Fundamentals of Data Structures and Algorithms

Data Structures

- A **data structure** is a special format for storing and organizing data.
- Two (2) types of data structure:
  - **Linear**: Elements are accessed in a sequential order but may be stored unsystematically.
  - **Non-Linear**: Elements are stored and accessed in a non-sequential order.
- An **abstract data type (ADT)** is a logical description of how data is viewed as well as the operations that are allowed without regard to how they will be implemented.
- Benefits of using ADT:
  - Code is easier to understand.
  - Implementations of ADTs can be changed without requiring changes to the program that uses the ADTs.
  - ADTs can be used in future programs.
- Two (2) parts of ADT:
  - **Public** or **external** – the data and the operations
  - **Private** or **internal** – the representation and the implementation
- Abstract data types:
  - **Linked list** – used for storing elements where each is a separate object.
  - **Stack** – an ordered list in which the last element added is the first element retrieved or removed (Last-In, First-Out).
  - **Queue** – an ordered list in which the first element added is the first element retrieved or removed (First-In, First-Out).
  - **Tree** – represents a hierarchical nature of a structure in a graphical form.
  - **Priority queue** – a special type of queue where elements are processed based on their order (natural or custom).
  - **Heap** – a complete binary tree where the value of each of each parent node is either higher or lower than the value of its child nodes.
  - **Set** – a collection of elements where each element is unique.
  - **Map** – a set of ordered pairs where elements are known as **keys** (identifiers) and **values** (content).
  - **Graph** – consists of a set of vertices (or nodes) and a set of edges (relations) between the pairs of vertices.
- The four (4) main operations that could be defined for each ADT are initializing, adding, accessing, and removing of data.

Algorithm

- An **algorithm** is a step-by-step set of instructions to be executed in sequence for solving a problem.
- Characteristics of an algorithm:
  - **Finiteness**: An algorithm must terminate after a specified number of steps.
  - **Definiteness**: Each instruction has to be clear and unambiguous.
  - **Input**: An algorithm should have zero or more well-defined data given before the algorithm begins.
  - **Output**: An algorithm must have one (1) or more results, with specified relation to the input.
  - **Uniqueness**: The result of each step depends on the input and/or the result of the previous step.
- Elements of an algorithm:
  - Sequential operations
  - Actions based on the state of a data structure
  - **Iteration** – repeating an action multiple times
  - **Recursion** – occurs when a function calls itself once or multiple times to solve a problem
- Algorithm design paradigms:
  - **Divide and Conquer**: A problem is broken into smaller subproblems.
  - **Greedy Algorithms**: The optimal approach is always chosen in solving a problem.
  - **Dynamic Programming**: Similar to Divide and Conquer except that the results of the subproblems are reused for overlapping subproblems.

References:

Karumanchi, N. (2017). *Data structures and algorithms made easy.* Hyderabad: CareerMonk Publications.

Lee, K. and Hubbard, S. (2015). *Data structures and algorithms with Python.* Cham: Springer International Publishing Switzerland.

Runestone Academy (n.d.). *Citing sources.* Retrieved from https://interactivepython.org/runestone/static/pythonds/index.html