Practice Problems: Lists vs Linked Lists

Section 1: Memory and Indexes

Problem 1

A Python list starts at memory address 5000. Each element occupies 4 bytes.

- What is the memory address of index 0?
- What is the memory address of index 7?
- What is the memory address of index 15?

<details> <summary>Click for Answer</summary>

- Index 0: 5000 + (0 × 4) = **5000**
- Index 7: 5000 + (7 × 4) = 5000 + 28 = **5028**
- Index 15: 5000 + (15 × 4) = 5000 + 60 = **5060**

</details>

Problem 2

A Python list starts at memory address 2000. Each element occupies 8 bytes. You want to access the element at memory address 2056.

- What index are you accessing?

<details> <summary>Click for Answer</summary> ``` 2056 = 2000 + (index × 8) 56 = index × 8 index = 56 ÷ 8 = 7 ``` **Answer: Index 7** </details>

Problem 3

Why can't we use the same mathematical formula $\boxed{\text{start\_address} + (\text{index} \times \text{element\_size})}$ for linked lists?

<details> <summary>Click for Answer</summary>

Because linked list nodes are stored at **random, non-contiguous memory addresses**. There's no mathematical pattern between node positions and their memory locations. We must follow pointers from Head through each node sequentially.

</details>

Section 2: Time Complexity

Problem 4

Fill in the Big O time complexity for accessing elements:

Operation	Python List	Linked List
Access element at index/position 0	?	?
Access element at index/position 50	?	?
Access element at index/position n	?	?

<details> <summary>Click for Answer</summary>

Operation	Python List	Linked List
Access element at index/position 0	O(1)	O(1)
Access element at index/position 50	O(1)	O(50) = O(n)
Access element at index/position n	O(1)	O(n)

Note: For linked lists, accessing the first node (position 0) is O(1) because Head points directly to it. Any other position requires traversal.

</details>

Problem 5

You have a linked list with 1000 nodes. How many nodes must you visit to access:

- The 1st node (position 0)?
- The 500th node (position 499)?
- The last node?

<details> <summary>Click for Answer</summary>

- **1st node (position 0):** 1 node (Head points directly to it)
- **500th node (position 499):** 500 nodes (must traverse from Head through 499 nodes to reach the 500th)

- **Last node:** 1000 nodes (must traverse the entire list, OR if you have Tail pointer, just 1!)

Important note: If the linked list has a Tail pointer, accessing the last node is O(1)!

</details>

Section 3: Conceptual Understanding

Problem 6

True or False: "Linked lists don't have indexes because they're slower than Python lists."

<details> <summary>Click for Answer</summary>

False. Linked lists don't have indexes because their nodes are scattered in memory at random addresses. There's no mathematical relationship between a node's position and its memory address, making index-based access impossible. The speed difference is a *consequence* of this memory layout, not the cause.

</details>

Problem 7

Match the component to its description:

Component	Description
1. Head	A. Points to None
2. Tail	B. Contains data and pointer to next node
3. Node	C. Points to the last node
4. Last node's pointer	D. Points to the first node

<details> <summary>Click for Answer</summary>

1. Head → **D.** Points to the first node
2. Tail → **C.** Points to the last node
3. Node → **B.** Contains data and pointer to next node
4. Last node's pointer → **A.** Points to None

</details>

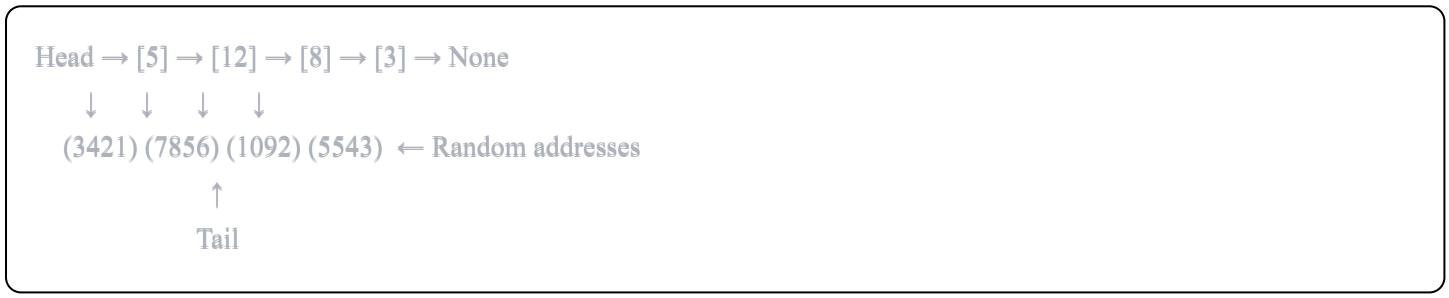
Problem 8

Draw a linked list with 4 nodes containing values [5, 12, 8, 3]. Label:

