1. Array: A data structure used for storing elements of the same type in contiguous memory locations. The size of the array is fixed and allocated before program execution.

2. Linked List: A data structure consisting of a collection of nodes connected to each other. Each node contains a data field and a link or next field pointing to the next node.

3. Node: An element or individual unit of a linked list that contains data and a reference to the next node.

4. Head: The first node in a linked list.

5. Tail: The last node in a linked list that points to NULL.

6. Initialization: The process of setting up the initial state of nodes and assigning them memory.

7. Traversal: The process of accessing or displaying the contents of a linked list by moving through each node.

8. Insertion: The process of adding a new node to the linked list. It can be done at the beginning, middle, or end of the list.

9. Deletion: The process of removing an existing node from the linked list. It can be done from the beginning, end, or a specific position.

10. Search: The process of finding a specific node in the linked list by comparing its data with a given item.

11. Sorting: The process of arranging the nodes in a linked list in a specific order, such as ascending or descending.

12. Doubly Linked List: A type of linked list where each node contains a reference to both the next and previous nodes, allowing traversal in both directions.

13. Circular Linked List: A type of linked list where the last node points back to the first node, forming a circular structure.

14. Dynamic Memory Allocation: The ability of a linked list to allocate memory dynamically at runtime, allowing for flexible storage of data.

15. Advantages of Linked List: Dynamic memory allocation, easy implementation of stack and queue, reduced access time.

16. Disadvantages of Linked List: Difficult reverse traversing, sequential access, and additional memory requirement for storing pointers.

1. Doubly Linked List: A type of linked list in which each node contains a next pointer and a previous (back) pointer, allowing traversal in both directions.

1. Tree data structure: A non-primitive, nonlinear data structure that organizes data in a hierarchical design. It represents the relationship among data elements (nodes).

2. Node: Every singular element in a tree data structure is called a node.

3. Cycle: A tree data structure does not contain any cycles, which means there are no circular dependencies between nodes.

4. Binary Tree: A special data structure used for data storage purposes. Each node in a binary tree can have a maximum of two children.

5. Sub-tree: Each child of a node forms a sub-tree recursively. Each child node consists of a sub-tree with its parent node.

6. Root node: The first node in a tree is called the root node. It is the ancestor of all other nodes in the tree.

7. Edge: The connecting link between any two nodes in a tree is called an edge. A tree with several nodes has a maximum of N-1 edges.

8. Parent node: The node from which another node branches out is called the parent node. It is the ancestor of its child node(s).

9. Child node: A node that is descended from another node is called a child node. It has an edge connecting it to its parent node.

10. Siblings: Nodes that share the same parent are called siblings. They are also known as brother and sister nodes.

11. Leaf node: A node with no children is called a leaf node or external node. It does not have any child nodes.

12. Internal node: A node that has at least one child is called an internal node or non-leaf node. It is not a leaf node.

13. Degree: The total number of children a node has is called its degree. The highest degree among all nodes in a tree is called the degree of the tree.

14. Level: Each step from the root node to a node in a tree is called a level. The root node is at level 0, and the level count increases by one at each step.

15. Depth: The total number of edges from the root node to a particular node is called the depth of that node. The depth of the tree is the maximum depth among all nodes.

16. Path: The sequence of nodes and edges from one node to another node is called a path. The length of a path represents the number of nodes on that path.

17. Strictly Binary Tree: A binary tree in which each parent node contains either no children or exactly two children.

18. Complete Binary Tree: A binary tree in which all levels of the tree are filled completely, except possibly the lowest level, which is filled from the leftmost side.

19. Perfect Binary Tree: A binary tree in which all the leaf nodes are at the lowest level, and all non-leaf nodes have two child nodes.

20. Balanced Binary Tree: A binary tree in which the depths of the subtrees of all its nodes do not differ by more than 1.

21. AVL Trees: Also known as height-balanced trees, they are a type of binary search tree that maintains a balanced structure by performing rotations when necessary.

22. Binary Search Tree (BST): A binary tree in which all elements in the left subtree of a node are lesser than or equal to the contents of the node, and all elements in the right subtree are greater than the contents of the node.

1. Tree representation: A linked data structure in which each node is an object. It consists of a key field, satellite data, a pointer to the left child, and a pointer to the right child.

2. Binary search tree property: If y is in the left subtree of x, then y->key < x->key. If y is in the right subtree of x, then y->key ≥ x->key.

3. Traversing a Binary Search Tree: There are three types of tree traversal:
- Inorder tree walk: Prints keys in ascending order (left, root, right).
- Preorder tree walk: Prints root first (root, left, right).
- Postorder tree walk: Prints root last (left, right, root).

4. Searching for a Key: Given a pointer to the root of a tree and a key k, this operation searches for a node with key k and returns a pointer to that node if it exists, otherwise returns NULL.

5. Finding the Minimum in a Binary Search Tree: It aims to find the node with the minimum value in a BST by following the left child pointers from the root until a NULL is encountered.

6. Finding the Maximum in a Binary Search Tree: It aims to find the node with the maximum value in a BST by following the right child pointers from the root until a NULL is encountered.

7. Successor: The successor of a node x is the node with the smallest key greater than x->key.

8. Predecessor: The predecessor of a node x is the node with the largest key smaller than x->key.

9. Insertion: The operation of inserting a new node with a given key into a BST while maintaining the BST property.

10. Deletion: The operation of removing a node z from a BST while maintaining the BST property. There are three cases: when z has no children, when z has one child, and when z has two children.