**Task Management System**

**Explain the different types of linked lists (Singly Linked List, Doubly Linked List).**

**Definition:**

A **linked list** is a linear data structure where each element or node contains a reference or link to the next element in the sequence. Linked lists are used for their dynamic size and efficient insertions and deletions.

* Linked lists can grow and shrink in size by allocating and deallocating memory as needed.
* Inserting or deleting nodes does not require shifting elements, resulting in O(1) time complexity if the position is known.

**Types of Linked Lists:**

1. **Singly Linked List:**

* Each node points to the next node in the sequence.
* The last node points to null.
* Traversal is unidirectional (from head to end).

1. **Doubly Linked List:**

* Each node points to both the next and the previous nodes.
* The first node's previous pointer and the last node's next pointer point to null.
* Traversal is bidirectional (both forward and backward).

**Time Complexity Analysis:**

* 1. **Analyze the time complexity of each operation.**

This exercise implements a singly linked list for managing tasks. The time complexity analysis of each operation:

* + **addTask (O(n))**

In worst case, adding a new task requires iterating through the entire list to reach the end and append the new task. This scenario happens when the list is empty or all existing tasks have a higher task ID than the new one. However**, if the list is empty or the new task needs to be inserted at the beginning, it takes constant time (O(1)).**

* + **searchTask (O(n))**

Finding a task by its ID involves iterating through the linked list, comparing each task's ID with the provided search ID. In worst case, the search might need to visit all nodes until it finds a match.

* + **traverseTasks (O(n))**

Printing the details of all tasks requires visiting each node in the list, leading to linear time complexity.

* + **deleteTask (O(n))**

It involves searching for the target task by ID (which can take O(n) time in the worst case). Once found, removing the node involves adjusting references, which is a constant time operation O(1).

* 1. **Discuss the advantages of linked lists over arrays for dynamic data.**
* **Dynamic size:** Linked lists can grow or shrink dynamically, unlike arrays which have a fixed size.
* **Efficient insertions and deletions:** Adding or removing elements in a linked list is generally faster than in an array, especially for large datasets.
* **Memory utilization:** Linked lists don't require contiguous memory allocation, making them more efficient in terms of memory usage.
* **Flexibility:** Linked lists can be used to implement various data structures like stacks, queues, and graphs.