DSA Topics

- Collections
- Abstraction
- Abstract Data Type (ADTs) = ADS + operations
- Recursion ➔ definitions & functions
- Performance ➔ O(1), O(log n), O(n), O(n log n), $O(n^2)$, $O(n^3)$
- Algorithms
  - Sequence ➔ sorting, searching, hashing, heap
  - Tree ➔ general ➔ BT ➔ BST ➔ AVL; DFS, BFS
  - Graph ➔ Dijkstra / Floyd / Warshall / Prim / Kruskal
    ➔ topological sort, Travelling Salesman
  - Greedy ➔ Dijkstra, Prim – what does greedy mean here?
- **TERMINOLOGY – know by heart**

Collections

- General collection operations
  - add, remove, find, count, is_empty, join 2 collections

- **Set** + set operations (from set theory)
- **Sequence** + sequence operations
  - Restrictions on a sequence ➔ **stack & queue**
- **Tree** + tree operations + navigation (DFS, BFS)
- **Graph** + graph operations

Definitions

- Collections = entities (elements) + relationships
  - SET
    - unique elements, unordered, no relationships
  - SEQUENCE
    - elements, ordered, successor relationship, elements may be sorted
  - TREE
    - elements, ordered/unordered, parent/child relationship
  - GRAPH is (V,E)
    - V set of elements, E set of edges (general relationships)

Applications

- Set
  - Collections with no relationships, Relational Data Bases
- Sequence
  - Text (text handling, editors, program text)
  - Execution of a program – sequence of instructions
- Tree – hierarchical systems
  - Computer file directories, taxonomies, family trees
  - Arithmetic operations, parse trees (in compilers)
- Graph
  - Network systems
    - Computer, telephone, transport, disease vectors
    - State diagrams, Flow problems

Abstraction

- Modelling abstraction
  - Real world ➔ computer model
- Implementation abstraction
  - Hiding the actual implementation
    - E.g. arrays + indexes / pointers + structures
    - Collection of elements and references to an element
- Collection abstraction
  - Set/sequence/tree/graph ➔ collection with common operations

ADTs + operations

- An ADT is implementation independent
  - Implemented as an entity + attributes + relationships
  - The actual data structure is hidden using get/set functions for each attribute + create_entity to give a reference to an entity
  - Other operations are implemented as implementation independent (abstract) functions (methods)

- In this way an abstract sequence can be used as a
  - Stack – add / remove restricted to position 1
  - Queue - add restricted to position last / remove to position 1

Recursion

- Recursive definitions – know these by heart
  - Sequence **S** ::= H **T** | empty; H ::= element; **T ::= S;**
  - Binary tree **BT** ::= **LC** N **RC** | empty; N ::= element
    LC ::= BT; RC ::= BT;

- Recursive functions for sequences and trees
  - These follow from the recursive definitions

- Pattern for recursive code
  - The **stop condition** - usually is_empty (the empty case)
  - The **non-recursive operation** (at head or on node for BT)
  - The **recursive call** – often in a **cons function**

Performance

- Know what Big-oh means
- Know Big-oh for common operations: add, remove, find
- Know Big-Oh for common algorithms
  - Sort & search
  - Dijkstra, Floyd, Warshall, Prim, Kruskal

- How to measure and interpret performance

Algorithms

- Sequence
  - Sort algorithms
    - swap → bubble, insert, selection, Shell
    - divide and conquer → quicksort, merge sort
  - Search algorithms → linear, binary
  - Hashing & collision detection
  - Heap
- Tree
  - Navigation → depth-first (pre-/in-/post-order), breadth-first
  - General tree to BT, AVL & tree balancing & rotation, heap
- Graph – directed / undirected
  - Depth-first search & spanning forest
  - DAGs & topological sort (+ alternative → in-degree = 0)
  - Dijkstra's (+SPT), Floyd, Warshall, Prim, Kruskal, TSP heuristic

Terminology

- Sequences
  - First, next, last, head, tail, (total) order, successor, predecessor
- Trees
  - Parent/child, root, node, leaf, full, perfect, complete
- Graphs
  - Degree (in/out), node, edge, path, simple path, cycle, simple cycle
  - SPT (Shortest Path Tree – extension to Dijkstra)
  - DAG (Directed Acyclic Graph), partial order
  - Free tree, spanning tree/forest, MST (minimal spanning tree)
  - Strong components, connectivity, reachability
- Algorithms
  - Computable, non-computable, tractable, intractable, polynomial

Motivation for the course

- To introduce a greater degree of abstraction into your thinking and programming – **a mental toolbox**
- To introduce a programming style (abstract) independent of the programming language
- To present the **ADTs set, sequence, tree and graph** together with their implementations, use and some common algorithms – these are found throughout computer science
- To introduce performance and Big-oh
- To improve your reading and understanding of the principles behind algorithms

Exam hints

- **Before**
  - Prepare – read the lecture notes and other material
  - Work through the given examples
  - Know the **principles** behind the algorithms
  - Check previous exams and facits
- **During**
  - Plan your time
  - Read all questions
  - Do the easiest questions first
  - Note key words: **briefly** or **in detail** or **stepwise**
  - The **process** as well as the answer is important