# Exercise 5: Task Management System

## 1. Understand Linked Lists

**Types of Linked Lists:**

**1. Singly Linked List:**

- Description: Consists of nodes where each node contains data and a reference (or pointer) to the next node. It allows unidirectional traversal.

- Advantages:

- Dynamic Size: Can easily adjust in size by adding or removing nodes without needing resizing.

- Efficient Insertions/Deletions: Adding or removing nodes is efficient if you have a reference to the node before the insertion or deletion point.

**2. Doubly Linked List:**

- Description: Consists of nodes where each node has data and two references: one to the next node and one to the previous node. It supports bidirectional traversal.

- Advantages:

- Bidirectional Traversal: Can traverse the list in both forward and backward directions.

- Efficient Deletions: Easier to delete a node since it has references to both previous and next nodes.

## 2. Setup

**Create a Class `Task`:**

- Purpose: Represents a task with attributes like task ID, name, and status.

- Methods:

- Constructor: Initializes a `Task` instance with the given ID, name, and status.

- Getters and Setters: Provide access to and modification of `taskId`, `taskName`, and `status`.

- `toString()` Method: Returns a string representation of the task.

## 3. Implementation

**Singly Linked List Implementation:**

**1. Method: `addTask(Task task)`**

- Purpose: Adds a new task to the end of the linked list.

- Functionality:

- Creates a new node with the task.

- If the list is empty, sets the new node as the head.

- Otherwise, traverses to the end of the list and appends the new node.

**2. Method: `searchTask(String taskId)`**

- Purpose: Searches for a task by its ID.

- Functionality:

- Traverses the list from the head.

- Compares each node's task ID with the given ID.

- Returns the task if found, or `null` if not found.

**3. Method: `traverseTasks()`**

- Purpose: Displays all tasks in the list.

- Functionality:

- Traverses the list from the head.

- Prints the details of each task.

**4. Method: `deleteTask(String taskId)`**

- Purpose: Deletes a task by its ID.

- Functionality:

- If the head node's task matches the ID, updates the head to the next node.

- Otherwise, traverses the list to find the node with the matching ID.

- Updates the references to remove the node from the list.

## 4. Analysis

**Time Complexity Analysis:**

**- Add Operation:**

- Worst Case: O(n) if adding at the end of the list because it requires traversing to find the last node.

- Best Case: O(1) if adding at the head of the list.

**- Search Operation:**

- Time Complexity: O(n) as it involves traversing the list to find the task.

**- Traverse Operation:**

- Time Complexity: O(n) because each node must be visited once.

**- Delete Operation:**

- Time Complexity: O(n) due to the need to find the node and update references.

**Advantages of Linked Lists over Arrays for Dynamic Data:**

- Dynamic Size: Can dynamically grow and shrink in size without needing resizing like arrays.

- Efficient Insertions/Deletions: Insertions and deletions are more efficient (O(1)) compared to arrays, where shifting elements takes O(n) time.

- Memory Utilization: Only uses memory for the elements currently in the list, avoiding unused allocated space.

**When to Use Linked Lists:**

- Frequent Insertions/Deletions: When frequent insertions or deletions are needed.

- Unknown Size: When the dataset size is not known in advance or changes frequently.

- Less Concern About Index Access: When direct index access is less crucial, as linked lists do not support O(1) access to arbitrary elements.