**Time Complexity**

**Linked Lists:**

Inserting at index: O(n)

Deleting at index: O(n

Printing Size: O(n)

Is empty: O(1)

Rotating right: O(n)

Reversing: O(n)

Appending : O(n)

Prepending : O(1)

Merging : O(n

Interleaving :

Get middle: O(n)

Index of: O(n)

Spliting at index: O(n)

**Dynamic Arrays:**

Inserting at index: O(n)

Deleting at index: O(n)

Getting size: O(1)

Is empty: O(1)

Rotating right: O(n)

Reversing : O(n)

Appending : O(1)

Prepending : O(n)

Merging: O(n + m)

Interleaving : O(n + m)

Get middle: O(1)

Index of: O(n)

Spliting at index: O(n)

Space Complexity

**Linked Lists:**

- Each node requires additional space for the storage of a pointer/reference.

- O(n) for n elements plus O(n) for pointers.

**Dynamic Arrays:**

- Requires a contiguous block of memory.

- O(n) for n elements.

- Additional space for resizing: can be up to O(n) (usually, the array doubles in size when capacity is reached).

Advantages and Disadvantages

**Linked Lists**:

Advantages:

Dynamic Size: Can easily grow and shrink in size by adding or removing nodes without any need to allocate or deallocate a large block of memory.

Efficient Insertions/Deletions: Insertions and deletions at the beginning or middle (once the position is known) are more efficient as no shifting of elements is required.

-Memory Utilization: More efficient in memory usage if there are many insertions and deletions because no resizing is required.

Disadvantages:

Memory Overhead: Requires extra memory for storing pointers/references.

Sequential Access: Accessing elements is slower (O(n)) as it requires traversal from the head to the desired node.

-Cache Locality: Poor cache performance due to scattered memory locations.

**Dynamic Arrays:**

Advantages:

Random Access: Provides O(1) time complexity for accessing elements.

Memory Utilization: Efficient for scenarios where the size of the array is stable or grows gradually.

Compact Memory: Stores elements in contiguous memory locations, improving cache performance.

Disadvantages:

Fixed Size: Initially requires allocation of a fixed size. Resizing (when the array grows beyond its capacity) can be expensive (O(n)).

Insertions/Deletions: Insertions and deletions, especially in the middle, require shifting of elements, leading to O(n) time complexity.

Wasted Space: Can have unused allocated memory, leading to potential wasted space.

**Comparison:**

Both linked lists and dynamic arrays are fundamental data structures used in computer science for storing collections of elements. They each have their own strengths and weaknesses, making them suitable for different types of applications. This report compares these two data structures in terms of their time complexity, space complexity, and practical advantages and disadvantages.