**ALGORITHMS DATASTRUCTURES**

**Exercise 5: Task Management System**

**Scenario:**

You are developing a task management system where tasks need to be added, deleted, and traversed efficiently.

**Steps:**

1. **Understand Linked Lists:**
   * Explain the different types of linked lists (Singly Linked List, Doubly Linked List).
2. **Setup:**
   * Create a class Task with attributes like taskId, taskName, and status.
3. **Implementation:**
   * Implement a singly linked list to manage tasks.
   * Implement methods to add, search, traverse, and delete tasks in the linked list.
4. **Analysis:**
   * Analyze the time complexity of each operation.
   * Discuss the advantages of linked lists over arrays for dynamic data.

**DIFFERENT TYPES OF LINKED LISTS:**

* **Singly Linked List:**
  + Nodes point to the next node only.
  + Efficient for insertion and deletion at the beginning or middle.
* **Doubly Linked List:**
  + Nodes have pointers to both next and previous nodes.
  + Allows backward traversal; more memory usage.

**PROGRAM:**

class Task {

int taskId;

String taskName;

String status;

Task next;

public Task(int taskId, String taskName, String status) {

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

this.next = null;

}

public String toString() {

return "ID: " + taskId + ", Name: " + taskName + ", Status: " + status;

}

}

public class TaskManagementSystem {

static Task head = null;

public static void main(String[] args) {

addTask(new Task(1, "Design UI", "Pending"));

addTask(new Task(2, "Write Backend", "In Progress"));

addTask(new Task(3, "Test Module", "Pending"));

System.out.println("All Tasks:");

traverseTasks();

System.out.println("\nSearching for Task ID 2:");

searchTask(2);

System.out.println("\nDeleting Task ID 1:");

deleteTask(1);

traverseTasks();

}

static void addTask(Task newTask) {

if (head == null) {

head = newTask;

} else {

Task current = head;

while (current.next != null) {

current = current.next;

}

current.next = newTask;

}

}

static void traverseTasks() {

Task current = head;

while (current != null) {

System.out.println(current);

current = current.next;

}

}

static void searchTask(int id) {

Task current = head;

while (current != null) {

if (current.taskId == id) {

System.out.println(current);

return;

}

current = current.next;

}

System.out.println("Task not found.");

}

static void deleteTask(int id) {

if (head == null) return;

if (head.taskId == id) {

head = head.next;

return;

}

Task current = head;

while (current.next != null) {

if (current.next.taskId == id) {

current.next = current.next.next;

return;

}

current = current.next;

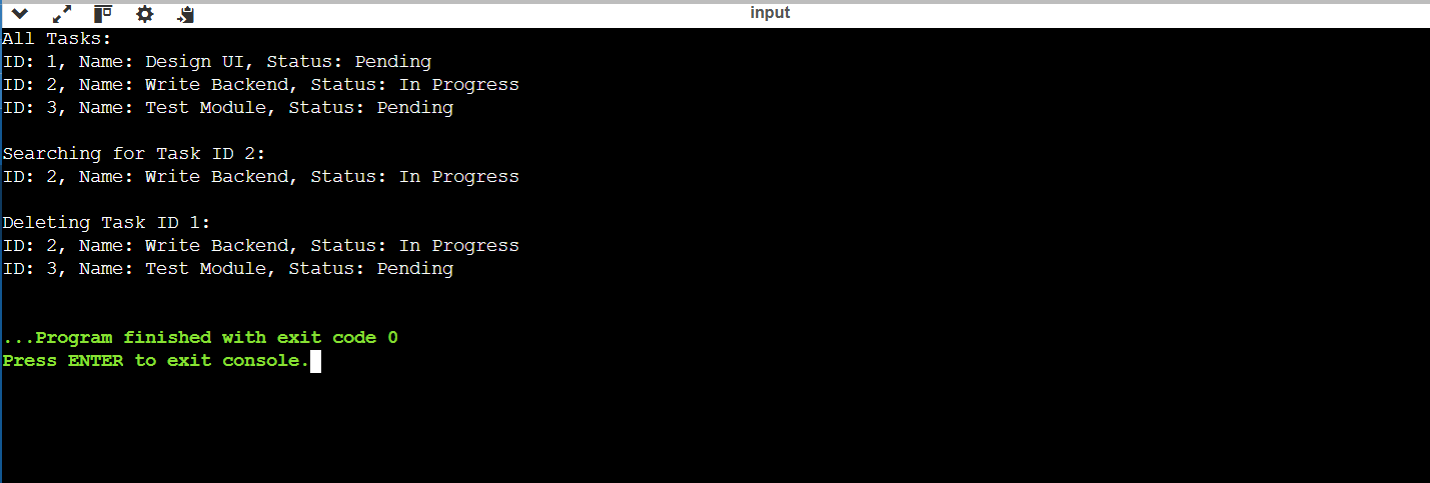
}

System.out.println("Task not found for deletion.");

}

}

**OUTPUT:**

****

**ANALYSIS:**

**Time Complexity**

* Add Task (at end): O(n)
* Search Task: O(n)
* Traverse Tasks: O(n)
* Delete Task: O(n)

**ADVANTAGES OF LINKED LIST OVER AN ARRAY:**

* **Dynamic Size:** No need to predefine size.
* **Efficient Deletion/Insertion**: Especially at the beginning or middle.
* **No Shifting Needed:** Unlike arrays, elements don't need to shift after insert/delete.
* **Memory Allocation:** Uses memory only when needed (no wasted space).