| Linked Lists | Dynamic Arrays |
| --- | --- |
| Access by Index: O(n)   * To access an element, you need to traverse the list from the beginning. | Access by Index: O(1)   * Direct access to any element is constant time. |
| Search: O(n)   * On average, you might need to traverse half the list to find an element. | Search: O(n)   * Similar to linked lists, you may need to check each element. |
| Insertion (at head): O(1)   * Adding an element at the beginning is a constant-time operation. | Insertion (at end): Amortized O(1)   * Typically O(1), but O(n) when resizing is necessary. |
| Insertion (at tail)     * If you don't keep a reference to the tail, you must traverse the list to the end. | Insertion (at arbitrary position): O(n)   * Requires shifting elements. |

Space complexity

| Linked Lists | Dynamic Arrays |
| --- | --- |
| Space for Data: O(n)   * Each element has a node with the data and a reference (pointer) to the next node. | Space for Data: O(n)   * Stores elements in contiguous memory. |
| Overhead per Element: O(n)   * Each element requires extra space for the pointer. | Overhead for Resizing: O(n)   * May have extra unused space until the array resizes. |

**Advantages and Disadvantages of linked list**

| **Advantages** | **Disadvantages** |
| --- | --- |
| * Dynamic Size: Easily grows and shrinks by adding or removing nodes. | * Memory Overhead: Extra memory required for storing pointers. |
| * Efficient Insertions/Deletions: Fast insertions/deletions, especially at the beginning of the list. | * Sequential Access: Slow access time; cannot perform direct indexing. |
| * Memory Utilization: Only uses as much memory as needed for the elements, without pre-allocation. | * Sequential Access: Slow access time; cannot perform direct indexing. |

**Advantages and Disadvantages of linked list**

| **Advantages** | **Disadvantages** |
| --- | --- |
| * Direct Access: Fast access time with O(1) indexing. | * Resize Overhead: Expensive resizing operation when the array needs to grow. |
| * Memory Locality: Better cache performance due to contiguous memory allocation. | * Insertion/Deletion Overhead: Costly insertions and deletions in the middle due to shifting elements. |
| * Efficient Traversal: Easier to traverse due to contiguous storage. | * Memory Allocation: May allocate more memory than necessary to accommodate future growth, leading to potentially wasted space. |