**Singly Linked List:**

* **Description:** A data structure consisting of nodes, where each node contains data and a reference (or link) to the next node in the sequence. The list starts with a head node and ends with a node pointing to null.
* **Advantages:**
  + **Dynamic Size:** Can grow or shrink in size dynamically.
  + **Efficient Insertions/Deletions:** Inserting or deleting elements, particularly at the beginning, can be done in O(1) time.
* **Disadvantages:**
  + **Sequential Access:** Accessing elements by index requires traversing from the head, leading to O(n) time complexity.
  + **Extra Memory:** Requires additional memory for storing pointers.

**Doubly Linked List:**

* **Description:** Similar to a singly linked list but each node contains an additional reference to the previous node. This allows traversal in both directions.
* **Advantages:**
  + **Bidirectional Traversal:** Can be traversed forwards and backwards.
  + **Efficient Deletions:** Deleting a node when given a reference to it can be done in O(1) time since there is a back pointer.
* **Disadvantages:**
  + **Extra Memory:** Requires more memory to store an extra pointer for the previous node.
  + **Complexity:** More complex to implement and manage compared to singly linked lists.

**Analysis**

**Time Complexity:**

* **Add Operation:** O(n) (since we add at the end and need to traverse the list to find the last node)
* **Search Operation:** O(n) (since we need to traverse the list to find the matching node)
* **Traverse Operation:** O(n) (since we need to visit each node once)
* **Delete Operation:**
  + **Best Case:** O(1) (if the node to delete is the head)
  + **Worst Case:** O(n) (if the node to delete is at the end or not found)

**Advantages of Linked Lists over Arrays:**

* **Dynamic Size:** Linked lists can grow and shrink in size dynamically, which is useful when the number of elements is not known in advance.
* **Efficient Insertions/Deletions:** Inserting or deleting elements can be done efficiently without needing to shift elements, especially when working with the beginning of the list.
* **Memory Utilization:** Memory allocation for linked lists is more flexible and doesn't require contiguous memory blocks as arrays do.

**When to Use Linked Lists:**

* **Use Linked Lists When:**
  + The size of the dataset changes frequently.
  + Insertions and deletions are more common than random access.
  + You want to avoid the overhead of resizing an array.
* **Avoid Linked Lists When:**
  + Fast random access is required, as linked lists provide O(n) access time compared to O(1) for arrays.
  + Memory overhead is a concern, as each node in a linked list requires extra memory for pointers.