# Relation of Algorithm and data structure

# **Search**

On each day, we search for something in our day-to-day life. Similarly, with the case of computer, huge data is stored in a computer that whenever the user asks for any data then the computer searches for that data in the memory and provides that data to the user. There are mainly two techniques available to search the data in an array:

1. Linear search
2. Binary search
3. **Linear Search:** Linear search is a very simple algorithm that starts searching for an element or a value from the beginning of an array until the required element is not found. It compares the element to be searched with all the elements in an array, if the match is found, then it returns the index of the element else it returns -1. This algorithm can be implemented on the unsorted list.
4. **Binary Search:** A Binary algorithm is the simplest algorithm that searches the element very quickly. It is used to search the element from the sorted list. The elements must be stored in sequential order or the sorted manner to implement the binary algorithm. Binary search cannot be implemented if the elements are stored in a random manner. It is used to find the middle element of the list.

# Sorting

Sorting algorithms are used to rearrange the elements in an array or a given data structure either in an ascending or descending order. The comparison operator decides the new order of the elements.

**Why do we need a sorting algorithm?**

1. An efficient sorting algorithm is required for optimizing the efficiency of other algorithms like binary search algorithm as a binary search algorithm requires an array to be sorted in a particular order, mainly in ascending order.
2. It produces information in a sorted order, which is a human-readable format.
3. Searching a particular element in a sorted list is faster than the unsorted list.

Sorting algorithm:

### Selection Sort:

### Bubble Sort

### Insertion Sort

### Merge Sort

### Quick Sort

### Heap Sort

### Counting Sort

# **Traversal Algorithms**

## Tree Traversal

Tree Traversal using Depth-First Search (DFS)

1. **Preorder Traversal (current-left-right):** Visit the current node before visiting any nodes inside the left or right subtrees. Here, the traversal is root – left child – right child. It means that the root node is traversed first then its left child and finally the right child.
2. **Inorder Traversal (left-current-right):** Visit the current node after visiting all nodes inside the left subtree but before visiting any node within the right subtree. Here, the traversal is left child – root – right child.  It means that the left child is traversed first then its root node and finally the right child.
3. **Postorder Traversal (left-right-current):** Visit the current node after visiting all the nodes of the left and right subtrees.  Here, the traversal is left child – right child – root.  It means that the left child has traversed first then the right child and finally its root node.

### Tree Traversal using Breadth-First Search (BFS)

1. Level Order Traversal:  Visit nodes level-by-level and left-to-right fashion at the same level. Here, the traversal is level-wise. It means that the most left child has traversed first and then the other children of the same level from left to right have traversed.

## Graph Traversal Algorithm

The process of visiting or updating each vertex in a graph is known as graph traversal. The sequence in which they visit the vertices is used to classify such traversals. Graph traversal is a subset of tree traversal.

There are two techniques to implement a graph traversal algorithm:

1. Breadth-first search
2. Depth-first search

Breadth-first search

There are many ways to traverse the graph, but among them, BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories - visited and non-visited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node.