Data Structures and Algorithms Notes Cleaned

Data Structures and Algorithims (Jomo Kenyatta University of Agriculture and Technology)

Data Structures and Algorithms Notes

1. Arrays

   - Definition: A collection of elements, all of the same type, stored in contiguous memory locations.
   - Operations:
     - Access: O(1)
     - Search: O(n)
     - Insertion: O(n) (for unsorted array)
     - Deletion: O(n)
   - Applications: Storing data in a fixed size, implementing lists, matrices, etc.

2. Linked Lists

   - Definition: A linear data structure where elements (nodes) are stored in non-contiguous memory locations. Each node contains data and a reference (or link) to the next node.
   - Types:
     - Singly Linked List: Each node points to the next node.
     - Doubly Linked List: Each node has a reference to both the next and previous node.
   - Operations:
     - Insertion/Deletion: O(1) if the node is given (efficient).
     - Traversing: O(n)
   - Applications: Dynamic memory allocation, implementing stacks/queues.

3. Stacks

   - Definition: A linear data structure that follows the **Last In First Out (LIFO)** principle.
   - Operations:
     - Push: O(1)

  - Pop: O(1)

  - Peek/Top: O(1)

- Applications: Undo operations in text editors, parsing expressions, recursion.

4. Queues

- Definition: A linear data structure that follows the **First In First Out (FIFO)** principle.

- Operations:

  - Enqueue: O(1)

  - Dequeue: O(1)

  - Peek/Front: O(1)

- Applications: Scheduling tasks, print spooling, breadth-first search in graphs.

5. Trees

- Definition: A hierarchical data structure made up of nodes, where each node has a value and references to its children.

- Binary Tree: A tree in which each node has at most two children.

- Binary Search Tree (BST): A binary tree where the left child's value is smaller and the right child's value is larger than the parent's value.

- Operations:

  - Insertion: O(log n) in balanced BSTs.

  - Search: O(log n) in balanced BSTs.

  - Deletion: O(log n) in balanced BSTs.

- Applications: Expression trees, decision trees, binary heaps, and searching algorithms.

6. Graphs

- Definition: A collection of nodes (vertices) connected by edges.

- Types:

- **Directed Graph (Digraph)**: Edges have a direction.

- **Undirected Graph**: Edges do not have a direction.

- **Weighted Graph**: Edges have weights/costs.

- Operations:

  - Traverse: BFS (Breadth-First Search) or DFS (Depth-First Search).

- Applications: Social networks, shortest path algorithms, network routing.

7. Sorting Algorithms

- Bubble Sort: O(n^2) - Simple but inefficient, repeatedly swaps adjacent elements if they're in the wrong order.

- Selection Sort: O(n^2) - Finds the minimum element and places it at the beginning, then repeats for the rest of the array.

- Insertion Sort: O(n^2) - Builds the sorted array one element at a time, inserting elements into the correct position.

- Quick Sort: O(n log n) - Divides the array into smaller sub-arrays and sorts them recursively.

- Merge Sort: O(n log n) - Divides the array into halves, sorts each half, and then merges them.

- Applications: Organizing data, searching, and optimization problems.

8. Searching Algorithms

- Linear Search: O(n) - Checks each element in the list sequentially.

- Binary Search: O(log n) - Works on sorted arrays, repeatedly divides the search space in half.

- Applications: Looking for elements in arrays, databases, etc.

Key Concepts & Big O Notation:

- Time Complexity: The amount of time an algorithm takes to complete as a function of the size of the input.

- Space Complexity: The amount of memory an algorithm uses as a function of the input size.

- Big O Notation: A way of describing the efficiency of an algorithm (worst-case scenario).
  - O(1): Constant time.
  - O(n): Linear time.
  - O(log n): Logarithmic time.
  - O(n^2): Quadratic time.

Advanced Topics (Optional):

- Hashing: Efficient data retrieval using hash functions.
- Heaps: A special tree-based structure used to implement priority queues.
- Dynamic Programming: Solving complex problems by breaking them down into simpler subproblems.