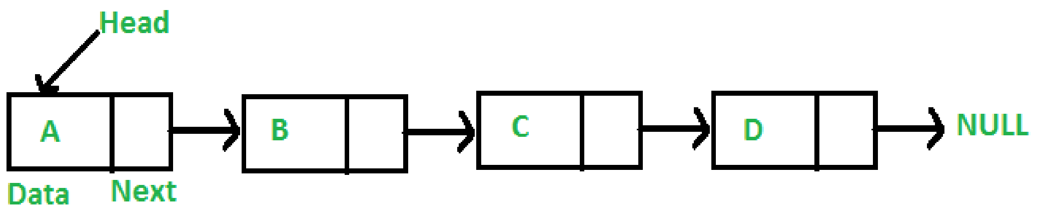
Basics of Linked List:

AGENDA:

1) **Why is linked list.**
2) **What is the difference between the array and linked list?**
3) **Advantages and Drawbacks.**
4) **Basic structures.**
5) **Types of linked list.**

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.



Why Linked List?

Arrays can be used to store linear data of similar types, but arrays have the following limitations.

1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.

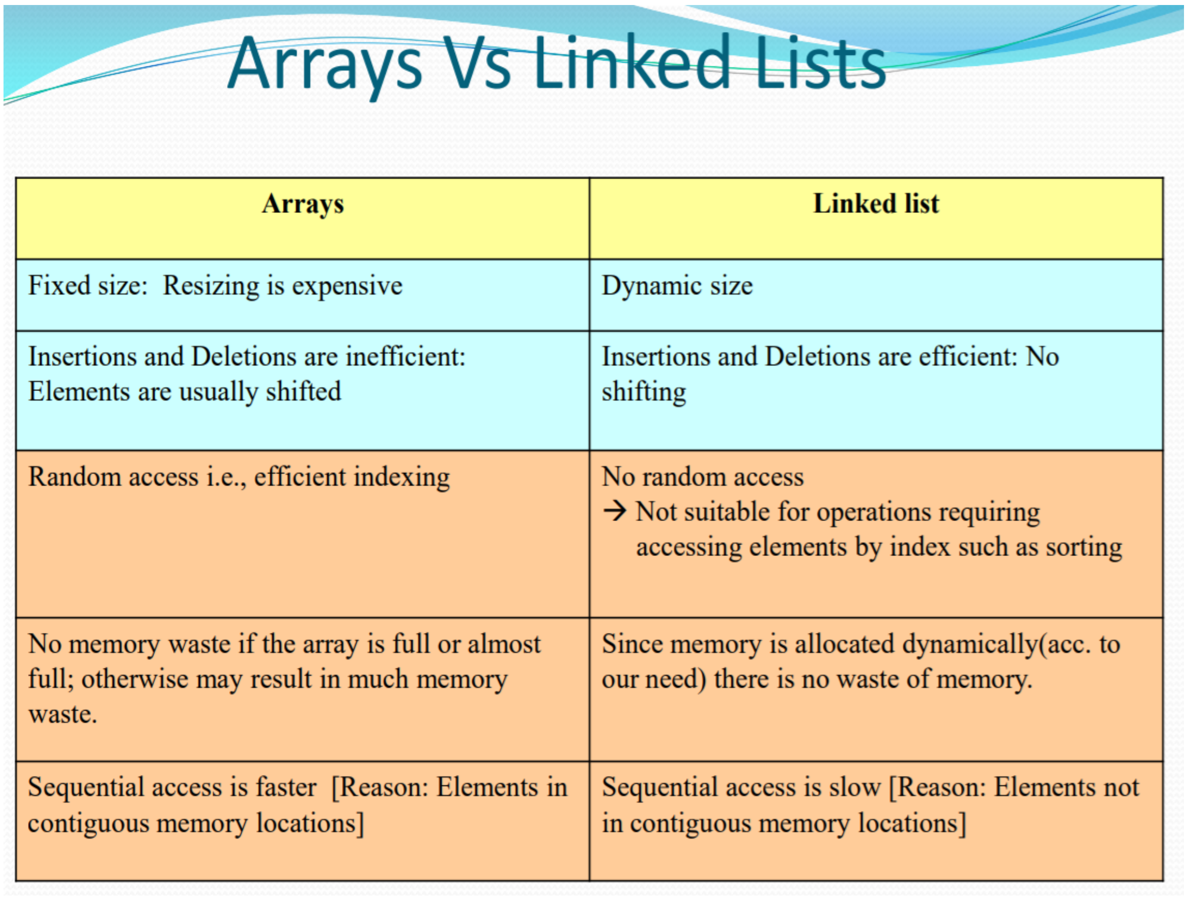
2) Inserting a new element in an array of elements is expensive because the room has to be created for the new elements and to create room existing elements have to be shifted.

