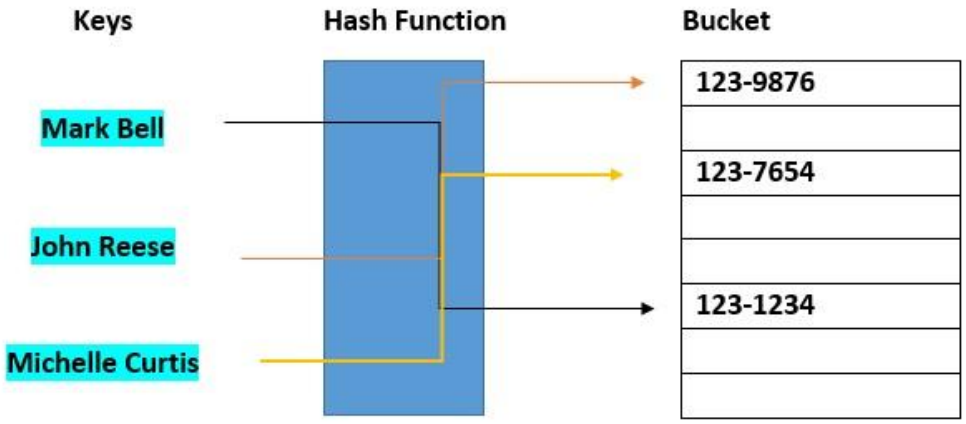
Data Structure Role-Play Guide

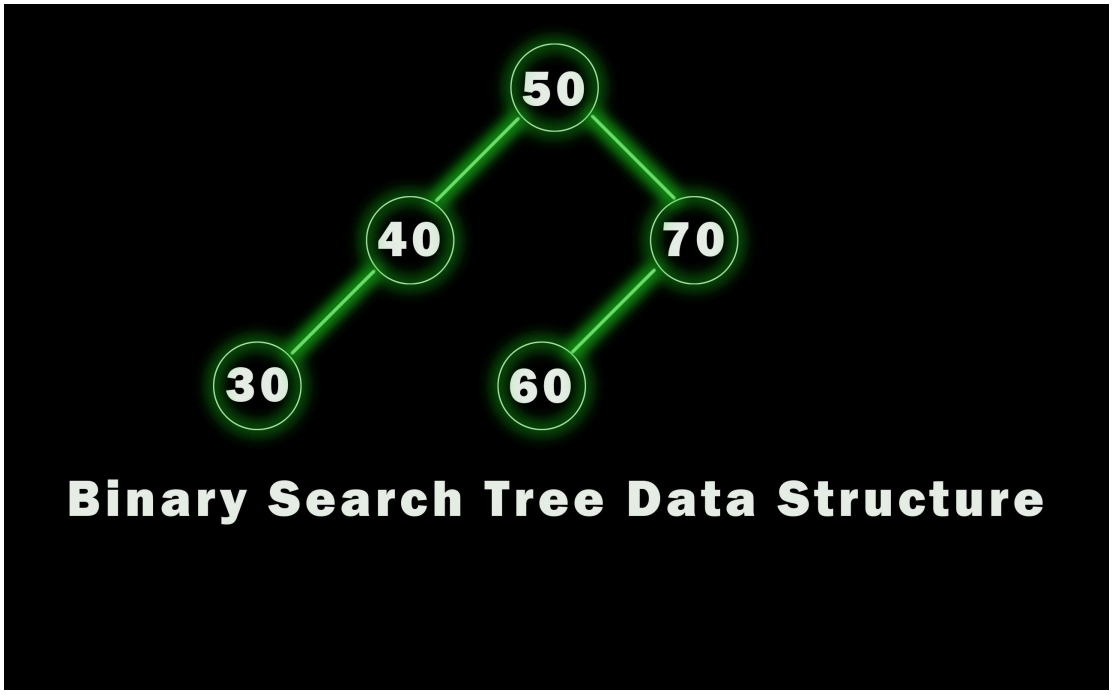
1. Hash Table

- ✧ Scenario: Fast search by ID
- ✧ Why it works: A Hash Table uses a "key" (like an ID) to calculate an exact index in memory. This allows you to jump directly to the data without checking every single item, making retrieval instantaneous (O(1) complexity on average).



