1.Use examples to explain the sorting algorithms.

a)

1)Selection sort algorithm sorts an array by repeatedly finding the minimum element from unsorted part and putting it at the beginning.

Eg:

arr[] = 64 25 12 22 11

11 25 12 22 64

11 12 25 22 64

11 12 22 25 64

11 12 22 25 64

2) Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Eg:

(5 1 4 2 8) –> (1 5 4 2 8)

(1 5 4 2 8) –> (1 4 5 2 8)

(1 4 5 2 8) –> (1 4 2 5 8)

(1 4 2 5 8) –> (1 4 2 5 8)

Second Pass:

(1 4 2 5 8) –> (1 4 2 5 8)

(1 4 2 5 8) –> (1 2 4 5 8)

(1 2 4 5 8) –> (1 2 4 5 8)

3)Insertion sort is a simple sorting algorithm in which The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

12, 11, 13, 5, 6

11, 12, 13, 5, 6

11, 12, 13, 5, 6

5, 11, 12, 13, 6

5, 6, 11, 12, 13

4) Merge Sort is a Divide and Conquer algorithm. It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves.

39 27 43 3 9 82 10

39 27 43 3 9 82 10

39 27 43 3 9 82 10

39 27 43 3 9 82 10

39 27 43 3 9 82 10

3,9,10,27,39,43,82

5) Quick sort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.

arr[] = {10, 80, 30, 90, 40, 50, 70}

arr[] = {10, 80, 30, 90, 40, 50, 70}

arr[] = {10, 30, 80, 90, 40, 50, 70} // We swap 80 and 30

arr[] = {10, 30, 40, 90, 80, 50, 70} // 80 and 40 Swapped

arr[] = {10, 30, 40, 50, 80, 90, 70} // 90 and 50 Swapped

arr[] = {10, 30, 40, 50, 70, 90, 80} // 80 and 70 Swapped

2. What Are the Benefits of Stacks?

a)

1)Helps you to manage the data in a Last In First Out(LIFO) method which is not possible with Linked list and array.

2)When a function is called the local variables are stored in a stack, and it is automatically destroyed once returned.

3)A stack is used when a variable is not used outside that function.

4)It allows you to control how memory is allocated and deallocated.

5)Stack automatically cleans up the object.

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

a)

Stacks

1.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

2. Insertion and deletion in stacks takes place only from one end of the list called the top.

3. Insert operation is called push operation.

4. Delete operation is called pop operation.

5. In stacks we maintain only one pointer to access the list, called the top, which always points to the last element present in the list.

Queues

1.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.

2. 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.

3. Insert operation is called enqueue operation.

4.Delete operation is called dequeue operation.

5. In queues we maintain two pointers to access the list. The front pointer always points to the first element inserted in the list and is still present, and the rear pointer always points to the last inserted element.

4. What are the different forms of queues?

a)

There are four different types of queues:

Simple Queue

Circular Queue

Priority Queue

Double Ended Queue

5. Why should I use Stack or Queue data structures instead of Arrays or

Lists, and when should I use them?

a)

Because they help manage our data in more a particular way than arrays and lists.

Arrays and lists are random access. They are very flexible and also easily corruptible. If we want to manage your data as FIFO or LIFO it's best to use those, already implemented, collections.

More practically we should:

Use a queue when you want to get things out in the order that you put them in (FIFO)

Use a stack when you want to get things out in the reverse order than you put them in (LIFO)

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

a)

This is because the nature of nested function calling and returning is the same as that of a stack. The one that is called at top level (or the earliest) is returned the last. The later a function is called in the nested hierarchy, the earliest it returns. So, the start/ return of a function call turns out analogous to push/ pop operations of a stack and thus a stack can be used to mimic this functionality.