Q1. What is a Linked List?

A Linked List is a linear data structure which looks like a chain of nodes, where each node is a different element.
Unlike Arrays, Linked List elements are not stored at a contiguous location.

It is basically chains of nodes, each node contains information such as data and a pointer to the next node in the chain.
In the linked list there is a head pointer, which points to the first element of the linked list, and if the list is empty
then it simply points to null or nothing.

Q2. Why linked list data structure needed?

a. Ease of Insertion/Deletion: The insertion and deletion of elements are simpler than arrays since no elements need to be
shifted after insertion and deletion, Just the address needed to be updated.

b. Efficient Memory Utilization: As we know Linked List is a dynamic data structure the size increases or decreases as
per the requirement so this avoids the wastage of memory.

c. Music players: Linked lists are commonly used in music players to create playlists and play songs either from the beginning
or the end of the list. Each song in the playlist is stored as a node in the linked list, with the next pointer pointing to the next song in the list.

Q3. Types of linked lists:

1. Single LinkedList
2. Doubly LinkedList
3. Circular LinkedList

Q4. LinkedList vs Array

Array	LinkedList
1. Array are stored in contiguous location.	1. Linked List are not stored in contiguous location.
2. Fixed in size.	2. Dynamic in size.
3. Memory is allocated at compile time.	3. Memory is allocated at run time.
4. Uses less memory than LinkedList.	4. Uses more memory because it stores both data and address of next node.
5. Element can be access easily.	5. Element accessing requires the traversal of whole LinkedList.
6. Insertion and deletion operation takes time.	6. Insertion and deletion operation is faster.

Advantages –

- **Dynamic nature:** Linked lists are used for dynamic memory allocation.
- **Memory efficient:** Memory consumption of a linked list is efficient as its size can grow or shrink dynamically according to our requirements, which means effective memory utilization hence, no memory wastage.
- **Ease of Insertion and Deletion:** Insertion and deletion of nodes are easily implemented in a linked list at any position.

- **Implementation:** For the implementation of stacks and queues and for the representation of trees and graphs.
- The linked list can be expanded in constant time.

Disadvantages –

- **Memory usage:** The use of pointers is more in linked lists hence, complex and requires more memory.
- **Accessing a node:** Random access is not possible due to dynamic memory allocation.
- **Search operation costly:** Searching for an element is costly and requires O(n) time complexity.
- **Traversing in reverse order:** Traversing is more time-consuming and reverse traversing is not possible in singly linked lists.
- Searching for an element is costly and requires **O(n)** time complexity.
- Random access is not possible due to dynamic memory allocation.

Applications –

- The list of songs in the music player is linked to the previous and next songs.
- In a web browser, previous and next web page URLs are linked through the previous and next buttons.
- In the image viewer, the previous and next images are linked with the help of the previous and next buttons.
- Switching between two applications is carried out by using **"alt+tab**" in windows and "**cmd+tab**" in mac book. It requires the functionality of a circular linked list.

Q5. Why do we need linked list data structure ?

There are some important advantages to using linked lists over other linear data structures. This is unlike arrays, as they are resizable at runtime. Additionally, they can be easily inserted and deleted.

Q6. Why is a linked list preferred over an array?

- Nodes in a linked array, insertions, and deletions can be done at any point in the list at a constant time.
- Arrays are of fixed size and their size is static but Linked lists are dynamic and flexible and can expand and shrink their size.
- Linked lists provide an efficient way of storing related data and performing basic operations such as insertion, deletion, and updating of information at the cost of extra space required for storing the address.
- Insertion and deletion operations in the linked list are faster as compared to the array.
-

Q7. Why insertion/deletion are faster in a linked list?

If any element is inserted/ deleted from the array, all the other elements after it will be shifted in memory this takes a lot of time whereas manipulation in Linked List is faster because we just need to manipulate the addresses of nodes, so no bit shifting is required in memory, and it will not take that much of time.

----------Doubly LinkedList ----------

Q1. What is Doubly Linked List?

A **doubly linked list** (DLL) is a special type of linked list in which each node contains a pointer to the previous node as well as the next node of the linked list.

Advantages of Doubly Linked List over the singly linked list :

- A DLL can be traversed in both forward and backward directions.

- The delete operation in DLL is more efficient if a pointer to the node to be deleted is given.

- We can quickly insert a new node before a given node.

- In a singly linked list, to delete a node, a pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using the previous pointer.

Disadvantages of Doubly Linked List over the singly linked list:

- Every node of DLL Requires extra space for a previous pointer. It is possible to implement DLL with a single pointer though

- All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with the next pointers. For example in the following functions for insertions at different positions, we need 1 or 2 extra steps to set the previous pointer.

Applications of Doubly Linked List:

- It is used by web browsers for backward and forward navigation of web pages

- LRU (Least Recently Used) / MRU (Most Recently Used) Cache are constructed using Doubly Linked Lists.

- Used by various applications to maintain undo and redo functionalities.
    - In Operating Systems, a doubly linked list is maintained by thread scheduler to keep track of processes that are being executed at that time.

