The Ultimate DSA Reviewer

March 27, 2025

Introduction to Data Structures and Algorithms

- **Data Structures**
  - serves as containers for storing data
  - efficient data storage

Features of Data Structures

1. **Interface**
   - Set of operations and parameters that a data structure supports, accepts and return type
2. **Implementation**
   - Internal representation of a data structure

** An interface describes what a data structure does (methods, operations, etc), while an implementation describes how the data structure does it (the way the method or operation does something)*

Characteristics of Data Structures

1. **Correctness**
   - Implementation should run interface accurately
2. **Time Complexity**
   - Running / execution time should be as small as possible
3. **Space Complexity**
   - Memory usage should be as little as possible

Types of Data Structures

1. **Linear Data Structures**
   a. Arrays
   b. Linked Lists
   c. Stacks
   d. Queues

2. **Non-linear Data Structures**
    a. Trees
    b. Graphs
    c. Hash Tables

Operations on Data Structures

1. **Creation**
    - Creating a structure, first operation
2. **Insertion**
    - Adding new data elements in the structure
3. **Deletion**
    - Removal of a particular data element
4. **Updating**
    - Modifying a data element
5. **Searching**
    - Find location of a particular element
6. **Sorting**
    - Process of arranging data in some order
7. **Merging**
    - Combining data elements from two different lists
8. **Traversal**
    - Accessing each element exactly once so it can be processed
9. **Destruction**
    - Deletion of entire data structure

Basic Terminologies

1. **Data** - set of values
2. **Data item** - single unit of values
3. **Group items** - data items divided into sub items
4. **Elementary items** - data items that cannot be divided
5. **Attribute and entity** - entity which contains certain attributes or properties
6. **Entity set** - entities of similar attributes
7. **Field** - single elementary unit of information representing an attribute of an entity
8. **Record** - collection of field values of a given entity

9. **File** - a collection of records of the entities in a given entity set

- **Algorithms**
  - step-by-step procedures for solving problems
  - logic and sequence of operations

Characteristics of an algorithm:

1. **Input**
   - Reads the data of the given problem
2. **Output**
   - Desired result produced
3. **Process / Definiteness**
   - Each step is unambiguous (precise)
4. **Effectiveness**
   - Each step is concise and accurate
5. **Finiteness**
   - Each step is finite

Approaches for designing an algorithm

1. **Top-down approach**
   - Complicated algorithm should be divided into smaller modules
2. **Bottom-up approach**
   - Start with the basic up to the high level (complicated) modules

Analyzing an algorithm

1. **Time Complexity**
   - Amount of time taken by an algorithm to run the program completely

Execution time cases

1. **Worst Case**
   - Maximum time an operation can take
2. **Average Case**
   - Average time an operation can take

3. **Best Case**
    - Least possible execution time an operation can take

2. **Space complexity**
    - Amount of memory space required to run the program completely
    - Depends upon the input size

● Big O Notation

- Notates the time complexity of an algorithm
- Uses worst-case analysis
- Ignores constant factors, lower-order terms and adds notations if there are multiple terms

1. **Constant - O(1)**
    - Quickest time complexity, runs immediately once executed

2. **Logarithmic - O(log n)**
    - Reduces the input size in each step
      **example, start with 1000 items, then 500, then 250, 125, so on*

3. **Linear - O(n)**
    - Time complexity depends on the input size (n)

4. **Superlinear - O(n log n)**
    - Still individually processes each element (O(n)) but reduces it in each step (O(log n))

5. **Quadratic - O(n^2)**
    - Number of steps is equivalent to the square of the input size, where each element is compared with each other

6. **Cubic - O(n^3)**
    - Number of steps is equivalent to the cube of the input size

7. **Exponential - O(2^n)**

- Number of steps double as input size grows

8. **Factorial - O(n!)**
    - Number of steps is based off the factorial of the input size

Arrays

- ## Lists
    - An ordered set of a variable number of elements to which additions and deletions may be made
    - Simplest and commonly found type of data

    ### Linear Lists
        - A list which displays the relationship of physical adjacency of a finite sequence of data items or records
        - Has a single successor and single predecessor
- ## Array

- A type of linear data structure that is defined as a collection of elements with same or different data types
- Exists in both single and multiple dimensions

    - **Array index**
        - Value that labels the elements in an array *(ex: array[0])*
    - **Memory address**
        - Starting address of free memory available
    - **Element**
        - Item stored in an array
    - **Matrix / Matrices**
        - A mathematical object. A general matrix consists of m rows and n columns.
        - Usually made from multi-dimensional arrays.

Linked List

- ## Linked List

- A collection of nodes that together form a linear ordering
- Divided into two parts: Information and address

Types of Linked List

1. **Singly**
    - pointer is connected to the next variable
2. **Doubly**
    - pointer is connected to the next and previous variable
3. **Circular**
    - end value is connected to the start value

Stacks

- ## Stacks

- A fundamental data structure that operates on Last-In-First-Out (LIFO).
- Elements can only be added or removed from one end (the top)
- O(1) time complexity for insertion and deletion

Key Operations

1. **push( )**
    - Adds an element on top of the stack
2. **pop( )**
    - Removes and returns top element from stack
3. **peek( ) / top( )**
    - Returns the top element without removing it
4. **isEmpty( )**
    - Checks if the stack is empty

- All time complexity of all key operations is O(1)
- **Overflow:** Condition when a stack is full
- **Underflow:** Condition when trying to pop from an empty stack

Applications of Stacks

1. **Function Calls**
    - Store return addresses and local variables
2. **Expression Evaluation**
    - Evaluate arithmetic expressions
3. **Backtracking**

- Keep track of the state of a problem during backtracking algorithms

4. **Undo/Redo Functionality**
    - Implements undo and redo functions in applications

- **Expression Notations and Stack Implementations**

Expression Notations

A. Infix

- Standard mathematical notation where operators are placed between operands (A + B * C)

B. Postfix / Reverse Polish Notation

- Operators follow their operands (A B C * +) = (A + B * C)

C. Prefix / Polish Notation

- Operators precede their operands (+ A * B C) = (A + B * C)