Linked List – Insertion in Linked List

1. Introduction

Insertion in a Linked List is the operation of **adding a new node** into the linked list at a specific position.

Unlike arrays, insertion in a linked list **does not require shifting elements**, making it more efficient.

Insertion is done by **updating links (pointers)** between nodes.

---

2. Why Insertion is Easy in Linked List?

In arrays:

- Elements must be shifted
- Time complexity is high

In linked lists:

- Only pointers are updated
- No shifting required
- Memory is dynamically allocated

---

3. Types of Insertion in Linked List

Insertion in a singly linked list can be done at:

1. **Insertion at the Beginning**
2. **Insertion at the End**
3. **Insertion at a Specific Position (Middle)**

---

4. Insertion at the Beginning

What Happens?

A new node is added **before the head node**.

Logic (Plain English)

1. Create a new node
2. Store data in the new node
3. Set new node’s next to current head
4. Update head to point to the new node

Example

Before:

```
Head → 10 → 20 → 30 → NULL
```

After inserting 5:

```
Head → 5 → 10 → 20 → 30 → NULL
```

---

5. Insertion at the End

What Happens?

A new node is added **after the last node**.

Logic (Plain English)

1. Create a new node
2. Store data and set next to NULL
3. Traverse the list to the last node
4. Set last node’s next to the new node

Example

Before:

```
Head → 10 → 20 → 30 → NULL
```

After inserting 40:

```
Head → 10 → 20 → 30 → 40 → NULL
```

6. Insertion at a Specific Position (Middle)

What Happens?

A new node is inserted **after a given position or node**.

Logic (Plain English)

1. Create a new node
2. Traverse the list to the desired position
3. Set new node’s next to current node’s next
4. Update current node’s next to new node

Example

Insert 25 after 20:

```
Head → 10 → 20 → 30 → NULL
```

After insertion:

```
Head → 10 → 20 → 25 → 30 → NULL
```

7. Pointer Adjustment Visualization

```
Before:
20 → 30
```

```
After:
20 → 25 → 30
```

Only **links change**, data movement is not required.

8. Time Complexity of Insertion

Type of Insertion	Time Complexity
At Beginning	O(1)
At End	O(n)
At Position	O(n)

9. Edge Cases

- Inserting into an empty list
- Inserting at position 1
- Inserting beyond list length (invalid)
- Inserting after last node

10. Advantages of Linked List Insertion

- No shifting of elements
- Efficient memory usage
- Dynamic size
- Faster than array insertion

11. Limitations

- Traversal needed for middle/end insertion
- No random access
- Pointer handling is error-prone

12. Real-World Applications

- Playlist management
- Task scheduling
- Dynamic memory allocation
- Undo/Redo systems
- Implementing stacks and queues

13. Summary

- Insertion adds a new node to the list
- Can be done at beginning, end, or middle
- Only pointers are updated
- Efficient compared to arrays
- Time complexity depends on position