learning Track:

This track runs for 16 weeks with 2 sessions per week, totaling 32 sessions (128 hours). Based on the 80% attendance requirement, students must attend **at least 25 of the 32 sessions**. A session may be either a theory lecture or a practicum.

4 Deliverables

A software solution that matches requitement and description for the given case in each track.

5 Evaluation

The assessment comprises two components: a theory exam and a case_based assessment. The final grade is the average of these two scores, provided that a minimum grade of 5.5 is achieved in each.

The Learning objective 1 to 4 will be assessed during exam, whereas the learning objectives 5 to 7 will be assessed during the case-based assessment.

5.1 Theory exam (50% of your final grade)

The theory exam (in ANS) consists of multiple-choice questions to assess the learning objectives 1 to 4. There will be 30 multiple exam questions roughly 25% of the questions from each learning objective. The retake will follow the same form as the first chance exam. The grading scheme is included in this document.

5.1.1 Theory exam grading scheme

Correct Answers	Grade	Remarks
<=20	0	Fail
21	5,5	Pass
22	6	Pass
23	6,5	Pass
24	7	Pass
25	7,5	Pass
26	8	Pass
27	8,5	Pass
28	9	Pass
29	9,5	Pass
30	10	Pass

5.2 Case Assessment (50% of your final grade)

This course includes a practical case to implement. The practical case is a team-based assignment, and you are expected to actively participate as a constructive and reliable team member. Together with your team, you will design and implement an application that meets the requirements outlined in the case description document.

Dedicated time to work on the case will be provided during practicum sessions, where guidance and feedback will be available. However, most of the work will need to be completed independently outside of class time (see the study points and workload information in the module description above).

Throughout the course, you will receive formative feedback on your progress. This feedback is intended to help you improve and should be used to strengthen your work toward the final assessment. Instructions regarding submission and/or presentation will be shared during the course.

While the case is completed as a group, grading is individual and based on your personal contribution and your understanding of the concepts applied in the codebase. During assessments, you will be asked to explain your own contributions.

Everyone is expected to make a meaningful contribution to the practical case. Contributions will be monitored throughout the course, and clarification may be requested when needed. The solution will be assessed using the rubric included in this document. Detailed interpretation of the rubric will be communicated within each track.

5.3 Grading Criteria / Cesuur

- A student will receive a **Go** for the exam if they have attended **the number of sessions as described Attendance**. Failure to meet this attendance requirement will result in a **No Go**, meaning the student will not be allowed to participate in any form of assessment, and there will be **no opportunity for repair**.
- To successfully pass the course, a student must achieve a **sufficient grade (≥ 5.5)** in **both** the exam and the practical case components.
- The **final grade** will be the **average of the exam grade and the practical case grade**, but **only if both components are sufficient (≥ 5.5)**.
- If either the exam or the practical case grade is insufficient (< 5.5), that grade will **not contribute** to the final grade.
- Students are allowed to improve **only the insufficient component(s)** during the second attempt.
- **Partial grades are valid only within the current academic semester.** If the course is not successfully completed within the same semester, all assessment activities (exam and practical case) must be repeated in the following academic year/semester.

5.4 Assessment Schedule

The exam will be scheduled in **week 18 (first attempt)** and **week 20 (second attempt)**. The exact **date, time, and location** will be published officially. The **exam review (inzage)** will take place during the same exam week.

The **practical assessments**, both formative and summative, will be scheduled by the class teacher and communicated at the **start of the course**.

5.5 Rubric

The case will be assessed based on the following global rubric for course learning objectives. A detailed interpretation of the criteria will be communicated within each track.

Criterion (CLO)	Excellent (8.5–10)	Sufficient (5.5–8.4)	Not Sufficient (<5.5)
5. Apply appropriate data structures	Data structures are **fully correct**, well-designed, and matched to the problem (e.g., optimal choice). Implementation is clean, complete, and demonstrates deep understanding.	**Most of the expected data structures** are implemented correctly and used appropriately. Minor inefficiencies or small logical issues are acceptable.	Data structures missing, incorrect, misapplied, or implemented with major errors. The solution does not demonstrate sufficient understanding.
6. Apply algorithmic thinking	**All** required algorithms (and any additional chosen ones) are correctly implemented using appropriate algorithmic design principles. Demonstrates the ability to generalize or adapt algorithms to new contexts.	**Most of the expected algorithms** are correctly implemented and follow learned patterns. Solution works for standard cases but is not fully generalized.	Algorithms missing, incorrect, or poorly structured. No evidence of algorithmic reasoning.

7. Evaluate correctness and efficiency	Prototype works correctly across **varied scenarios**, with systematic testing. Student evaluates both correctness AND performance trade-offs (e.g., time/space complexity). Clear evidence of debugging, validation, and analysis.	Prototype works for **most scenarios** and is tested on common use cases. Basic reasoning about correctness is present, limited or no performance analysis.	Prototype fails standard scenarios. Little or no testing. No evaluation of correctness or efficiency.