**Types of Linked Lists**

**1. Singly Linked List**

* **Structure**:
  + Each node in a singly linked list contains two parts:
    - **Data**: The value stored in the node.
    - **Next**: A reference (or pointer) to the next node in the sequence.
  + The list starts with a head node and ends with a node whose next reference is null.
* **Operations**:
  + Insertion Search
  + Deletion Traversal

**2. Doubly Linked List**

* **Structure**:
  + Each node contains three parts:
    - **Data**: The value stored in the node.
    - **Next**: A reference to the next node in the sequence.
    - **Prev**: A reference to the previous node in the sequence.
  + The list has a head and often a tail
* **Operations**:
  + Insertion Search
  + Deletion Traversal

**Time Complexity of Linked Lists**

**Singly Linked List**

* Insertion at the Head: O(1)
* Insertion at the End: O(n)
* Insertion at a Specific Position: O(n)
* Deletion: O(n)
* Search: O(n)
* Traversal: O(n)

**Doubly Linked List**

* Insertion at the Head: O(1)
* Insertion at the End: O(1)
* Insertion at a Specific Position: O(n)
* Deletion: O(n)
* Traversal: O(n)

**Advantages of Linked Lists Over Arrays**

1. **Dynamic Size**:
   * **Linked Lists**: Can grow or shrink dynamically as nodes are added or removed. There’s no need to allocate a large block of memory upfront or handle resizing operations.
   * **Arrays**: Have a fixed size defined at creation. Changing the size requires creating a new array and copying elements, which can be costly.
2. **Efficient Insertions and Deletions**:
   * **Linked Lists**: Allow for efficient insertions and deletions at various positions, especially if you already have a reference to the position. This is because nodes can be added or removed without shifting other elements.
   * **Arrays**: Inserting or deleting elements, particularly in the middle of the array, requires shifting elements to maintain order, which can be inefficient.
3. **Flexible Memory Usage**:
   * **Linked Lists**: Use memory dynamically, allocating space for each node as needed. This avoids issues with memory fragmentation and unused space.
   * **Arrays**: Require a contiguous block of memory. If the array is resized, it can lead to inefficient use of memory and require time-consuming copying.
4. **No Need for Resizing**:
   * **Linked Lists**: Automatically adjust their size as nodes are added or removed, with no need for manual resizing or reallocation.
   * **Arrays**: Resizing involves creating a new array and copying elements, which can be a time-consuming operation.