# VIVA DSA LAB 2

1. **Stack using Linked List:**
   Q: What is a stack?
   A: A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle.

   Q: How is a stack implemented using a linked list?
   A: Each node in the linked list contains an element and a pointer to the next node.

2. **Queue using Linked List:**
   Q: What is a queue?
   A: A queue is a linear data structure that follows the First-In-First-Out (FIFO) principle.

   Q: How is a queue implemented using a linked list?
   A: It is implemented similarly to a stack, but the difference lies in how elements are removed from the list (front instead of rear).

3. **Binary Search Trees:**
   Q: What is a binary search tree?
   A: A binary search tree is a binary tree in which the left child is less than the parent, and the right child is greater.

   Q: How is a node inserted into a binary search tree?
   A: To insert a node, we compare the value with the current node and proceed either to the left or right subtree until an empty spot is found.

4. **Hashing:**
   Q: What is hashing?

A: Hashing is a technique used to map data to a fixed-size array (hash table) using a hash function.

Q: What is a hash collision?
A: A hash collision occurs when two different data elements produce the same hash value.

5. **Balanced Parentheses:**
Q: What are balanced parentheses?
A: Balanced parentheses mean that for every opening bracket, there is a corresponding closing bracket in the correct order.

Q: How can you check for balanced parentheses using a stack?
A: You can scan the expression from left to right, push opening brackets onto the stack, and when you encounter a closing bracket, pop from the stack and check if they match.

6. **Infix to Postfix Conversion:**
Q: What is infix notation?
A: Infix notation is the standard way of writing expressions, where operators are placed between operands.

Q: How can you convert infix to postfix notation?
A: Using the stack, you can scan the infix expression and apply certain rules to convert it to postfix notation.

7. **Infix to Prefix Conversion:**
Q: How can you convert infix to prefix notation?
A: By first converting the infix expression to postfix and then reversing the resulting postfix expression, you can obtain the prefix notation.

8. **Postfix Evaluation:**
   Q: What is postfix evaluation?
   A: Postfix evaluation is the process of calculating the result of a postfix expression using a stack.

   Q: How do you perform postfix evaluation using a stack?
   A: You scan the postfix expression from left to right, and when you encounter an operand, push it onto the stack. When you encounter an operator, pop the required number of operands, perform the operation, and push the result back onto the stack.

9. **Sparse Matrix Conversion:**
   Q: What is a sparse matrix?
   A: A sparse matrix is a matrix with a large number of zero elements.

   Q: How can you efficiently represent a sparse matrix?
   A: You can use a triplet representation or a linked list representation to store only the non-zero elements.

10. **Sequential Search:**
   Q: What is sequential search?
   A: Sequential search is a linear search algorithm that sequentially checks each element in a list until a match is found.

   Q: What is the time complexity of sequential search?
   A: The worst-case time complexity of sequential search is O(n), where n is the number of elements in the list.

11. **Binary Search:**
   Q: What is binary search?

A: Binary search is a search algorithm that efficiently finds the position of a target value within a sorted array.

Q: What is the time complexity of binary search?
A: The time complexity of binary search is O(log n) in the worst case.

12. Difference between #define and const

#define: A preprocessor directive for text replacement, lacks type checking, no memory allocation, and global scope.
const: A keyword to declare constants, has type checking, memory allocation, follows scoping rules, and allows namespacing.

13. Difference between infix, postfix and prefix expression

Infix Expression:

The most common way of writing expressions.
Operators are written between operands.
Requires parentheses to define operator precedence.
Example: (2 + 3) * 4

Postfix Expression (Reverse Polish Notation - RPN):

Operators are written after their operands.
No need for parentheses as operator precedence is implicit.
Easier to evaluate using a stack-based algorithm.
Example: 2 3 + 4 *

Prefix Expression (Polish Notation):

Operators are written before their operands.
No need for parentheses as operator precedence is implicit.
Less commonly used compared to infix and postfix.
Example: * + 2 3 4

14. Precedence

Precedence refers to the order in which operators are evaluated in an expression.
Operators with higher precedence are evaluated before those with lower precedence.
Parentheses can be used to override default precedence.
Example: In 2 + 3 * 4, the multiplication has higher precedence, so 3 * 4 is evaluated first.

