DSA: Rational – Why do this?

Donald F. Ross

21D 413

donald.ross@kau.se

Expected Workload (200 hours)

Component	Time
Assessment - Exam 40p	66%
Assessment - Labs 20p	33%
16 Lectures à 2 hours	32
16 Labs à 2 hours	32
Contact time	64
Self-study Coursework	104
Self-study Lab	32
Total Time	200
Lab groups 1-2 students	

Labs	total
Seq Ex	12
1. Tree	17
2. Performance	18
3. Graph	17
Time	64

Daily Plan: am + pm			
lecture	lab	lab or study	lab or study
2	2	2	2

Goals

- **Hard**
  - **Data structures**
    - Set
    - Sequence
    - Tree
    - Graph
  - **Algorithms**
  - Modelling
  - **Implementations**
  - **Improve C programming**
- **Soft**
  - **Abstraction**
  - **Generalisation**
  - **Recursion**
  - Mental “toolkit”
  - Change mindset
  - **Articulate Ideas**
- **Terminology**
  - 50% of u-grad course
  - important

Data Structures & Algorithms

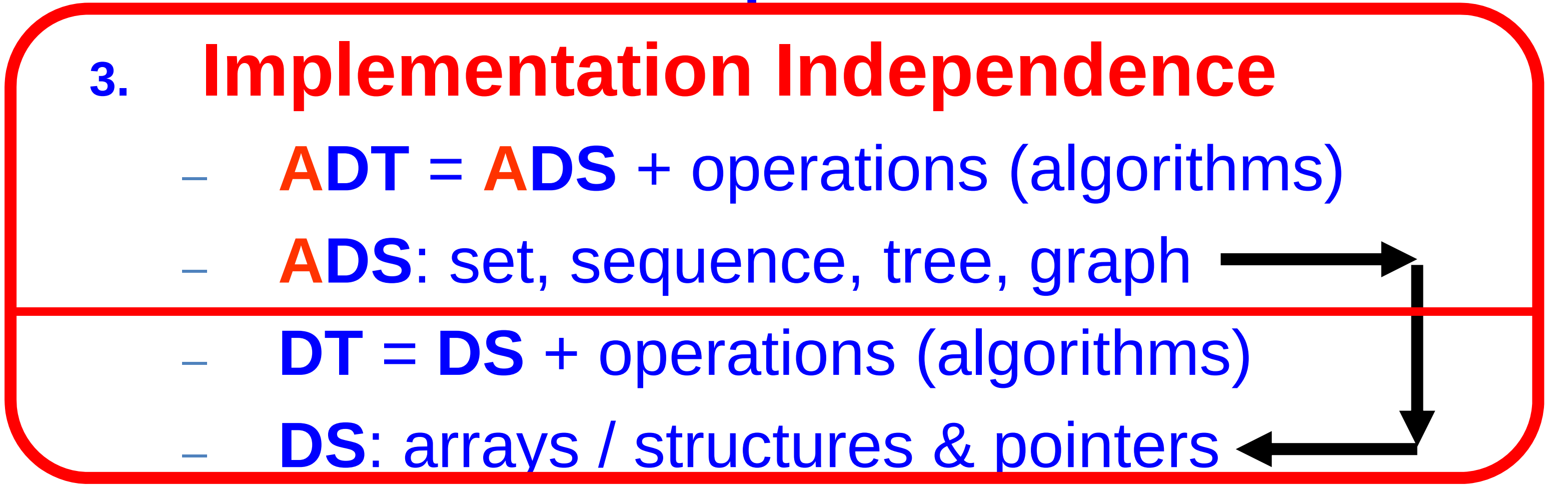
- **Data**: **info** about real world entities
- **Structures**: ways of **organising data**
- **Algorithms**: **operations** on data structures
  - Sorting & **searching**
  - Navigating through the data structure
  - Manipulating collections

Course content

Content	Details
Data Structures	Set, sequence, tree, graph (collections)
Operations	Add, find, remove, size, is_empty, display
ADT	**ALGORITHMS**
Sequence	Sorting & searching, hashing, heap
Trees: BST, AVL	In-, pre-, post-order, depth/breadth-first **search**
Graphs	Dijkstra, SPT, Floyd, Warshall, Prim, Kruskal, topological sort, TSP
ABSTRACTION	Collection, modelling, ***implementation***
RECURSION	Definitions (sequence & tree) + code
Analysis	Big-Oh and performance analysis

ABSTRACTION: modelling, collection & implementation

1. Reality to a **model**
   - **Entities & relationships + attributes**
2. Data Structures (set, sequence, tree, graph)
   - **Collections + operations**



3. **Implementation Independence**
   - **ADT** = **ADS** + operations (algorithms)
   - **ADS**: set, sequence, tree, graph
   - **DT** = **DS** + operations (algorithms)
   - **DS**: arrays / structures & pointers

Collections (set, sequence, tree, graph)

- **2 levels**
  - Collection
  - Entities (**members** of the collection)
- **Operations**
  - Collection: **create, destroy, display, sort, navigate, count, is_empty, merge, compare,**
  - Entities: **add, remove, find, display**

ADT: set (non-linear; unordered)

- **Properties:** a collection of **unique** entities
- **Relationship:** none
- **Operations:**
  - As for collections
  - Mathematical set operations
- **Implementations**:
  - Structures + pointers (linked lists) / arrays
  - **NB: the implementation is a sequence hence can use recursion!**
- **Used for:** Relational Databases

ADT: sequence (linear; ordered)

- **Properties:** a collection of ordered entities
- **Relationships:** successor (E_n, E_{n+1}) (next)
  predecessor (E_{n-1}, E_n) (previous)
- **Operations**
  - As for collections
  - Sorting & Searching
- **Implementations**: struct+ptrs (linked lists) / arrays

  NB: the implementation is a sequence hence can use recursion!
- **Used for**: hashing, heaps, implementing graphs

ADT: tree (non-linear; (un)ordered)

- **Properties:** a collection of hierarchical entities
- **Relationships:** parent/child
- **Kinds:**
  - general, binary, binary search, AVL, B-trees
- **Operations**
  - As for collections
  - **Searching** depth/breadth first
- **Implementations:** struct+ptrs / arrays
  - **NB: the implementation uses recursion!**
- **Used for:** DB indexes, hierarchies

ADT: graph – G =(V, E) (non-linear; unordered)

- **Properties:** a collection of entities
- **Relationships:** directed (N_x, N_y)
  undirected $(N_x, N_y), (N_y, N_x)$
- **Operations**
  - As for collections
  - Searching depth / breadth first
  - **A 2 B** problem shortest distance
- **Implementations:** struct+ptrs (linked lists) / arrays
- **Used for:** computer networks
- **Algorithms: Dijkstra (SPT), Floyd, TSP, Warshall, Prim, Kruskal**

Summary

real world

model

implementation

- Course Goals – Learn about
  1. **abstraction** ➔ model ➔ implementation
  2. **abstraction** ➔ **ADT**s as collections
  3. **abstraction** ➔ implementation independence
  4. **ADT**s – set, sequence, tree, graphs + ops
  5. **Algorithms** + some implementations
  6. Labs – **application** of the above

Sequence (**linear**; **ordered**) ; **Set** & **Graph** G=(V,E) (**non-linear**; **unordered**)
Tree (**non-linear**; **unordered (GT)** / **ordered (GT, BT, BST, AVL)**)