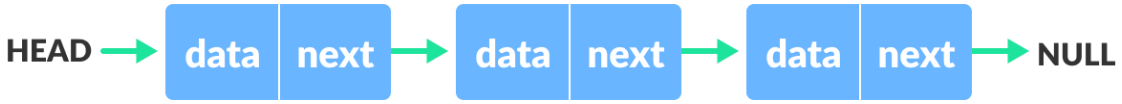
Linked list Data Structure

A linked list is a linear data structure that includes a **series of connected nodes.** Here, each node stores the **data** and the **address** of the next node. For example,



Linked list Data Structure

You have to start somewhere, so **we give the address of the first node a special name called HEAD.** Also, the **last node in the linked list can be identified because its next portion points to NULL.**

Linked lists can be of multiple types: **singly**, **doubly**, and **circular linked list**. In this article, we will focus on the **singly linked list**. To learn about other types, visit Types of Linked List.

> **Note:** You might have played the game Treasure Hunt, where each clue includes the information about the next clue. That is how the linked list operates.

Representation of Linked List

Let's see how each node of the linked list is represented. Each node consists:

- **A data items**
- **An address of another node**

Linked List Complexity

Time Complexity

	Worst case	Average Case
Search	O(n)	O(n)
Insert	O(1)	O(1)
Deletion	O(1)	O(1)

Linked List Operations: Traverse, Insert and Delete

There are various linked list operations that allow us to perform different actions on linked lists. For example, the insertion operation adds a new element to the linked list.

Here's a list of basic linked list operations that we will cover in this article.

- Traversal - **access each element of the linked list**
- Insertion - adds a new element to the linked list
- Deletion - removes the existing elements
- Search - find a node in the linked list
- Sort - sort the nodes of the linked list

Before you learn about linked list operations in detail, make sure to know about Linked List first.

Things to Remember about Linked List

- `head` points to the first node of the linked list
- `next` pointer of the last node is `NULL`, so if the next current node is `NULL`, we have reached the end of the linked list.

1. Insert at the beginning

- Allocate memory for new node
- Store data
- Change next of new node to point to head
- Change head to point to recently created node

2. Insert at the End

- Allocate memory for new node
- Store data
- Traverse to last node
- Change next of last node to recently created node

3. Insert at the Middle

- Allocate memory and store data for new node
- Traverse to node just before the required position of new node
- Change next pointers to include new node in between

Delete from a Linked List

You can delete either from the beginning, end or from a particular position.

1. Delete from beginning

- Point head to the second node

2. Delete from end

- Traverse to second last element
- Change its next pointer to null

3. Delete from middle

- Traverse to element before the element to be deleted
- Change next pointers to exclude the node from the chain

Search an Element on a Linked List

You can search an element on a linked list using a loop using the following steps. We are finding item on a linked list.

- **Make head as the current node.**
- **Run a loop until the current node is NULL because the last element points to NULL.**
- In each iteration, **check if the key of the node is equal to item.** If it the key matches the item, return true otherwise return false.

Sort Elements of a Linked List

We will use a simple sorting algorithm, Bubble Sort, to sort the elements of a linked list in ascending order below.

1. Make the head as the current node and **create another node index for later use.**
2. If head is null, return.
3. Else, run a loop till the last node (i.e., NULL).

4. In each iteration, follow the following step 5-6.
5. **Store the next node of current** in index.
6. **Check if the data of the current node is greater than the next node.** If it is greater, swap current and index.

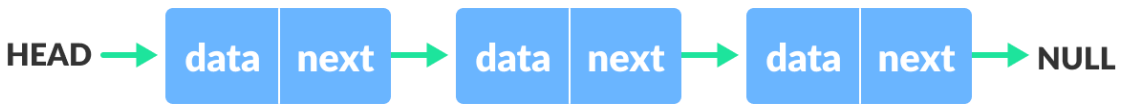
Types of Linked List - Singly linked, doubly linked and circular

Before you learn about the type of the linked list, make sure you know about the LinkedList Data Structure. There are three common types of Linked List.

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List

Singly Linked List

It is the most common. Each node has data and a pointer to the next node.

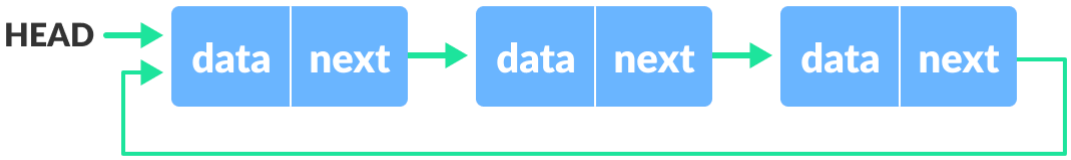


Doubly Linked List

We add a pointer to the previous node in a doubly-linked list. Thus, we can go in either direction: forward or backward.

Circular Linked List

A circular linked list is a variation of a linked list in which the last element is linked to the first element. This forms a circular loop.



Circular linked list

A circular linked list can be either singly linked or doubly linked.

- for singly linked list, next pointer of last item points to the first item
- In the doubly linked list, prev pointer of the first item points to the last item as well.