15. Sortings
**Quick Sort:**
- Definition: Quick Sort is a divide-and-conquer sorting algorithm that selects a pivot element and partitions the array into two subarrays, with elements less than the pivot in one subarray and elements greater in the other. It recursively sorts the subarrays.
- Advantages: Efficient for large datasets, in-place sorting (space-efficient), faster on average compared to other sorting algorithms.
- Disadvantages: Can have poor performance on already sorted or nearly sorted data, not stable (may change the order of equal elements).
- Worst Case: O(n^2) when the pivot choice consistently leads to highly unbalanced partitions.
- Best Case: O(n log n) when the pivot choice consistently leads to balanced partitions.
- Average Case: O(n log n).

**Insertion Sort:**
- Definition: Insertion Sort is a simple sorting algorithm that builds the final sorted array one element at a time. It repeatedly takes the next element and inserts it into the correct position in the sorted part of the array.
- Advantages: Simple implementation, efficient for small datasets or partially sorted data, stable (preserves the order of equal elements).
- Disadvantages: Inefficient for large datasets, time complexity is O(n^2) for worst and average cases.
- Worst Case: O(n^2) when the array is in reverse order.
- Best Case: O(n) for already sorted data.
- Average Case: O(n^2).

**Selection Sort:**
- Definition: Selection Sort is a simple sorting algorithm that repeatedly finds the minimum element from the unsorted part of the array and swaps it with the first unsorted element.
- Advantages: Simple implementation, space-efficient (in-place sorting).
- Disadvantages: Inefficient for large datasets, not stable (may change the order of equal elements).
- Worst Case: O(n^2).
- Best Case: O(n^2).
- Average Case: O(n^2).

**Merge Sort:**
- Definition: Merge Sort is a divide-and-conquer sorting algorithm that divides the array into halves, recursively sorts them, and then merges the sorted halves to produce the final sorted array.
- Advantages: Efficient for large datasets, stable (preserves the order of equal elements), guaranteed O(n log n) time complexity.
- Disadvantages: Requires additional memory for merging the subarrays.
- Worst Case: O(n log n).

- Best Case: O(n log n).
- Average Case: O(n log n).

**Bubble Sort:**
- Definition: Bubble Sort is a simple sorting algorithm that repeatedly swaps adjacent elements if they are in the wrong order until the whole array is sorted.
- Advantages: Simple implementation.
- Disadvantages: Highly inefficient for large datasets, not stable (may change the order of equal elements).
- Worst Case: O(n^2) when the array is in reverse order.
- Best Case: O(n) for already sorted data.
- Average Case: O(n^2).

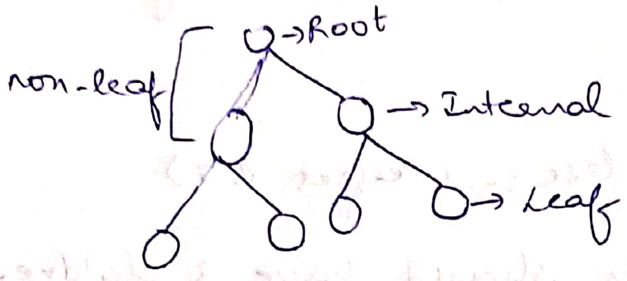
**Heap Sort:**
- Definition: Heap Sort is a comparison-based sorting algorithm that builds a max-heap (for ascending order) or min-heap (for descending order) from the array and repeatedly extracts the root element to obtain the sorted array.
- Advantages: Efficient for large datasets, in-place sorting.
- Disadvantages: Not stable (may change the order of equal elements), requires additional space for building the heap.
- Worst Case: O(n log n).
- Best Case: O(n log n).
- Average Case: O(n log n).

Remember that the actual performance of these sorting algorithms may vary depending on the specific dataset and the implementation details.

