**Module 2**

**DATA STRUCTURES**

**1: Arrays**

-> Ordered sequence of heterogeneous elements, can hold elements with different data types.

-> Good for Searching, inefficient in terms of insertion as compared to the linked list.

-> Fixed size.

**2: Linked List**

-> Linear data structure that stores information as a set of linearly connected nodes. Each node in the list contains data along with a pointer to another node.

Singly LL: Contains a pointer to the next node only.

Doubly LL: Contains pointers for the previous node as well as the next node.

-> Faster insertion and deletion, inefficient in terms of searching.

-> Dynamic.

**3: Stacks**

-> Stacks follow the *Last in First Out (LIFO)* ordering.

-> Last element pushed will be the top.

-> Element pushed at last will be the one to be deleted first.

**4: Queues**

-> Queue implements the FIFO method, which is short for First in First Out.

-> Element is inserted from the rear end and deleted from the front end.

-> The element that came first will be the one to be deleted first as well.

**5: Trees**

-> Trees consist of vertices (nodes) and edges that connect them.

-> Hierarchical, just like graphs only except acyclic(means a cycle can’t be formed in a tree.).

**6: Graphs**

-> A graph is a set of nodes that are connected to each other in the form of a network.

-> A collection of vertices forms a graph.

-> An edge is a link between two vertices.

-> Can be cyclic.

**7: Hashing**

-> Hashing is a process used to store an object according to a unique key. This means that hashing always creates a key-value pair. A collection of such pairs forms a *dictionary* where every object or value can be looked up according to its key. Hence, the search operation can be performed in *O(1).*

-> Hash Function: A hash function is used to transform a given key into a specific slot index. it is used to map each and every possible key into a unique slot index. If every key is mapped into a unique slot index, then the hash function is known as a perfect hash function.

**8: Heap**

-> Heaps are advanced data structures that are useful in applications where you want to sort and implement priority queues. They are regular binary trees with two special properties.

### Heaps must be Complete Binary Trees.

### The nodes must be ordered according to the Heap Order Property.

2.1 Max Heap

-> All the parent node keys must be greater than or equal to their child node keys in max-heaps. So, the root node will always contain the largest element in the heap. If Node A has a child node B, then, *key*(*A*) >= *key*(*B*).

2.2 Min Heap

-> All the parent node keys must be less than or equal to their child node keys in max-heaps. So, the root node will always contain the smallest element in the heap. If Node A has a child node B, then, *key*(*A*) <= *key*(*B*).