STACK AND QUEUE

Use examples to explain the sorting algorithms.

Sorting algorithms are a set of instructions that take an array or list as an input and arrange the items into a particular order.

Since sorting can often reduce the complexity of a problem, it is an important algorithm in Computer Science. These algorithms have direct applications in searching algorithms, database algorithms, divide and conquer methods, data structure algorithms, and many more.

Some Common Sorting Algorithms

Some of the most common sorting algorithms are:

Bucket Sort

Bucket Sort is a comparison sort algorithm that operates on elements by dividing them into different buckets and then sorting these buckets individually. Each bucket is sorted individually using a separate sorting algorithm or by applying the bucket sort algorithm recursively.

Bucket Sort is mainly useful when the input is uniformly distributed over a range.

Counting Sort

Counting Sort is a sorting technique based on keys between a specific range. It works by counting the number of objects having distinct key values (kind of hashing). Then doing some arithmetic to calculate the position of each object in the output sequence.

Insertion Sort

Insertion sort is a simple sorting algorithm for a small number of elements.

Example:

In Insertion sort, you compare the `key` element with the previous elements. If the previous elements are greater than the `key` element, then you move the previous element to the next position.

Heapsort

Heapsort is an efficient sorting algorithm based on the use of max/min heaps. A heap is a tree-based data structure that satisfies the heap property – that is for a max heap, the key of any node is less than or equal to the key of its parent (if it has a parent).

What Are the Benefits of Stacks?

The advantages of using stack are listed below:

Efficient data management: Stack helps you manage the data in a LIFO (last in, first out) method, which is not possible with a Linked list and array.

Efficient management of functions: When a function is called, the local variables are stored in a stack, and it is automatically destroyed once returned.

Control over memory: Stack allows you to control how memory is allocated and deallocated.

Smart memory management: Stack automatically cleans up the object.

Not easily corrupted: Stack does not get corrupted easily; hence it is more secure and reliable.

Does not allow resizing of variables: Variables cannot be resized.

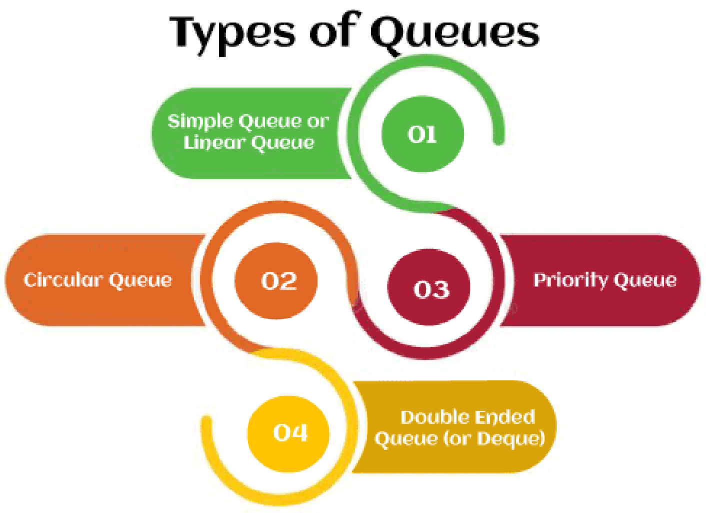
3. What is the difference between a stack and a queue?

Stacks	Queues
Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list.	Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list.
Insertion and deletion in stacks takes place only from one end of the list called the top.	Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list.
Insert operation is called push operation.	Insert operation is called enqueue operation.
Delete operation is called pop operation.	Delete operation is called dequeue operation.

What are the different forms of queues?

Types of Queue

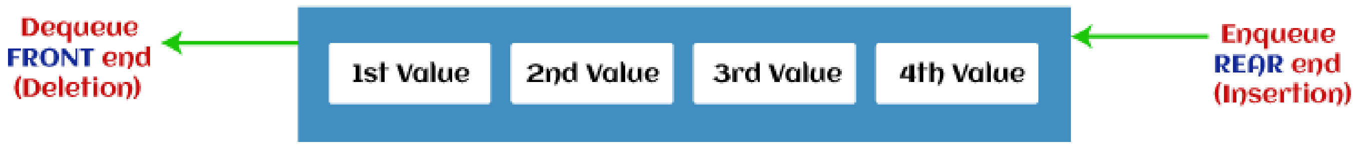
There are four different types of queue that are listed as follows -



