**ID: hrajranj**

**Day 16 – 9th July 2025**

**1. What is the difference between binary tree and binary search tree (bst)**

**Can you explain diff between structure and operation of Binary tre and BST.**

**Ans:**

**Binary Tree: A Binary Tree is a hierarchical data structure in which:**

**• Each node has at most two children.**

**• These children are referred to as the left and right child.**

**Binary Search Tree (BST): A Binary Search Tree (BST) is a special type of binary tree where:**

**• The left child contains only nodes with values less than the parent.**

**• The right child contains only nodes with values greater than the parent.**

**• No duplicate nodes are typically allowed.**

Operations Differences

Search

* Binary Tree: Must traverse all nodes → O(n) time.
* BST: Can ignore half the tree at each step → O(log n) time (balanced BST).

Insert

* Binary Tree: Insert wherever there’s space.
* BST: Insert such that order is preserved.

Delete

* Binary Tree: Just remove or replace node.
* BST: Must preserve order → more complex rules (especially with 2 children).

**2. In sorted array why do you think binary search tree is best than linear search.. Can you explain plz**

**Ans:**

When working with a sorted array, binary search (or using a Binary Search Tree) is better than linear search because it's much faster. In linear search, you check each element one by one until you find the target. This takes time depending on the size of the array — O(n) time. But in binary search, you keep dividing the array in half and compare with the middle element. This means fewer steps — only O(log n) time. A Binary Search Tree (BST) also works like this when balanced. It allows fast search, insert, and delete operations. So, binary search or BST is best for sorted data because it’s quicker and more efficient than checking every element.

**3. Difference between static and dynamic arrays.. Plz list them**

**Ans:**

Static Array:

Fixed Size – Size must be declared at creation.

Memory Allocation – At compile-time.

Faster Access – Slightly faster as memory is contiguous.

Less Flexible – Cannot resize once created.

Dynamic Array:

Resizable – Can grow or shrink at runtime.

Memory Allocation – At runtime (heap memory).

Slightly Slower – Due to resizing and copying elements.

More Flexible – Suitable when size is unknown in advance.

**4. In BFS, DFS which one is more preferred in terms of shortest path for the unweighted graphs.**

**Note: BFS explores nodes in increasing distance order from the source, ensuring shortest paths are found first.**

**Ans:**

For unweighted graphs, BFS (Breadth-First Search) is more preferred when finding the shortest path.

BFS explores all nodes level by level (i.e., in increasing order of distance from the source). As soon as it reaches the destination node, it guarantees that the path found is the shortest in terms of the number of edges. It uses a queue, ensuring that nodes closer to the source are explored before distant ones.

DFS is not suitable for shortest path in unweighted graphs. DFS goes deep into one branch before backtracking, so it might reach the destination via a longer path. It does not guarantee the shortest path unless you explore all paths and compare (which is inefficient).

Use BFS when you want the shortest path in an unweighted graph. DFS is better suited for path existence, topological sorting, or exploration, but not for shortest paths.

**5. Write a code to implement a stack using an array.**

**Note: plz ensure bounds are checked to avoid overflow/underflow**

Answer:

public class ArrayStack {

int maxSize;

int[] stack;

int top;

// Constructor to initialize stack

public ArrayStack(int size) {

maxSize = size;

stack = new int[maxSize];

top = -1; // Initially stack is empty

}

// Push operation with overflow check

public void push(int value) {

if (top == maxSize - 1) {

System.out.println("Stack Overflow! Cannot push " + value);

return;

}

stack[++top] = value;

System.out.println(value + " pushed to stack.");

}

// Pop operation with underflow check

public int pop() {

if (top == -1) {

System.out.println("Stack Underflow! Nothing to pop.");

return -1;

}

int poppedValue = stack[top--];

System.out.println(poppedValue + " popped from stack.");

return poppedValue;

}

// Peek operation to see top value

public int peek() {

if (top == -1) {

System.out.println("Stack is empty.");

return -1;

}

return stack[top];

}

// Check if stack is empty

public boolean isEmpty() {

return top == -1;

}

// Display all elements in the stack

public void display() {

if (top == -1) {

System.out.println("Stack is empty.");

return;

}

System.out.print("Stack elements: ");

for (int i = 0; i <= top; i++) {

System.out.print(stack[i] + " ");

}

System.out.println();

}

// Main method to test

public static void main(String[] args) {

ArrayStack myStack = new ArrayStack(5); // Stack size 5

myStack.push(10);

myStack.push(20);

myStack.push(30);

myStack.push(40);

myStack.push(50);

myStack.push(60); // Overflow

myStack.display();

myStack.pop();

myStack.pop();

myStack.display();

System.out.println("Top element is: " + myStack.peek());

}

}