**Assignment 2**

1. **Explain the sorting algorithms with examples**.

Ans: Sorting Algorithms are methods of reorganizing a large number of items into some specific order such as highest to lowest, or vice-versa, or even in some alphabetical order.

These algorithms take an input list, processes it (i.e, performs some operations on it) and produce the sorted list.

The most common example we experience every day is sorting clothes or other items on an e-commerce website either by lowest-price to highest, or list by popularity, or some other order

**Types of Sorting Algorithms:**

1. Quick sort
2. Bubble sort
3. Merge sort
4. Insertion sort
5. Selection sort
6. Heap Sort
7. Radix Sort
8. Bucket Sort

**Quick sort**

The algorithm was developed by a British computer scientist Tony Hoare in 1959. The name "Quick Sort" comes from the fact that, quick sort is capable of sorting a list of data elements significantly faster (twice or thrice faster) than any of the common sorting algorithms. It is one of the most efficient sorting algorithms and is based on the splitting of an array (partition) into smaller ones and swapping (exchange) based on the comparison with 'pivot' element selected. Due to this, quick sort is also called as "Partition Exchange" sort. Like Merge sort, Quick sort also falls into the category of divide and conquer approach of problem-solving methodology.

**Bubble sort**

Bubble sort, also referred to as comparison sort, is a simple sorting algorithm that repeatedly goes through the list, compares adjacent elements and swaps them if they are in the wrong order. This is the most simplest algorithm and inefficient at the same time. Yet, it is very much necessary to learn about it as it represents the basic foundations of sorting.

**Merge Sort**

Merge sort is one of the most efficient sorting algorithms. It works on the principle of Divide and Conquer. Merge sort repeatedly breaks down a list into several sublists until each sublist consists of a single element and merging those sublists in a manner that results into a sorted list.

**Insertion Sort**

Insertion sort is the sorting mechanism where the sorted array is built having one item at a time. The array elements are compared with each other sequentially and then arranged simultaneously in some particular order. The analogy can be understood from the style we arrange a deck of cards. This sort works on the principle of inserting an element at a particular position, hence the name Insertion Sort.

**Selection sort**

Selection sort is a simple comparison-based sorting algorithm. It is in-place and needs no extra memory. The idea behind this algorithm is pretty simple. We divide the array into two parts: sorted and unsorted. The left part is sorted subarray and the right part is unsorted subarray. Initially, sorted subarray is empty and unsorted array is the complete given array.

2. **Why Are Stacks Useful?**

Ans:A stack is an example of a linear data structure. Linear data structures are collections of components arranged in a straight line. When we add or remove components of linear data structures, they grow and shrink. If we restrict the growth of a linear data structure so that new components can only be added or removed only at one end, we have a stack.

Stacks are useful data structures and are used in a variety of ways in computer science. You already know that stacks are used to implement functions and you have seen that each running program has a stack and how a program's stack grows and shrinks during calls to functions/procedures. This is especially important in understanding how recursion works.

In general, stacks are useful for processing nested structures or for functions which call other functions (or themselves). A nested structure is one that can contain instances of itself embedded within itself. For example, algebraic expressions can be nested because a subexpression of an algebraic expression can be another algebraic expression. Stacks are used to implement functions, parsers, expression evaluation, and backtracking algorithms.

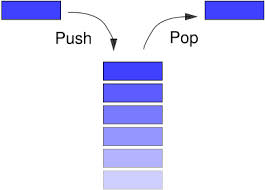
A pile of books, a stack of dinner plates, a box of pringles potato chips can all be thought of examples of stacks. The basic operating principle is that last item you put in is first item you can take out. That is, that a stack is a Last In First Out (LIFO) structure.

As an abstract entity, a stack is defined by the operations of adding items to the stack, push(), and the operation of removing items from the stack, pop(). There are a couple of other incidentals, that we need to take care of such as not push()ing an item onto a full stack and not trying to pop() items from an empty stack. This Abstract Data Type definition of a stack does not define how a stack is implemented in a computer program, it only defines the operations on a stack. In general, an Abstract Data Type (ADT) consists of a collection of data structures together with a set of operations defined on those data structures. We are already familiar with the structuring methods that C provides, like arrays, structs, and strings. To define an ADT, in addition to defining the basic data structures, we need to define a set of permissible operations on those data structures. In the case of stacks we need to define a structure (probably using struct)) to hold the data, as well as functions for manipulating the data, namely push(), pop() and two functions to check whether a stack is full() or empty().

3. **What is Stack and Queue?**

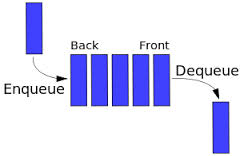
Ans: **Stack:**

In the pushdown stacks only two operations are allowed: push the item into the stack, and pop the item out of the stack. A stack is a limited access data structure - elements can be added and removed from the stack only at the top. push adds an item to the top of the stack, pop removes the item from the top. A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top.



**Queue:**

An excellent example of a queue is a line of students in the food court of the UC. New additions to a line made to the back of the queue, while removal (or serving) happens in the front. In the queue only two operations are allowed enqueue and dequeue. Enqueue means to insert an item into the back of the queue, dequeue means removing the front item. The picture demonstrates the FIFO access. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.