- Simple Queue or Linear Queue
- Circular Queue
- Priority Queue
- Double Ended Queue (or Deque)

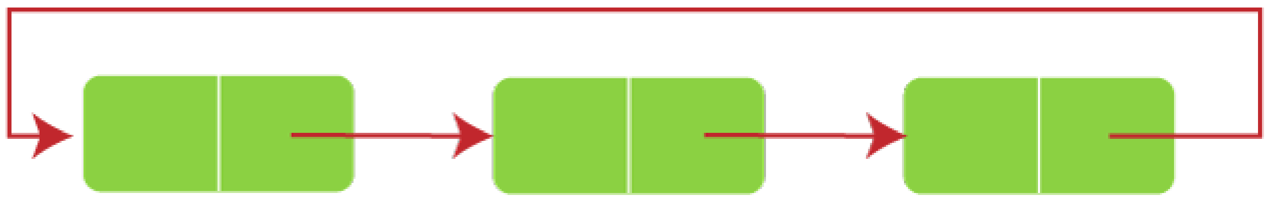
Simple Queue or Linear Queue

In Linear Queue, an insertion takes place from one end while the deletion occurs from another end. The end at which the insertion takes place is known as the rear end, and the end at which the deletion takes place is known as front end. It strictly follows the FIFO rule.



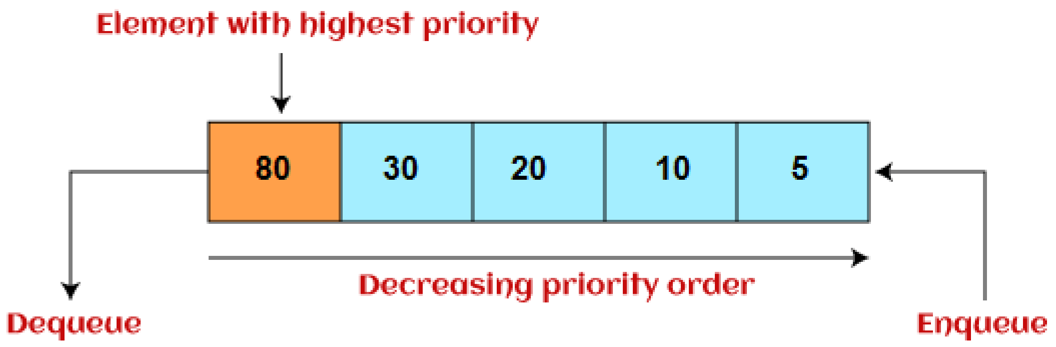
Circular Queue

In Circular Queue, all the nodes are represented as circular. It is similar to the linear Queue except that the last element of the queue is connected to the first element. It is also known as Ring Buffer, as all the ends are connected to another end. The representation of circular queue is shown in the below image -


Circular Queue

Priority Queue

It is a special type of queue in which the elements are arranged based on the priority. It is a special type of queue data structure in which every element has a priority associated with it. Suppose some elements occur with the same priority, they will be arranged according to the FIFO principle. The representation of priority queue is shown in the below image -



Why should I use Stack or Queue data structures instead of Arrays or Lists, and when should I use them?

It comes down to requirements.
Stack/Queue are ideal for enforcing sequential rules of access (LIFO & FIFO as you stated) while Array is ideal for allowing random access as desired.

Let me give some real life examples (since OOP programming often mimics real world scenarios) :
Stack : Washing of plates - the one at the top gets washed first.
Queue : Booking movie tickets (offline mode :P)
Array : Visit to a grocery store where the customer picks what he wants.

6. What is the significance of Stack being a recursive data structure?

1. "Recursion" is technique of solving any problem by calling same function again and again until some breaking (base) condition where recursion stops and it starts calculating the solution from there on. For eg. calculating factorial of a given number
2. Thus in recursion last function called needs to be completed first.
3. Now Stack is a LIFO data structure i.e. (Last In First Out) and hence it is used to implement recursion.
4. The High level Programming languages, such as Pascal , C etc. that provides support for recursion use stack for book keeping.
5. In each recursive call, there is need to save the
    1. current values of parameters,
    2. local variables and
    3. the return address (the address where the control has to return from the call).
6. Also, as a function calls to another function, first its arguments, then the return address and finally space for local variables is pushed onto the stack.
7. Recursion is extremely useful and extensively used because many problems are elegantly specified or solved in a recursive way.
8. The example of recursion as an application of stack is keeping books inside the drawer and the removing each book recursively.