# DSA Practical - 2

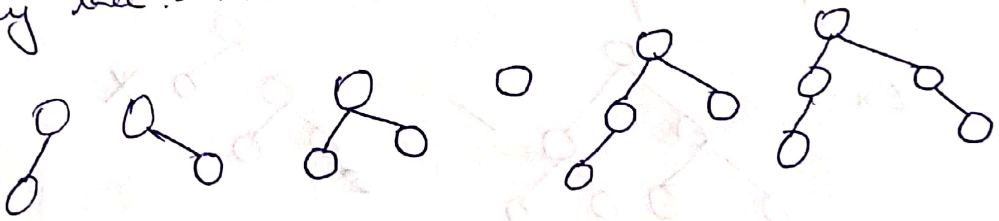
## VIVA

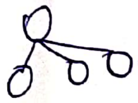
→ Tree



→ Binary tree :- Maximum 2 children for each node

Ex:-

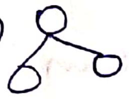




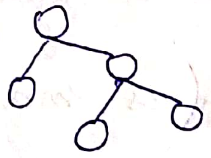
→ Not a binary tree, but a ternary tree

→ full Binary / strict binary tree :- 0 (or) 2 children

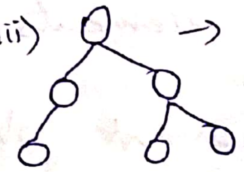
Ex:- i)



ii)



iii)

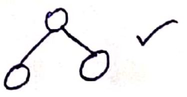


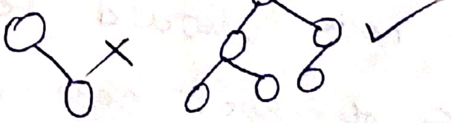
→ Binary tree but not full binary tree

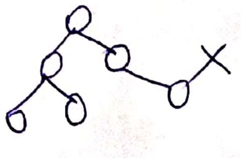
→ Almost Complete binary tree (ACBT):-

Filling of leafs should be done from left to right

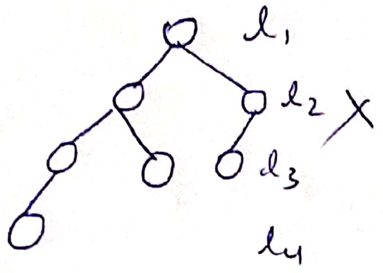
Ex:-







levels should also fill in ACBT

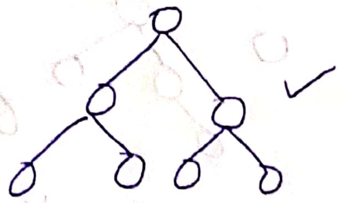


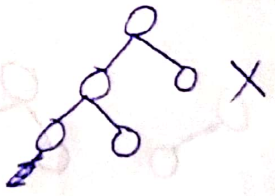
$\rightarrow$ $l_3$ is not filled completely

→ complete binary tree :- (Perfect B.T)

All the nodes should have 2 children except leaf nodes

Ex :-





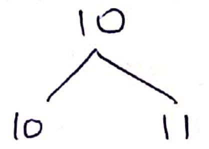
→ Binary search tree :-

↳ It a binary tree in which we put elements in a systematic order

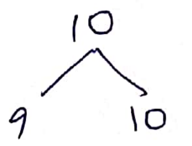
↳ The element which is smaller than the root node should be on the left side of root node.

↳ The element which is greater than root node should be on the right side of root node.

→ If the elements are equal, then you can either place it on left side of the tree or on the right side of the tree.

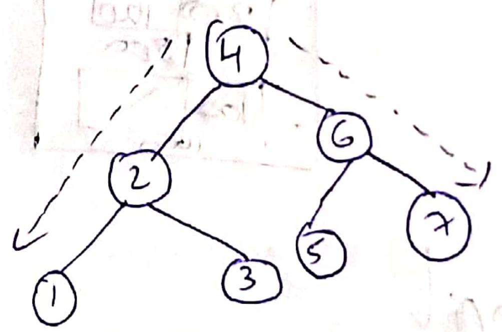
→ 

→ left - biased



→ right - biased

Ex:- 4, 2, 3, 6, 5, 7, 1, create BST with the given elements.



→ As you go on to the left side of tree you'll find the least element.

→ As you go on to the right side of tree you'll find the greatest element.

→ If you write the inorder of the BST, then ~~it will be~~ you'll get a Sorted array.