4**. Mention the types of Queue?**

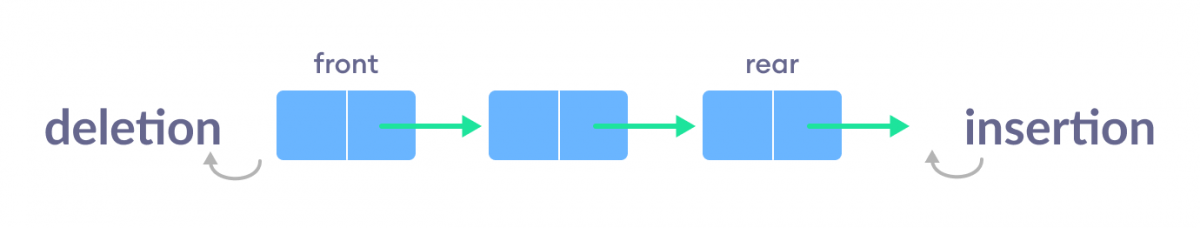
Ans: A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.

There are four different types of queues:

1. Simple Queue
2. Circular Queue
3. Priority Queue
4. Double Ended Queue

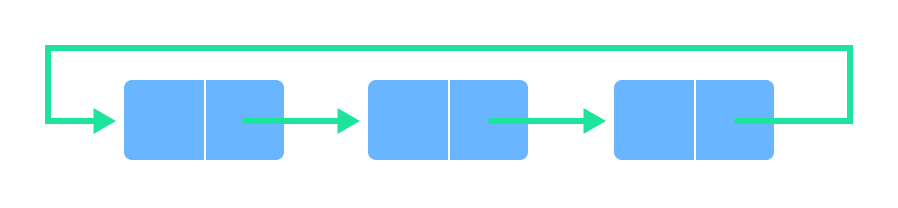
**Simple Queue**

In a simple queue, insertion takes place at the rear and removal occurs at the front. It strictly follows the FIFO (First in First out) rule.



**Circular Queue**

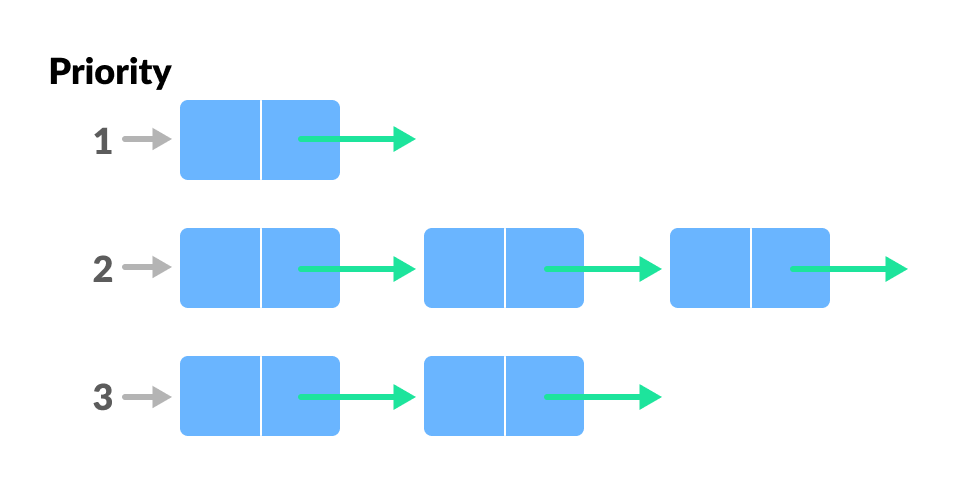
In a circular queue, the last element points to the first element making a circular link.



The main advantage of a circular queue over a simple queue is better memory utilization. If the last position is full and the first position is empty, we can insert an element in the first position. This action is not possible in a simple queue.

**Priority Queue**

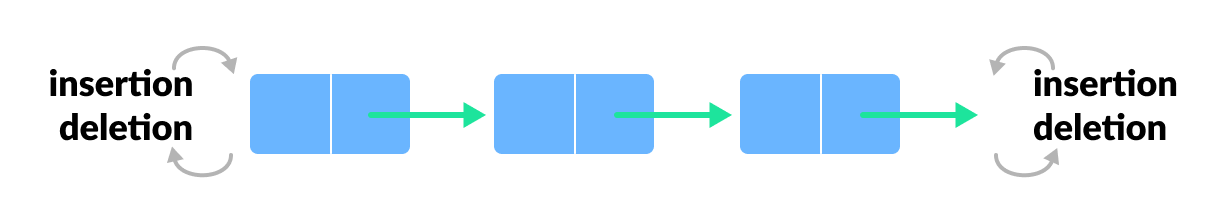
A priority queue is a special type of queue in which each element is associated with a priority and is served according to its priority. If elements with the same priority occur, they are served according to their order in the queue.



Insertion occurs based on the arrival of the values and removal occurs based on priority.

**Deque (Double Ended Queue)**

In a double ended queue, insertion and removal of elements can be performed from either from the front or rear. Thus, it does not follow the FIFO (First In First Out) rule.



1. **Why and when should I use Stack or Queue data structures instead of Arrays/Lists?**

Ans: Use a queue when you want to get things out in the order that you put them in. Use a stack when you want to get things out in the reverse order than you put them in. Use a list when you want to get anything out, regardless of when you put them in (and when you don't want them to automatically be removed).

1. **Explain why Stack is a recursive data structure**

Ans: Many programming languages implement recursion by means of stacks. Generally, whenever a function (caller) calls another function (callee) or itself as callee, the caller function transfers execution control to the callee. This transfer process may also involve some data to be passed from the caller to the callee.

This implies, the caller function has to suspend its execution temporarily and resume later when the execution control returns from the callee function. Here, the caller function needs to start exactly from the point of execution where it puts itself on hold. It also needs the exact same data values it was working on. For this purpose, an activation record (or stack frame) is created for the caller function. This activation record keeps the information about local variables, formal parameters, return address and all information passed to the caller function.