**Task Management System**

**Different Types of Linked Lists**

1. Singly Linked List:

- Each node contains data and a reference to the next node.

- Traversal is possible in only one direction (forward).

- Simple to implement but lacks efficient backward traversal.

2. Doubly Linked List:

- Each node contains data, a reference to the next node, and a reference to the previous node.

- Allows traversal in both directions (forward and backward).

- More complex to implement due to the additional reference.

3. Circular Linked List:

- The last node points back to the first node, forming a circle.

- Can be singly or doubly linked. Ensures no null references in the list.

- Applications requiring cyclic traversal, such as round-robin scheduling.

4. Circular Doubly Linked List:

- Combines the properties of both circular and doubly linked lists. Each node points to the next and previous nodes, with the last node pointing back to the first and the first pointing back to the last.

- Allows bidirectional traversal and cyclic properties.

- Complex data structures requiring efficient traversal in both directions with cyclic behavior.

5. Skip List:

- A layered linked list structure with additional forward links to allow faster search, insertion, and deletion.

- Balances the simplicity of linked lists with the efficiency of binary search.

- Scenarios requiring fast search times, such as databases and caching.

6. Unrolled Linked List:

- A linked list where each node contains an array of elements.

- Reduces memory overhead and increases cache performance by storing multiple elements in a single node.

-Memory-sensitive applications requiring fast traversal and storage efficiency.

7. Self-Organizing List:

- A linked list that reorders its nodes based on access patterns, such as moving accessed elements to the front.

- Optimizes search performance for frequently accessed elements.

- Caching and scenarios where access patterns are highly skewed.

**Analysis**

1. Add Task

- Time Complexity: O(n) (If adding to the end, where n is the number of nodes)

- If adding to the beginning: O(1)

2. Search Task - Time Complexity: O(n)

3. Traverse Task - Time Complexity: O(n)

4. Delete Task - Time Complexity: O(n)

**Advantages of Linked Lists Over Arrays for Dynamic Data**

1. Dynamic Size: Linked lists can grow and shrink dynamically without the need to resize, unlike arrays which have a fixed size.

2. Efficient Insertions/Deletions: Insertions and deletions in a linked list can be done in O(1) time if the position is known, whereas in arrays, it may require shifting elements, leading to O(n) time complexity.

3. Memory Utilization: Linked lists use memory more efficiently for large datasets where the size may change frequently, as they do not require contiguous memory allocation like arrays.

In conclusion, linked lists are advantageous for managing dynamic data where frequent insertions and deletions occur, and the size of the collection is not known in advance. However, they have higher overhead due to the extra memory required for storing references and are less cache-friendly compared to arrays.