**ID – hrajranj**

**Day 19 – 17th July 2025**

**Task 01:**

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

1. It stores keys in a hash map allowing faster lookup than string comparison.

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

3. It compresses all values into a single hash index for instant access.

4. It eliminates the need for traversal by maintaining precomputed suggestions for each node.

**Task 02:**

**What do you understand by stable and unstable sorting.**

**Ans:**

**Stable Sorting**

A **stable sorting algorithm** maintains the **relative order of records with equal keys**.

* If two elements are equal, their original order remains unchanged after sorting.
* Useful when data has multiple keys and you've already sorted it by one—stable sort lets you preserve previous ordering.

**Example:** Imagine sorting students by age, and several are 18. If they were previously sorted by name, a stable sort will keep that name order intact among the 18-year-olds.

🔹 Stable sorts in Java include:

* Bubble Sort
* Insertion Sort
* Merge Sort
* TimSort (used in Java’s Arrays.sort() for objects)

**Unstable Sorting**

An **unstable sorting algorithm** may **change the relative order of equal elements**.

* It still sorts correctly but doesn’t guarantee that equal items will stay in the same sequence.

**Example:** Back to the students—sorting by age using an unstable sort might rearrange the 18-year-olds, even if they were already ordered by name.

🔸 Unstable sorts in Java include:

* Quick Sort
* Heap Sort
* Selection Sort

**Task 03**

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

1. DFS uses a queue to ensure all sibling nodes are visited before moving deeper.

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

3. DFS ensures minimum number of edges are traversed by default.

4. DFS always finds the shortest path in weighted graphs

**Task 04:**

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

1. To convert singly linked list into doubly linked list

2. To delete all the nodes in reverse order

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

4. To traverse backwards using a stack

**Task 05:**

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

1. It visits all right nodes first and then left nodes

2. It performs in-order traversal level by level

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

4. It uses recursion for pre-order traversal

**Task 06:**

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

1. Depth-first traversal order from node 1

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

3. Nodes printed in reverse due to stack usage

4. Only prints the root node as others are skipped

**Task 07:**

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

class TreeNode {

    int val;

    TreeNode left, right;

    TreeNode(int v) {

        val = v;

    }

}

public class BSTInOrder {

       public void inorder(TreeNode root) {

        if (root == null) {

            return;

        }

        inorder(root.left);

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

        inorder(root.right);

    }

}

1. Pre-order traversal where root is visited first

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

3. Level-order traversal using recursion

4. Post-order traversal used for deleting node

**Task 08:**

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

1. The number of steps grows linearly with the size of the input.

2. The operation takes exponential time depending on tree height.

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

4. The operation performs a constant number of steps for each input regardless of size.

**Task 09:**

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

1. Array-based queues allow two-directional traversal, making them superior for complex operations.

2. Array-based queues can expand without limit, offering better memory efficiency.

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

4. Linked list-based queues require preallocation of memory which improves speed.

**Task 10:**

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

1. Binary search modifies the array structure, so sorting prevents errors.

2. Binary search only works with integer values, which are easier to sort.

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

4. Sorting ensures that every item has a fixed memory address, improving cache locality.

**Task 11:**

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

1. Linked list implementation leads to slower operations but saves space due to non-contiguous memory.

2. Linked list stacks prevent duplicate values and automatically enforce element uniqueness.

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

4. Linked list stacks operate using a tree-like structure for better depth analysis.