# Algorithms and Data Structures Questions

1. **How would you reverse a singly linked list?**

**Good Response:** To reverse a singly linked list, you can use three pointers: prev, current, and next. Initially, prev is null, and current points to the head. In each iteration, store the next node (next = current->next), set current->next to prev, move prev to current, and move current to next. After the loop, prev becomes the new head.

**Prompt if struggling:** "Think about how you would walk through the list and reassign the pointers. How do you make sure you don’t lose track of the nodes?"

**2. What is an AVL tree, and why is it useful?**

**Good Response:** An AVL tree is a self-balancing binary search tree where the difference in height between the left and right subtrees (the balance factor) is no more than 1 for every node. It’s useful because it ensures O(log n) time complexity for insertions, deletions, and lookups by keeping the tree balanced.

**Prompt if struggling:** "Think about binary search trees. How could you ensure that their height stays balanced, so operations remain efficient?"

1. **What is the time complexity of binary search, and why?**

**Good Response:** The time complexity of binary search is O(log n) because in each step, it reduces the search space by half. This logarithmic behaviour ensures that even with large datasets, the number of comparisons remains relatively small.

**Prompt if struggling:** "Consider how binary search works by dividing the array in half each time. How does this affect the number of comparisons?"

**4. What is a heap, and what are its key properties?**

**Good Response:** A heap is a complete binary tree where each parent node is either greater than (max-heap) or less than (min-heap) its children. The key properties are that the tree is complete (every level is filled except possibly the last), and the heap property must be maintained.

**Prompt if struggling:** "Think about a tree where you can quickly get the maximum or minimum element. What does a heap enforce to keep this property?"

**5. When programming pathfinding for NPCs in games, what are the key considerations?**

**Good Response:** First, I would consider performance, especially if there are lots of entities that need to navigate the space. A\* is an efficient algorithm that is mostly commonly used so I would look to implement that first with a suitable heuristic depending on the terrain. If the world is pre-generated, then I would calculate the pathing ahead of time with waypoints. Another consideration is realism, for some entities efficient pathfinding is not necessarily suitable and a more brute-force approach would be a better fit. Finally if there are multiple NPCs then I would consider a space buffer to stop bottlenecks or unnatural movement.

**Prompt if struggling:** “How would these considerations be different in a FPS game or a 2D platformer? What can be done to stop walls or obstacles from preventing pathfinding from running successfully?”

**6. How could a hash table be used to improve performance in managing in-game assets, such as textures or sound files?**

**Good Response:** A hash table can be used to map asset names (like texture or sound file names) to their actual data in memory. This allows for quick lookups of assets by name, reducing the time it takes to load or access them during gameplay. By using a hash table, you avoid linear searches through lists of assets, improving performance when dealing with large numbers of assets.

**Prompt if struggling:** "Think about how a game might need to quickly access textures or sounds during gameplay. How could a hash table help speed up these lookups?"

**7. How would you use a linked list to manage the inventory system of a player in a game?**

**Good Response:** A linked list can be used to represent a player's inventory, where each node contains information about an item (e.g., name, description, quantity). This allows for efficient insertions and deletions, enabling players to easily add or remove items. A doubly linked list could be particularly useful, allowing for traversal in both directions, which can enhance user interfaces where players can see items and navigate back and forth.

**Prompt if struggling:** "Consider how players might add or remove items from their inventory frequently. Why would a linked list be a good choice for managing this?"

**8. What data structure would you use for implementing a playable level editor?**

**Good Response:** A stack is ideal for implementing an undo feature because it follows a Last In, First Out (LIFO) principle. Each action taken in the level editor can be pushed onto the stack. When the user wants to undo an action, the most recent action is popped off the stack and reversed. This allows for a straightforward implementation of the undo mechanism, enabling users to revert their last changes seamlessly.

**Prompt if struggling:** Why might a stack be suitable?

**9. What would be a suitable way of managing resource gathering tasks for NPCs in a survival game?**

**Good Response:** Some form of prioritisation would be needed, so a priority queue is a good idea. Implemented over a heap, priority queues have O(n log(n)) complexity. This way we can use values for urgency alongside various resources to represent time pressure or dwindling supplies and can get the most important task in constant time.

**Prompt if Struggling:** "Consider how players in a survival game might have many tasks to complete. How could a priority queue help them prioritize those tasks?"