Q2. Difference between a singly and doubly linked list?

Singly linked list (SLL)	Doubly linked list (DLL)
1. SLL nodes contains 2 field data field and next link field.	1. DLL nodes contains 3 fields data field, a previous link field and a next link field.
2. In SLL, the traversal can be done using the next node link only. Thus, traversal is possible in one direction only.	2. In DLL, the traversal can be done using the previous node link or the next node link.Thus, traversal is possible in both directions (forward and backward).
3. The SLL occupies less memory than DLL as it has only 2 fields.	3. The DLL occupies more memory than SLL as it has 3 fields.
4. The Complexity of insertion and deletion at a given position is O(n).	4. The Complexity of insertion and deletion at a given position is O(n / 2) = O(n) because traversal can be made from start or from the end.
5. Complexity of deletion with a given node is O(n), because the previous node needs to be known, and traversal takes O(n)	5. Complexity of deletion with a given node is O(1) because the previous node can be accessed easily
6. A singly linked list consumes less memory as compared to the doubly linked list.	6. The doubly linked list consumes more memory as compared to the singly linked list.

Q3. Can we reverse a linked list in less than O(n) ?

No, it is not possible to reverse a linked list in less than O(n) time, where n is the number of nodes in the list. Reversing a linked list requires visiting each node at least once, which results in a time complexity of O(n).

Every node in the linked list will be visited once to reverse it, even if we transverse the list only once.

Q4. Why Quicksort is preferred for. Arrays and Merge Sort for LinkedList ?

Quick Sort is generally preferred for arrays because it has good cache locality and can be easily implemented in-place, which means it does not require any extra memory space beyond the original array.

In Quick Sort, we can use the middle element as the pivot and partition the array into two sub-arrays around the pivot. This can be done by swapping elements, and the pivot can be placed in its final position in the sorted array. This process of partitioning is done recursively until the entire array is sorted. The cache locality of Quick

Sort is beneficial because it minimizes the number of cache misses, which improves performance.

On the other hand, Merge Sort is generally preferred for linked lists because it doesn't require random access to elements.

In Merge Sort, we divide the linked list into two halves recursively until we have individual elements. Then, we merge the individual elements by comparing and linking them in a sorted order.

The advantage of Merge Sort for linked lists is that it doesn't require random access to elements, which is not efficient for linked lists since we need to traverse the list linearly. Also, Merge Sort is a stable sort,

which means it maintains the relative order of equal elements in the sorted list. This is important for linked lists, where the original order of equal elements may be significant. However, Merge Sort requires extra memory space for the merge step, which can be a disadvantage in some cases.

Q. Why is Quick Sort preferred for arrays?

Quick sort is an in-place sorting algorithm, i.e. which means it does not require any additional space, whereas Merge sort does.
Allocating and de-allocating the extra space used for merge sort increases the execution time of the algorithm.

Comparing average complexity we find that both type of sorts have O(n log n) average complexity but the constants differ. For arrays, merge sort loses due to the use of extra O(N) storage space.

Quick sort is most commonly implemented using a randomized version with anticipated time complexity of O(n log n).

Quick Sort Advantages -

1. Fast and efficient for arrays, especially for large datasets.

2. In-place sorting which means it doesn't require any extra memory space beyond the original array.

3. Easy to implement and widely used in practice.

Quick Sort Disadvantages -

1. Not stable, which means it may change the relative order of equal elements in the sorted array.

2. Not suitable for linked lists, as it requires random access to elements.

Q. Why is Merge-Sort preferred for Linked Lists?

In case of linked lists the case is different mainly due to difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory.
Unlike array, in linked list, we can insert items in the middle in O(1) extra space and O(1) time if we are given reference/pointer to the previous node.
Therefore, merge operation of merge sort can be implemented without extra space for linked lists.

In arrays, we can do random access as elements are continuous in memory Unlike arrays, we cannot do random access in linked list. Quick Sort requires a lot of this kind of access. In linked list to access i'th index, we have to travel each and every node from the head to i'th node as we do not have continuous block of memory. Therefore, the overhead increases for quick sort.

Merge sort accesses data sequentially and the need of random access is low.

Unlike arrays, in linked lists, we can insert elements in the middle in O(1) extra space and O(1) time complexities if we are given a reference/pointer to the previous node. As a result, we can implement the merge operation in the merge sort without using any additional space.

Merge Sort Advantages:

1. Suitable for sorting large datasets and linked lists.

2. Stable sort which means it maintains the relative order of equal elements in the sorted list.

3. Guaranteed worst-case time complexity of O(n*log(n)).

4. Memory efficient because it doesn't require any extra memory space beyond the original data structure.

Merge Sort Disadvantages:

1. Requires extra memory space for the merge step, which can be a disadvantage in some cases.

2. Not as efficient as Quick Sort for small datasets.

3. Not in-place sorting, which means it requires extra memory space for temporary arrays during the merging process.