Summary

1. **What is the fundamental difference between arrays and linked lists regarding memory storage?**

   Arrays are stored in contiguous blocks of memory, meaning all elements are sequentially placed next to each other. This allows for efficient index-based access since the memory address of any element can be computed directly. In contrast, linked lists consist of nodes that are scattered throughout memory, with each node containing a value and a reference (or pointer) to the next node in the sequence. This structure enables efficient insertion and deletion operations but incurs overhead due to the pointers and lack of direct index-based access.

2. **What are the advantages and disadvantages of random and sequential access?**

   Random Access:
   Advantage: Allows immediate access to any element in the data structure, making operations like indexing very fast (O(1) time complexity).
   Disadvantage: It requires contiguous memory allocation, which can be a limitation in terms of flexibility and memory management.

   Sequential Access:
   Accesses elements in a linear order, which is slower for random access (O(n) time complexity) but allows for dynamic memory allocation and efficient insertion and deletion operations, especially in linked lists.

3. **What is the Big O notation of the insertion and deletion operations in an array and a linked list? Why is there a difference?**

   Array: Insertion and deletion have a time complexity of O(n) in the worst case, because elements need to be shifted to accommodate changes.

   Linked List: Insertion and deletion have a time complexity of O(1) if the position is known (e.g., inserting at the beginning or after a specific node). The difference arises because linked lists do not require shifting elements; only the pointers need to be updated.

4. **Describe the Selection Sort algorithm. What is its time complexity in Big O notation?**

   Selection Sort repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first unsorted element. This process continues until the entire list is sorted. The algorithm has a time complexity of O(n^2) because it involves nested loops: for each element, it scans the remaining unsorted elements to find the minimum (or maximum).

5. **How does "checking fewer elements each time" work in selection sort, and why does the time complexity remain O(n2)?**

   Although Selection Sort reduces the number of elements checked in each iteration (since the smallest elements are progressively moved to the sorted portion), the total number of comparisons still adds up to O(n^2). This is because the first pass involves n-1 comparisons, the second pass n-2, and so on, resulting in a total of (n-1) + (n-2) + ... + 1 comparisons, which sums to (n^2)/2 in the worst case.

Self-Reflection

What did you learn?

This week, I learnt data structures, particularly the differences in memory storage between arrays and linked lists, and their respective performance implications for various operations.

What went smoothly?

Understanding and implementing array and linked list operations went smoothly, as did the application of sorting algorithms to real-world-style problems.

What was difficult about the content this week?

Grasping the theoretical differences in time complexity and the practical implementation nuances between different data structures and algorithms was challenging.

How will you approach things differently next time?

I plan to tackle similar topics by breaking down complex concepts into smaller, manageable sections and reinforcing my learning with more hands-on coding exercises.

Do you have any feedback about the content for this week?

Good!