Array and Linked List Run Time

Here are the time complexities for common operations (reading, insertion, deletion, and update) for both arrays and linked lists, along with explanations:

Arrays

1. **Reading**:
   - **Time Complexity**: \(O(1)\)
   - **Explanation**: Arrays allow direct access to any element using its index. Accessing an element at a specific index is a constant-time operation because it involves a straightforward computation based on the base address of the array and the index.
2. **Insertion**:
   - **Time Complexity**: \(O(n)\) (worst case)
   - **Explanation**: Inserting an element in an array may require shifting all subsequent elements to the right to make space for the new element. In the worst case, when inserting at the beginning, all \(n\) elements need to be shifted.
3. **Deletion**:
   - **Time Complexity**: \(O(n)\) (worst case)
   - **Explanation**: Deleting an element from an array requires shifting all subsequent elements to the left to fill the gap. In the worst case, when deleting the first element, all \(n-1\) elements need to be shifted.
4. **Update**:
   - **Time Complexity**: \(O(1)\)
   - **Explanation**: Updating an element at a specific index is a constant-time operation, similar to reading, because it involves directly accessing the element by its index and changing its value.

Linked Lists

1. **Reading**:
   - **Time Complexity**: \(O(n)\)

- **Explanation**: To read an element in a linked list, you must traverse the list from the head node to the desired index, which can take up to \(n\) steps in the worst case.

2. **Insertion**:
    - **Time Complexity**: \(O(1)\) (if the position is known or at the head)
    - **Time Complexity**: \(O(n)\) (if the position is not known and you need to find it first)
    - **Explanation**: Inserting a new node at the head of the list or immediately after a known position takes constant time. However, if you need to insert at a specific position and that position is not known, you may need to traverse the list to find the position, which takes \(O(n)\) time.
3. **Deletion**:
    - **Time Complexity**: \(O(1)\) (if the position is known or at the head)
    - **Time Complexity**: \(O(n)\) (if the position is not known and you need to find it first)
    - **Explanation**: Deleting the head node or a node immediately after a known position takes constant time. However, if you need to delete a node at a specific position and that position is not known, you may need to traverse the list to find the position, which takes \(O(n)\) time.
4. **Update**:
    - **Time Complexity**: \(O(n)\)
    - **Explanation**: Similar to reading, updating an element in a linked list requires traversing the list to find the node to be updated, which can take up to \(n\) steps in the worst case. Once the node is found, updating its value is a constant-time operation.

Summary Table

Operation	Array Time Complexity	Linked List Time Complexity
Reading	\(O(1)\)	\(O(n)\)
Insertion	\(O(n)\)	\(O(1)\) or \(O(n)\)
Deletion	\(O(n)\)	\(O(1)\) or \(O(n)\)
Update	\(O(1)\)	\(O(n)\)

Key Points

- Arrays offer constant-time complexity for reading and updating but may require linear time for insertion and deletion due to the need for shifting elements.
- Linked lists offer constant-time complexity for insertion and deletion (at known positions) but require linear time for reading, updating, and locating positions for insertion and deletion.

Understanding these complexities helps in choosing the appropriate data structure based on the specific requirements of your application.