**WEEK2 HOMEWORK**

**Challenge 1 — Insert at the Front**

Q: Insert a new node at the start of a linked list. The complexity is O(1).

Discuss: It is this easier because we just change one pointer compared to inserting at index 0 in an array is different that we need to shift elements.

**Challenge 2 — Insert at the End**

Q: Append a new node to the end of a linked list. The complexity is O(n).

Discuss: Yes, we need to traverse the entire list to arrive last element(Node).

This differ from arrays that array can append more quickly in the end if having space.

Challenge 3 — Insert in the Middle Q:

Insert a node between two existing nodes.

Discuss: Two arrows we change are the previous node’s next and the new node’s next. In arrays, we must shift elements.

**Challenge 4 — Delete from the Front Q:**

Remove the first node.

Discuss: We move the head to the next node. We should delete the old first node to free its memory.

**Challenge 5 — Delete from the End**

Q: Remove the last node.

Discuss: How do we find the node before the last one? We find the node before the last one by traverse until it’s previous.

**Challenge 6 — Delete from the Middle Q:**

Remove a node between two others.

Discuss: The previous node’s next changes. If we forget to delete the node, it wastes memory

**Challenge 7 — Traverse the List Q:**

Print all elements in the linked list.

Discuss: Traversal differs from direct arr[i] access that we visit each node one by one using pointers.

**Challenge 8 — Swap Two Nodes**

Q: Swap two nodes in the list (not just their values). Discuss: Swapping values is easier because we just access to the matched elements and swap them directly.

**Challenge 9 — Search in Linked List**

Q: Search for a value in a linked list.

Discuss: How is this similar to linear search in arrays? Which one is faster for random access? It’s like linear search in arrays as it move through each element, but slower for random access since we must move through each node.

**Challenge 10 — Compare with Arrays**

Q: For each operation (insert, delete, access), write the complexity for arrays vs. linked lists.

Compare with Arrays

|  |  |  |
| --- | --- | --- |
| Operation | Array | linklist |
| Insert Front | O(n) | O(1) |
| Insert End | O(1) or O(n) | O(n) |
| Insert Middle | O(n) | O(n) |
| Delete Front | O(n) | O(1) |
| Delete Middle | O(n) | O(n) |
| Delete End | O(1) or O(n) | O(n) |

Discuss: Linked lists are better when we often insert or delete, especially at the front. Arrays are better when we need fast access using indexes.

1. Which operations were O(1) in linked lists but O(n) in arrays?  
Inserting or deleting at the front.

2.Which operation is clearly faster in arrays than in linked lists?  
Accessing elements by index.

3. Why must we manage memory carefully in linked lists?  
We must manage memory carefully in linked lists because we manually create and delete nodes, so forgetting to free memory causes leaks.

4. What does the head pointer represent?  
It represents the starting point of the linked list.

5. What happens if we lose the head pointer?  
We lose access to all nodes — the whole list is gone.

Scenario Analysis: Choose Array or Linked List

Read the following scenarios and decide whether an array or a linked list is a better fit. Justify your choice.

1. Real-time scoreboard where new scores are always added at the end and sometimes removed from the front.

Linked List, because adding and removing at ends is easy.

1. Undo/Redo feature in a text editor, where operations are frequently added and removed at the front.

Linked List, since it handles quick insert and delete at the front.

3.Music playlist that lets users add and remove songs anywhere in the list.

Linked List, because it’s flexible for insert/delete in the middle.

4.Large dataset search where random access by index is needed often.

Array, because accessing by index is faster.

5.Simulation of a queue at a bank, where customers join at the end and leave at the front.

Linked List, works well for queue operations.

6.Inventory system where you always know the item’s index and need quick lookups.

Array, because random access is instant.

7.Polynomial addition program where terms are inserted and deleted dynamically.

Linked List, since data keeps changing in size.

8.Student roll-call system where the order is fixed and access by index is frequent.

Array, simple and efficient for fixed data.