**day19\_107856406\_dsdipt\_sudipto\_16july2025**

**Employee Code:** 107856406

**Login ID:** dsdipt

**Email :** dsdipt@amazon.com

**Name:** Sudipto Das

**Date:** 17 July 2025 (Day 19)

### ***Task 01: Benefit of Trie in Autocomplete***

**Q:** In a trie (prefix tree), what is the most significant benefit it provides in information retrieval systems like autocomplete?

**✔️ 2. It enables prefix-based searching by storing characters in a tree-like format, reducing lookup time.**

### ***Task 02: Stable vs Unstable Sorting***

**Q:** What do you understand by stable and unstable sorting?  
 **A:**

* **Stable sorting** maintains the relative order of records with equal keys (i.e., values).
* **Unstable sorting** does **not** preserve the original relative order of equal elements.  
   Example: Merge sort is stable, while quicksort is generally unstable unless implemented carefully.

### ***Task 03: DFS vs BFS***

**Q:** In graph traversal, what is a defining feature of depth-first search (DFS) compared to breadth-first search (BFS)?

**✔️ 2. DFS explores each path as deeply as possible before backtracking, often implemented using a stack.**

### ***Task 04: Reversing Pointers in Linked List***

**Q:** What is the primary purpose of reversing the pointers in the linked list?

**✔️ 3. To perform in-place reversal of the list with O(1) space**

### ***Task 05: Binary Tree BFS Logic***

**Q:** How does the binary tree traversal logic work in BFS?

**✔️ 3. It performs level-by-level traversal using a queue**

### ***Task 06: Output of BFS from Node 1***

**Q:** What will be printed by BFS graph with starting node 1?

**✔️ 2. Level-order traversal of all connected nodes from node 1**

### ***Task 07: Traversal Type in BST In-Order Function***

public void inorder(TreeNode root) {

if (root == null) {

return;

}

inorder(root.left);

System.out.print(root.val + " ");

inorder(root.right);

}

**Q:** What is the traversal type in this BST in-order function?

**✔️ 2. In-order traversal resulting in sorted order for BST**

### ***Task 08: O(log n) in BST Context***

**Q:** What does O(log n) signify when used in the context of a binary search tree operation?

**✔️ 3. The number of steps grows proportionally to the logarithm of the input size, typical for balanced trees.**

### ***Task 09: Linked List Queue vs Array Queue***

**Q:** What distinguishes a queue implemented with a linked list from one implemented using an array in terms of performance?

**✔️ 3. Linked list-based queues avoid resizing operations, providing consistent performance during enqueue and dequeue.**

### ***Task 10: Why Binary Search Needs Sorted Data***

**Q:** In a binary search algorithm, why must the input data be sorted before execution?

**✔️ 3. Sorting allows the algorithm to eliminate half of the search space in each step, achieving O(log n) time.**

### ***Task 11: Stack Using Linked List***

**Q:** What is the significance of using a linked list to implement a stack instead of an array?

**✔️ 3. Linked list-based stacks avoid overflow by dynamically growing in memory without the need for resizing arrays.**