DATA STRUCTURE WEEK - 3

- **Tree** - A tree is a **nonlinear** hierarchical data structure that consists of nodes
    - **Advantages**
        - **Efficient searching**: Trees are particularly efficient for searching and retrieving data. The time complexity of searching in a tree is typically O(log n), which means that it is very fast even for very large data sets.
        - **Flexible size**: Trees can grow or shrink dynamically depending on the number of nodes that are added or removed. This makes them particularly useful for applications where the data size may change over time.
        - **Fast insertion and deletion**: Inserting and deleting nodes in a tree can be done in O(log n) time, which means that it is very fast even for very large trees
    - **Disadvantages**
        - **Imbalanced trees**: If a tree is not balanced, it can result in uneven search times. This can be a problem in applications where speed is critical.
- **Binary tree** - A binary tree is a type of tree data structure in which each node can have at most two child nodes. In a binary tree, each node can have zero, one, or two children that are less than the value of its parent node, while the value of the right child is greater than the value of its parent node. This property of BSTs ensures that the nodes in the tree are always ordered, making it easy to search, insert and delete nodes in the tree.BSTs have a time complexity of **O(log n)** for searching, insertion, and deletion operations. They are commonly used in various applications such as **database indexing**, **file systems**, and **sorting algorithms.**

        - **Complete tree** - A complete binary tree is a binary tree in which all levels are filled, except possibly the last level, which is filled from left to right.
        - **Full tree** - Full tree is a type of tree in which every node has either zero or two children.
        - **Perfect tree** - A perfect binary tree is a type of binary tree in which all internal nodes have exactly two children, and all leaf nodes are at the same level or depth. This means that the tree is completely filled, and every level has the maximum possible number of nodes.

        - **Iterative - O(1)**
        - **Recursive - O(logn) (balanced tree)**
            - **worst case - O(n) (unbalanced tree)**
        - **Traversing - Time complexity - O(n)ST**
            - **Inorder - (left, root, right)**
            - **Preorder - (Root, left, right)**
            - **Postorder - (left, right, root)**

- **Heaps** - If the heap is implemented as a binary heap, the time complexity of **inserting** an element is **O(log n) Delete - O(log n)**
    - **Min Heap** - in a min heap, the value of each parent node is less than or equal to the values of its children.
        - To find Parent - **(i - 1) / 2**
        - To Find Child - **2*i + 1***
            - **Max Heap** - In a max heap, the value of each parent node is greater than or equal to the values of its children.
                - To Find Parent - **(i - 1) / 2**
                - To Find Child - **2*i + 1***
            - ***Disad*vantages**
                - **Lack of flexibility:** The heap data structure is not very flexible, as it is designed to maintain a

specific order of elements. This means that it may not be suitable for some applications that require more flexible data structures.

- **Applications**
  - **Job Scheduling**: In job scheduling applications, a heap can be used to efficiently maintain a list of jobs, where each job has a priority or deadline associated with it.
  - **Priority Queues:** A heap is often used to implement a priority queue, where the elements are inserted and removed according to a priority or weight associated with each element. For example, in an emergency room, patients are often prioritized based on the severity of their condition, and a heap can be used to implement the queue.

- **Tries** - Trie, also known as a prefix tree, is a data structure used for efficient string searching and retrieval operations. It is a tree-like data structure that stores strings as sequences of characters in a path from the root to a leaf node. Each node in the tree represents a prefix of one or more strings.
  - Insertion - **O(L)TS** L- length
  - Deletion - **O(L) TS**
    - **Applications**
      - **Auto-completion**: Tries are often used to implement auto-completion features in text editors and search engines. With a trie, we can quickly search for all

possible strings with a given prefix, which is essential for providing suggestions as the user types.
- **Spell-checking**: Tries can be used to implement efficient spell-checking algorithms, where we need to search for a given word in a dictionary of valid words.

- **Graph** - It is a Non-structured data structure A graph is a collection of nodes (also known as vertices) connected by edges. data is stored in the Key Value pair, key - vertex, value - edge, and Graph is implemented in the hashtable. **O(E+V)TS**
    - **Unidirectional** - Direction to Only One side
    - **Bidirectional** - Direction to both sides
    - Cyclic graph, Noncyclic graph
    - Disconnected graph, Connected graph
    - **Weighted graph** - A graph in which the edges are already specified with suitable weight is known as a weighted graph.
    - **Spanning Tree** - A spanning tree is a subset of Graph G, which has all the vertices covered with the minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected.
    - **Minimum Spanning Tree(MST)** for a weighted, connected, undirected graph is a spanning tree having a weight less than or equal to the weight of every other possible spanning tree.
        - Delete - **O(E)T**, **O(1)S**
        - Insert - **O(E)T**
        - Traversal - **O(1)T**
            - **Applications**
                - Mutual friends in social media
                - Google map
            - **Advantages**
                - **Efficient data processing**: Graphs can be processed efficiently using graph algorithms, which are specifically designed to work with graph data structures. This makes it

possible to perform complex operations on large datasets quickly and effectively.

- **Breadth-first search (BFS)** is an algorithm used to traverse a graph or tree data structure. The algorithm starts at a particular node and explores all its neighbors at the same depth (also known as level) before moving on to the next level. The algorithm continues this process until all nodes in the graph have been explored. **O(E+V)TS**

- **Depth-first search (DFS)** is an algorithm used to traverse a graph or tree data structure. The algorithm starts at a particular node and explores as far as possible along each branch before backtracking. The algorithm continues this process until all nodes in the graph have been explored. **O(E+V)TS**