# DS - 2nd WEEK

## Sorting Algorithms

1. **Bubble Sort**
   - Concept and algorithm
   - Complexity analysis (O(n^2))
   - Stability and in-place sorting
   - Applications
2. **Insertion Sort**
   - Concept and algorithm
   - Complexity analysis (O(n^2))
   - Stability and in-place sorting
   - Applications
3. **Selection Sort**
   - Concept and algorithm
   - Complexity analysis (O(n^2))
   - Stability and in-place sorting
   - Applications
4. **Quick Sort**
   - Concept and algorithm
   - Complexity analysis (O(nlog n) average, O(n^2) worst-case)
   - Stability and in-place sorting
   - Applications
   - Deterministic vs. Non-deterministic
5. **Merge Sort**
   - Concept and algorithm
   - Complexity analysis (O(nlog n))
   - Stability and out-of-place sorting
   - Applications
   - Divide and conquer strategy

## Stack

1. **Concept of Stack**
   - Definition and properties (LIFO)
   - Applications
2. **Operations**
   - PUSH
   - POP
   - Peek
   - Display elements
3. **Concepts**
   - Stack overflow vs. underflow
   - Types of stacks
   - Pros of using Linked List for stack

## Queue

1. **Concept of Queue**
    - Definition and properties (FIFO)
    - Applications
2. **Operations**
    - Enqueue
    - Dequeue
    - Peek
    - Display elements
3. **Concepts**
    - Types of queues (simple, circular, priority)
    - Double-ended queue (Deque)
    - Pros of using Linked List for queue

## Hash Table

1. **Concept of Hash Table**
    - Definition and properties
    - Applications
2. **Hash Function**
    - Concept and types
    - Generating hash values
3. **Collisions**
    - Definition
    - Methods to prevent collisions (chaining, linear probing, quadratic probing, double hashing)
4. **Load Factor**
    - Definition and importance
    - Calculating load factor
5. **Comparison with Arrays**
    - Differences between arrays and hash tables
    - Pros and cons of each

# DS2- Practice

## Sorting Algorithms

1. **Bubble Sort**
    - Implement Bubble Sort and test it on various datasets.
    - Modify Bubble Sort to detect already sorted arrays and terminate early.
    - Compare the performance of Bubble Sort with other sorting algorithms for different types of input (e.g., sorted, reverse sorted, random).
2. **Insertion Sort**

   - Implement Insertion Sort and test it on various datasets.
   - Modify Insertion Sort to use binary search to find the position to insert an element.
   - Compare the performance of Insertion Sort with Bubble Sort and Selection Sort on small arrays.
3. **Selection Sort**
   - Implement Selection Sort and test it on various datasets.
   - Modify Selection Sort to find both the minimum and maximum in each iteration.
   - Compare the performance of Selection Sort with other O(n^2) sorting algorithms.
4. **Quick Sort**
   - Implement Quick Sort using different pivot selection strategies (first element, last element, middle element, random element).
   - Analyze the performance of Quick Sort on different types of input (e.g., sorted, reverse sorted, random).
   - Implement a non-recursive version of Quick Sort using an explicit stack.
5. **Merge Sort**
   - Implement Merge Sort and test it on various datasets.
   - Modify Merge Sort to sort linked lists.
   - Compare the performance of Merge Sort with Quick Sort for large datasets.

## Stack

1. **Basic Operations**
   - Implement a stack using arrays.
   - Implement a stack using linked lists.
   - Write functions to perform PUSH, POP, and Peek operations and display stack elements.
2. **Advanced Concepts**
   - Write a program to check for balanced parentheses in an expression using a stack.
   - Implement a stack that supports getting the minimum element in constant time.
   - Compare the performance of stack operations implemented using arrays vs. linked lists.

## Queue

1. **Basic Operations**
   - Implement a queue using arrays.
   - Implement a queue using linked lists.
   - Write functions to perform Enqueue, Dequeue, and Peek operations and display queue elements.
2. **Types of Queues**
   - Implement a circular queue and test it with various operations.
   - Implement a priority queue and test it with different priorities.
   - Implement a double-ended queue (Deque) and test all operations.

3. **Applications**
    - Write a program to simulate a printer queue.
    - Implement a breadth-first search (BFS) algorithm using a queue.
    - Compare the performance of queue operations implemented using arrays vs. linked lists.

## Hash Table

1. **Basic Implementation**
    - Implement a hash table using chaining for collision resolution.
    - Implement a hash table using linear probing for collision resolution.
    - Implement a hash table using quadratic probing for collision resolution.
    - Implement a hash table using double hashing for collision resolution.
2. **Hash Functions**
    - Write a custom hash function for strings.
    - Write a custom hash function for integers.
    - Analyze the distribution of hash values generated by different hash functions.
3. **Collision Handling**
    - Compare the performance of different collision handling techniques (chaining, linear probing, quadratic probing, double hashing) under various load factors.
    - Implement a resize operation for hash tables to maintain a low load factor.
    - Write a program to find the most frequent element in an array using a hash table.
4. **Applications**
    - Use a hash table to implement a dictionary.
    - Implement a spell checker using a hash table.
    - Write a program to find duplicates in an array using a hash table.

## General Practice

1. **Sorting Algorithms**
    - Write test cases to evaluate the performance of all sorting algorithms.
    - Visualize the sorting process using graphical representations.
2. **Stacks and Queues**
    - Implement a browser history using a stack.
    - Implement an undo feature using a stack.
    - Simulate a ticket booking system using a queue.
    - Write a program to schedule tasks based on priority using a priority queue.
3. **Hash Tables**
    - Write a program to count the frequency of words in a text file using a hash table.
    - Implement a cache system using a hash table.
    - Write a program to perform join operations on two datasets using hash tables.