- Head
- Tail
- Each node's pointer
- None

<details> <summary>Click for Answer</summary> ``` Head → [5] → [12] → [8] → [3] → None ↑ ↑ First Last (Tail points here) ```

Or with memory addresses to emphasize scattered nature:



</details>

Section 4: Comparison Questions

Problem 9

Complete this sentence: "Python lists use _____ memory layout, which allows _____ time access by index. Linked lists use _____ memory layout, which requires _____ time access by position."

<details> <summary>Click for Answer</summary>

"Python lists use **contiguous** memory layout, which allows **O(1)/constant** time access by index. Linked lists use **scattered/non-contiguous** memory layout, which requires **O(n)/linear** time access by position."

</details>

Problem 10

You need to access the middle element of a data structure 1000 times in your program. Should you use a Python list or a linked list? Why?

<details> <summary>Click for Answer</summary>

Use a Python list.

Reasoning:

- Python list: 1000 accesses × O(1) = O(1000) = constant time for each access
- Linked list: 1000 accesses × O(n) = O(1000n) = must traverse halfway through list 1000 times

For frequent random access operations, Python lists are significantly more efficient due to their O(1) index-based access.

Note: This analysis is based on access operations only. Other operations (like insertion/deletion) might favor linked lists, which we'll learn about next!

</details>

Section 5: Challenge Problems

Problem 11

A linked list has nodes at these memory addresses: 8000, 3500, 9200, 1500, 6800.

- Can you determine the order of nodes just from these addresses?
- What additional information do you need?

<details> <summary>Click for Answer</summary>

No, you cannot determine the order from addresses alone.

Memory addresses in a linked list are random and don't indicate position. You need:

1. The **Head pointer** (which address is the first node?)
2. Each **node's "next" pointer** value (which address does each node point to?)

Example possible order using these addresses:

- Head points to 3500
- Node at 3500 points to 9200
- Node at 9200 points to 1500
- Node at 1500 points to 8000
- Node at 8000 points to 6800
- Node at 6800 points to None

Order: 3500 → 9200 → 1500 → 8000 → 6800

</details>

Problem 12

Interview Question: "Why would anyone use a linked list if Python lists can access elements in O(1) time while linked lists take O(n)?"

<details> <summary>Click for Answer</summary>

Good answer: "While Python lists have faster access times, linked lists excel at other operations that we'll learn about, such as insertion and deletion at the beginning of the list. The choice between them depends on which operations your program performs most frequently. Data structures involve trade-offs—there's no single 'best' structure for all scenarios."

Why this answer works:

- Shows you understand trade-offs
- Acknowledges you're still learning (honest)
- Hints at knowledge to come
- Demonstrates systems thinking

Note: After the next video, you'll be able to give specific examples of when linked lists outperform Python lists!

</details>

Bonus: Self-Quiz

Before moving to the next topic, make sure you can answer these without looking:

1. What are the two main memory layout differences between Python lists and linked lists?
2. What is the formula for calculating memory address from index in a Python list?
3. What are the five components of a linked list structure?
4. What is the Big O for accessing index 50 in a Python list? What about position 50 in a linked list?
5. Why is the memory layout important for understanding data structures?

Answer Key Check

If you got:

- **10-12 correct:** Excellent! You're ready for the next topic.
- **7-9 correct:** Good understanding. Review missed concepts before moving on.
- **4-6 correct:** Need more practice. Re-read the main README and retry.
- **0-3 correct:** Review the video and notes again, then retry these problems.

Next: Big O Analysis for Linked List Operations (Append, Prepend, Insert, Remove, Lookup)