Programs: Algorithms + Data Structures

- **Algorithm:** a finite set of instructions that, if followed, accomplishes a general, well-specified task.
  - **It has:** Input/Output
  - **It should be:** Clear, finite and correct
  - **Good algorithms** are efficient and easy to implement
  - **Example:** Euclid algorithm for GCD calculation
- **Classifying algorithms by:**
  - **Problem domain:** numeric, text processing, sorting, searching, networks, machine learning, etc.
  - **Design Strategy:** divide and conquer, greedy, dynamic programming, backtracking, etc.
  - **Complexity:** constant, linear, quadratic, cubic, exponential, etc.
  - **Implementation Dimensions:** sequential, parallel, recursive, iterative, etc.
- **Data type:** Set of values and operations on those values.
  - **Primitive data types:** integer, float, Boolean, character, string*, pointer*, etc.
  - **Complex data types (class):** employee, list, stack, etc.
- **Data Structures:** Arrangement of data for being able to store and retrieve information.
  - **Choosing the right one depends on:**
    - Is it filled completely at the beginning? Are there insertions/deletions/lookups/updates?
    - Will the items be processed in order? Do we need random access?
  - **Example:** List, BST, Hash Table, etc.
- **Efficient software minimizes** coding/debugging/running/system integration time and memory.
- **Pseudocode:** A high-level description of an algorithm
  - It is **more structured** than English and **less detailed** than a program.
- **Algorithm analysis:** analyzing how the resource requirements of an algorithm will scale when increasing the input size.
  - We asymptotically analyze the worst case time complexity T(n) of algorithms using RAM model.
  - We do **theoretical analysis** because experimental analysis takes much time for coding and testing and depends on the hardware/software.
  - **Theoretical analysis**: characterizing running time as a function of the input size $T(n)$
    - It can be done on a pseudocode independently from the hardware/software environment
  - To measure T(n) we need to know **execution time** and **frequency count** for every instruction
    - T(n) [approximately] = **Execution time * Frequency count**
  - Execution time is measures using **RAM model**:
    - Each simple operation (+, *, -, =, if, call) takes exactly one-step.
    - Loops and subroutines are not considered simple operations.
    - Each memory access takes exactly one time stamp.
  - Example: Insertion sort algorithm T(n) = a.n^2 + b.n + c (where a,b,c are constants)
- **Big Oh Analysis:** O(n) = T(n) ignoring constant factors and lower-order terms
  - f(n) = O(g(n)) means c . g(n) is an upper bound on f(n).
    - $\exists$ c, $n_0 > 0$ | f(n) < c.g(n) $\forall n > n_0$
  - f(n) = Ω(g(n)) means c.g(n) is a lower bound on f(n).
    - $\exists$ c, $n_0 > 0$ | f(n) > c.g(n) $\forall n > n_0$
  - f(n) = Θ(g(n)) means c_1.g(n) is an upper bound on f(n) and c_2.g(n) is a lower bound on f(n)
    - $\exists$ c_1, c_2, $n_0 > 0$ | c_1.g(n) < f(n) < c_2.g(n) $\forall n > n_0$
    - This means that g(n) provides a nice, tight bound on f(n).
- **Space Complexity:** Determining how much space an algorithm requires by analyzing its storage requirements as a function of the input size.

Elementary data structure:

- **ADT vs DS**
- **List**
  - **Properties:**
  - **Methods:**
  - **ArrayList**
  - **LinkedList**
- **Stack**
  - **Properties:**
  - **Methods:**
  - **Array Implementation**
  - **Linked-List Implementation**
- **Queue**
  - **Properties:**
  - **Methods:**
  - **Array Implementation**
  - **Linked-List Implementation**
- **Map/Dictionary ADT**
  - **Properties:**
  - **Methods:**
  - **Implementations**
    - **TreeMap**
    - **HashMap**
      - **Hash function:**
        - **Ideal hash function**
      - **Hash code:**
        - **Memory address**
        - **Integer cast**
        - **Component sum**
        - **Polynomial accumulation**
      - **Hash table:**
      - **Compression function:**
        - **Using modulus**
        - **Multiply, add and take the modulus**
  - **Collisions:**
    - **Collision handling strategies:**
      - **Separate chaining**
      - **Open addressing**
      - **Double hashing**
    - **Analysis of collision handling strategies**
      - **Load factor**

Algorithmic Strategies:

- **Commonly used algorithmic strategies:**
  - **Brute Force:**
    - **Examples**
  - **Divide-and-conquer**
  - **Dynamic programming**
  - **Greedy algorithms**
- **Recursion:**
- **Recurrence relations**
  - **Master Theorem**
    - **Case1:**
    - **Case2:**
    - **Case3:**
- **Maximum subarray problem:**
  - **Using brute force**
    - **Optimized brute force**
  - **Divide-and-conquer**
  - **Dynamic programming**
    -
- **Sorting algorithms:**
  -
- **Binary Search Trees:**
  -
- **Binary heap**
  -
- **Graphs:**
  - **Representation:**
  - **Traversal algorithms:**
    - **BFS**
    - **DFS**
  - **Common problems:**
    - **Connected components**
    - **Minimum spanning tree**
      - **Kruskal**
      - **Prim**
    - **Shortest distance**
      - **Dijkstra**
      - **Bellman-Ford**
      - **Floyd Warshall**
    - **Max Flow/Min cut**
      - **Check document in Materials folder.**