23AID204 - Advanced Data Structures & Algorithm Analysis
Lab Questions

W1&2	Recap of DS1 - LL, Queues, Stacks, Graphs, DFS, BFS
W3	Binary Tree
	1. **Tree Construction:** • Construct a binary search tree from a given list of numbers. • Given an in-order and pre-order traversal of a binary tree, reconstruct the tree. 2. **Tree Height and Depth:** • Write a function to compute the height of a binary tree. • Determine the depth of a specific node in a binary tree. 3. **Advanced Operations:** • Write a function to delete a node from a binary search tree. Ensure the tree remains a valid BST after deletion. • Implement a function to check if a binary tree is balanced. 4. **Tree Analysis *(Optional)*** • Given a binary tree, write a function to check if it is a complete binary tree. • Write a function to find the lowest common ancestor (LCA) of two nodes in a binary search tree.
W4.	**Heaps**
	1. **Understanding Heap Property:** o Given a list of numbers, create a max-heap and a min-heap manually and show the binary tree representation of both. o Explain the difference between a max-heap and a min-heap. Provide an example of each. 2. **Basic Operations on Heaps:** o Write a function to insert a new element into a max-heap. Show each step of the insertion process on a provided heap. o Write a function to delete the root element from a min-heap. Describe how the heap is restructured after deletion. 3. **Heap Properties:** o Describe the properties of a binary heap. What makes a binary heap different from other binary trees? o Given a binary tree, determine if it satisfies the heap property. Justify your answer. 4. **Heap Construction:** • Write a function to convert an unsorted array into a max-heap using the heapify process. Demonstrate the steps with a given example array. • Implement a function to convert a max-heap into a min-heap. Explain the modifications needed to transform the heap structure. 5. **Heapsort Implementation:** • Write a program to implement the Heapsort algorithm on a given array of

	integers. Show the state of the heap after each extraction and sorting step. • Given a max-heap, perform Heapsort and detail each step of the sorting process, including the intermediate states of the heap. 6. **Heap Applications:** ***(Optional)*** • Explain how a priority queue can be implemented using a binary heap. Write a function to demonstrate priority queue operations (insert, extract-max/min). • Discuss the time complexity of heap operations (insert, delete, extract-max/min) and how Heapsort compares to other sorting algorithms in terms of performance.
W5 &6	AVL Trees
	Low-Level Questions 1. **Understanding the Balance Factor:** o Given a series of numbers to insert into an empty AVL tree, compute the balance factor for each node after each insertion. o Explain what a balance factor is in an AVL tree. Why is it important for maintaining tree balance? 2. **Identifying Rotations:** o Given a set of AVL tree nodes, determine if any rotations are needed after inserting a new node. Identify the type of rotation required (left, right, left-right, or right-left). o Draw the result of performing a single right rotation on a given subtree of an AVL tree. 3. **Basic Rotations:** o Perform a right rotation on the given AVL tree at the specified node. Show the new structure of the tree after rotation. o Perform a left rotation on the given AVL tree at the specified node. Illustrate the updated tree. 4. **Combination Rotations:** o Given an AVL tree, perform a left-right rotation to balance it. Describe the steps and the resulting tree structure. o Perform a right-left rotation on the given AVL tree and explain why this rotation was necessary. Show the intermediate and final steps. 5. **Balancing an AVL Tree:** o Write a function to insert a node into an AVL tree and ensure the tree remains balanced after each insertion. Demonstrate the function with a series of insertions that cause different types of rotations. o Implement a function to delete a node from an AVL tree and maintain its balance. Illustrate the deletion process with a tree where multiple rotations are needed to rebalance. 6. **Tree Analysis and Rotations:** ***(Optional)***

	- Given a set of insertions into an AVL tree, identify all the rotations that occur. Show the tree's structure after each rotation. - Explain the difference between single and double rotations in AVL trees. Provide an example where both types of rotations are needed.
W7	**Trie**
	1. **Basic Trie Construction:** - Write a function to insert a word into a Trie. Demonstrate this function by inserting the words "cat", "car", and "cart". - Given an empty Trie, insert the following words: "bat", "ball", "batter". Show the resulting structure of the Trie. 2. **Searching in a Trie:** - Write a function to search for a word in a Trie. Use this function to check if the words "bat", "ball", and "batman" are present in a Trie that contains "bat" and "ball". - Explain how searching for a word in a Trie differs from searching in a binary search tree (BST). 3. **Prefix Check:** ***(Optional)*** - Write a function to check if a given prefix exists in a Trie. Test this function with the prefixes "ba", "bat", and "cat" on a Trie containing the words "bat", "ball", and "basket". - Explain the difference between checking for a word and checking for a prefix in a Trie. 4. **Autocomplete Feature:** ***(Optional)*** - Implement an autocomplete function using a Trie that returns all words with a given prefix. Test this function with the prefix "ca" on a Trie containing the words "cat", "car", "cart", "cattle". - How would you modify the Trie to store additional data (like the frequency of a word) to improve the autocomplete feature? 5. **Word Deletion:** ***(Optional)*** - Write a function to delete a word from a Trie. Use this function to remove the word "bat" from a Trie that contains "bat", "ball", and "batter". Show the Trie structure after deletion. - What challenges arise when deleting words from a Trie? How do you handle cases where deleting a word affects other words with common prefixes? 6. **Longest Common Prefix:** ***(Optional)*** - Implement a function to find the longest common prefix among a set of words stored in a Trie. Test this function with the words "interview", "integrate", "integer". - How does the structure of a Trie facilitate finding the longest common prefix? 7. **Counting Words with a Given Prefix:** ***(Optional)*** - Write a function to count the number of words in a Trie that start with a given prefix. Test this function with the prefix "pre" on a

	Trie containing "prefix", "preposition", "presentation", "pretty". o What is the time complexity of this operation, and why is it efficient in a Trie?
W8	**Hash Tables**
	1. Write a simple hash function for integers using the modulus operator. Test your function with the integers `7`, `12`, and `15`. Explain the outputs. 2. Design a basic hash function that maps each string to an integer value. Use your function to hash the strings "apple", "banana", and "cherry". Document and explain your approach and results. 3. Create a hash table to store integer keys and their corresponding values. Write functions for inserting values and retrieving values based on their keys. Demonstrate these functions with at least five key-value pairs. 4. Using the hash function `h(x) = x % 5`, insert the integers `7` and `12` into a hash table. Show the table state and explain why a collision occurs. ***(Optional)*** 5. Write a function that implements quadratic probing for collision resolution. Insert the values `13`, `23`, `33`, and `43` into a hash table of size `10` using your function. Show the state of the table after each insertion and explain your observations. ***(Optional)***
W9	**Merkel trees**
	1. Write a function to calculate the hash of a data block using a simple hash function (e.g., SHA-256). Use this function to compute the leaf node hashes for data blocks `E`, `F`, `G`, and `H`. Construct a Merkle tree from these blocks and calculate the hash of the root node. 2. Implement a program that constructs a Merkle tree from an arbitrary number of data blocks. Test your program with the following set of data blocks: `["block1", "block2", "block3", "block4", "block5"]`. Explain how your program handles cases where the number of leaf nodes is not a power of two. 3. Explain the use of Merkle trees in blockchain technology. Create a simplified blockchain with three blocks using a Merkle tree to store transactions within each block. Show how the Merkle root changes when a transaction in the first block is altered. ***(Optional)***