For example, in a system, if we maintain a sorted list of IDs in an array id[].

id[] = [1000, 1010, 1050, 2000, 2040].

And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).

Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved.



Arrays Vs Linked Lists

Arrays	Linked list
Fixed size: Resizing is expensive	Dynamic size
Insertions and Deletions are inefficient: Elements are usually shifted	Insertions and Deletions are efficient: No shifting
Random access i.e., efficient indexing	No random access → Not suitable for operations requiring accessing elements by index such as sorting
No memory waste if the array is full or almost full; otherwise may result in much memory waste.	Since memory is allocated dynamically(acc. to our need) there is no waste of memory.
Sequential access is faster [Reason: Elements in contiguous memory locations]	Sequential access is slow [Reason: Elements not in contiguous memory locations]

Advantages over arrays

1) Dynamic size

2) Ease of insertion/deletion

Drawbacks:

1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists efficiently with its default implementation.

2) Extra memory space for a pointer is required with each element of the list.

3) Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

Representation:

A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head is NULL.

Each node in a list consists of at least two parts:

1)data

2) Pointer (Or Reference) to the next node

In C, we can represent a node using structures. Below is an example of a linked list node with integer data.

In Java or C#, LinkedList can be represented as a class and a Node as a separate class. The LinkedList class contains a reference of Node class type.

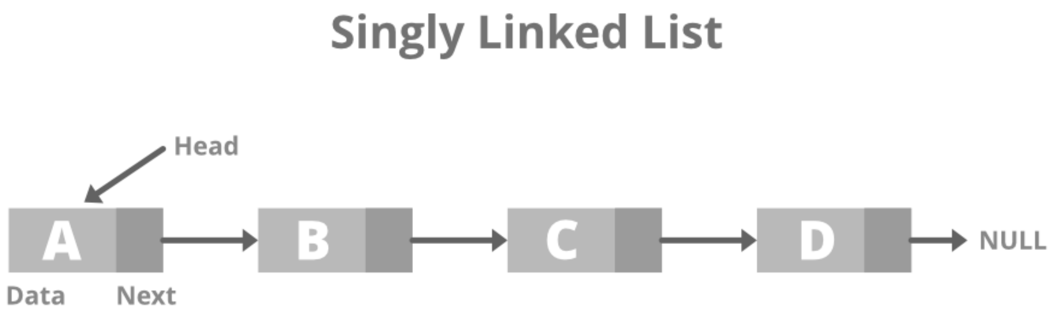
Example:

```
// A linked list node

struct Node {

    int data;

    struct Node* next;

};
```

Types of linked list:

Singly Linked List:

It is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type. The node contains a pointer to the next node means that the node stores the address of the next node in the sequence. A single linked list allows traversal of data only in one way.

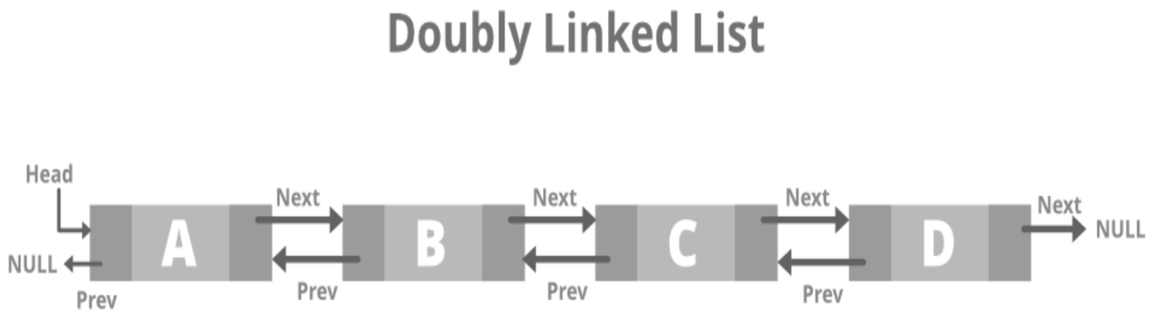


Basic structure:

```
struct Node {

   int data;

struct Node* next;

};
```

Doubly Linked List:

A doubly linked list or a two-way linked list is a more complex type of linked list which contains a pointer to the next as well as the previous node in sequence, Therefore, it contains three parts are data, a pointer to the next node, and a pointer to the previous node. This would enable us to traverse the list in the backward direction as well.

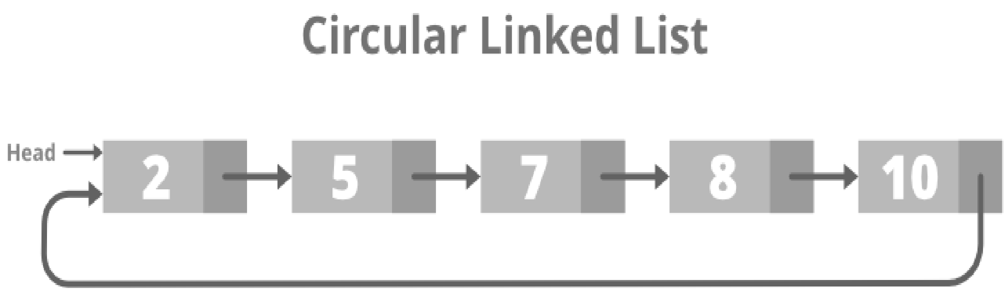


Basic Structure:

```
// Node of a doubly linked list
struct Node {
   int data;

   // Pointer to next node in DLL
   struct Node* next;

   // Pointer to the previous node in DLL
   struct Node* prev;
};
```

Circular Linked List:

A circular linked list is that in which the last node contains the pointer to the first node of the list. While traversing a circular liked list, we can begin at any node and traverse the list in any direction forward and backward until we reach the same node we started. Thus, a circular linked list has no beginning and no end. Below is the image for the same:



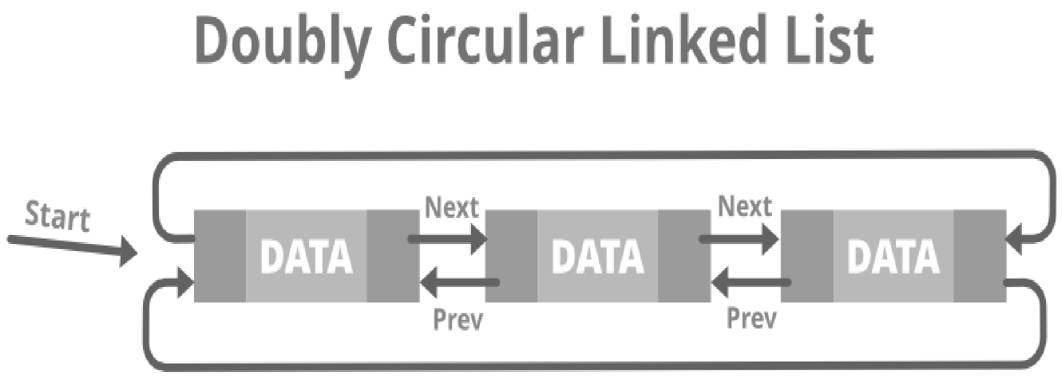
Basic Structure:

```
struct node {
   int data;
   int key;
   struct node *next;
};
```

```
};
```

Doubly Circular linked list:

A Doubly Circular linked list or a circular two-way linked list is a more complex type of linked-list that contains a pointer to the next as well as the previous node in the sequence. The difference between the doubly linked and circular doubly list is the same as that between a singly linked list and a circular linked list. The circular doubly linked list does not contain null in the previous field of the first node. Below is the image for the same:



```
// Node of doubly circular linked list

struct Node {

    int data;

    struct Node* next;

    struct Node* prev;

};
```