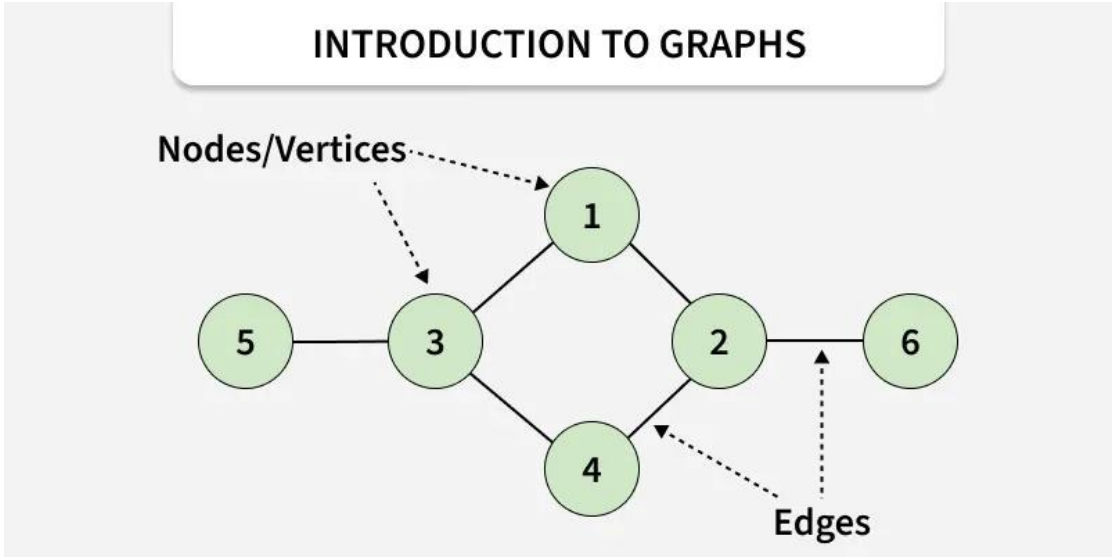
2. Tree

- ✧ Scenarios:
- ✓ Sorted Data: (specifically Binary Search Trees).
- ✓ Hierarchical Data: (like file systems or organizational charts).
- ✧ Why it works: Trees mirror parent-child relationships perfectly. For sorted data, a Binary Search Tree (BST) keeps smaller numbers on the left and larger numbers on the right, making it efficient to find or sort data.



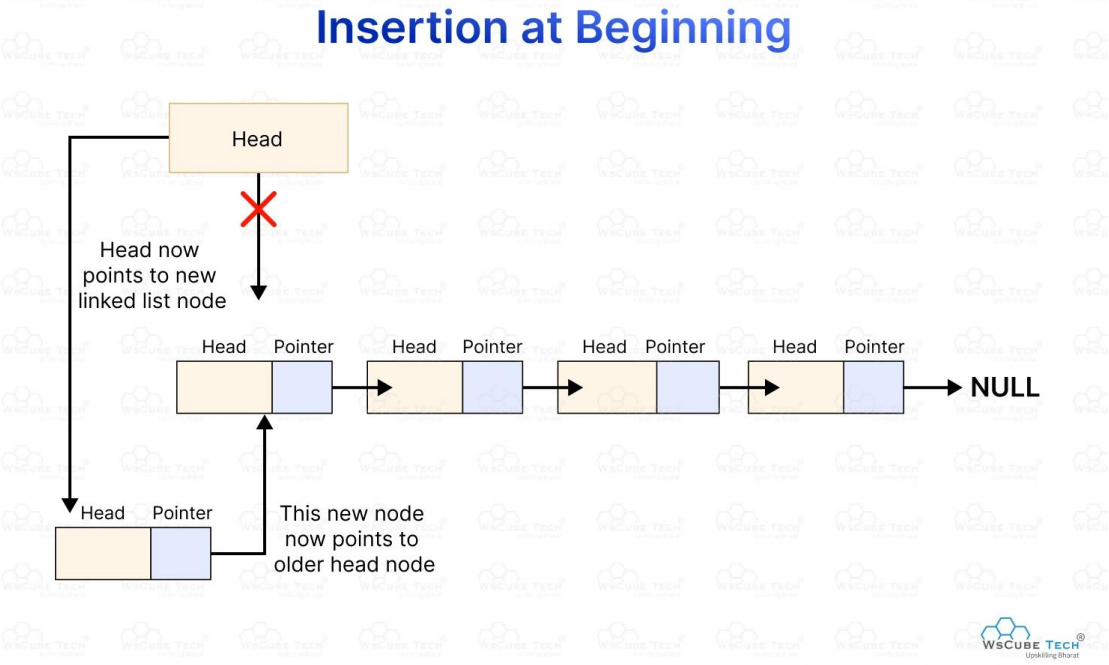
3. Graph

- ✧ Scenario: Relationships
- ✧ Why it works: Graphs are designed to represent connections (edges) between entities (nodes). This is the perfect structure for social networks (friends connecting to friends), GPS maps (cities connecting to roads), or network topologies.



4. Linked List

- Scenario: Frequent inserts
- Why it works: Unlike an array, a Linked List does not require a contiguous block of memory. To insert a new item, you simply change the "pointer" of the previous item to point to the new one. You don't have to shift all the other items down, making insertions very fast.



5. Array

- Scenario: Random access
- Why it works: Arrays are stored in a single contiguous block of memory. Because the computer knows exactly where the block starts and how big each item is, it can calculate the address of any index instantly. You can jump to Index[500] just as fast as Index[0].