**Question 1: What is the time complexity for searching the name in the linked list you created?**

The time complexity of the searching of name function is **O(n).**

The function which starts at the head of the linked list iterates through each node until it reaches the end. If there are n nodes, this requires n steps.

**Question 2: What is the time complexity for sorting the linked list?**

Sorting the linked list will have a time complexity of **O(n*logn)**

The function iterates through the linked list to a regular Python list, which requires O(n) time to go through all the nodes. It then uses Python's built in sort method which has the time complexity of O(nlogn) and then rebuilds the linked list from the sorted list.

**Question 3: Explain the key differences between linked lists and stacks**

A stack is a data structure that uses the Last In, First Out (LIFO) principle. This means the last element added is the first one to be removed. The typical operations are:

1. **Push** (Add an element to the top of the stack)
2. **Pop** (Removes the element from the top of the stack)
3. **Peek** (View and return the top element without removing it)

A linked list is a data structure that consists of a sequence of nodes (elements) where each nodes contains a data and a reference to the next node in the sequence. The typical operations are:

1. **Insertion (**Adding a new node at the beginning, end or specific position)
2. **Deletion** (Removing a node from the list)
3. **Traversal** (Visit all the elements in the Linked list sequentially)

The key differences are:

- **Access patterns**
    - Stack enforces a strict order of operations (LIFO) and do not support traversal (only top element is accessible)
    - Linked lists supports insertion/deletion at any node position
- **Use cases**
    - Linked lists are often used for dynamic data collections that can be modified from time to time
    - Stacks are used more for function calls or algorithms that need to be backtracked