**1.the complexity of searching in a balanced binary tree is** O(log (n)).

**2.What is the best-case time complexity of lookup in a map O(1).**

4.The **Rule of Three** refers to three important functions in classes that manage dynamic memory:

1. **Copy Constructor**: Creates a new object as a copy of an existing object.
2. **Assignment Operator**: Assigns one object to another of the same type.
3. **Destructor**: Cleans up when an object is no longer needed.

**Why It's Important**: If your class handles dynamic memory, you need to define all three to avoid memory leaks or issues like shallow copies, which can lead to unexpected behavior when objects share the same resources.Without defining these, your program could have memory management issues. For example, when two objects share a resource, like memory, changing one object could accidentally affect the other. A deep copy with a copy constructor ensures that each object has its own copy of the data. Without the Rule of Three, default constructors provided by the compiler will create shallow copies, which is often not what you want.

5.What is the big O of a single insertion into a binary search tree? insert=O(log n) worst case= O(n)

6.T/F: Nodes of linked lists are stored consecutively in memory

Nodes of linked lists are not stored consecutively in memory. Each node in a linked list is allocated dynamically and contains a reference (or pointer) to the next node in the list.

7.What type of traversal is used for generating a sorted list of numbers? inorder traversal

* Traversals are: **In-order:** Left → Root → Right
* **Pre-order:** Root → Left → Right
* **Post-order:** Left → Right → Root
* **Level-order:** Top to bottom, left to right by levels

8.What is the usual extension for a header filer=.h

9. What is the time complexity of a single for loop? O(n).

10. Which collision resolution strategy includes "stepping" to the next available slot in a hash table? Linear probing

11.What is the best-case time complexity of looking up an element in a hash table? O(1)

13.How do you find the first element in a queue? queue.front()

14.Given the following code what is the Big-O notation? for (int i = 0; i < size; i++){ for (int j = 0; j  
< size; j++){ doSomething() } }O(n^2)

15. T/F: Provided a picture of an AVL tree and asking the student true or false based on the validity of the tree.

1. **Binary Search Tree (BST) Property**:
   * Every node in the left subtree contains values smaller than the node.
   * Every node in the right subtree contains values larger than the node.
2. **Balance Factor Property**:
   * For every node in the tree, the balance factor (difference between the heights of the left and right subtrees) must be **-1, 0, or 1**.

16. What time complexity should have the fastest run time O(log(n))

1.**Why can a binary tree be more efficient than a linked list?** **Efficiency in Search:** Binary trees (e.g., binary search trees) allow **O(log(n))** search, insert, and delete operations if balanced, compared to O(n) for a linked list.**Hierarchical Structure:** Binary trees organize data hierarchically, enabling faster lookups.

**2.What is a binary search tree and why use one?Definition:** A binary search tree (BST) is a tree where each node has at most two children:Left subtree: Contains nodes with keys less than the node's key.Right subtree: Contains nodes with keys greater than the node's key.**Use Case:** Efficient searching, insertion, and deletion in O(log(n)) for balanced BSTs. Commonly used in databases and file systems.

**3.Given an AVL tree, which ways can you rotate it to balance it?**

**Left Rotation:** Performed when a right-right imbalance occurs.

**Right Rotation:** Performed when a left-left imbalance occurs.

**Left-Right Rotation:** Performed when a left-right imbalance occurs (left rotation on the child, then right rotation on the parent).**Right-Left Rotation:** Performed when a right-left imbalance occurs (right rotation on the child, then left rotation on the parent).

**4.What's the difference between a heap and an AVL Tree?**

a heap prioritizes the order between parent and child nodes based on their values (either always larger or smaller), while an AVL tree prioritizes maintaining a balanced structure by ensuring the height difference between left and right subtrees of each node is never more than one, allowing for efficient searching and retrieval based on key values; essentially, a heap cares about hierarchical ordering, while an AVL tree cares about relative ordering between siblings.

**5.how does separate chaining work:**

Separate chaining resolves hash collisions by maintaining a **linked list** at each index of the hash table.

**6.Linked list vs vector: Linked lists are better suited for Frequent insertions and deletions. Dynamic memory allocation (no need to declare size upfront). Vectors are better when: Fast random access (via index) is needed. Contiguous memory is beneficial for performance.**

**What is a “Bin” in a Hash Table and What Are the Pros/Cons of This Collision Resolution Method Relative to Other Collision Resolution Methods?**

**Bin in a Hash Table:**

**In the context of a hash table, a bin is a slot or bucket in which elements are stored. Each bin corresponds to a hash value and can hold multiple items that hash to the same index (this is known as a collision). Typically, the elements in each bin are stored in a linked list or another data structure.**

**8.What is a use case of recursion:** Tree traversal (pre-order, in-order, post-order) or solving the N-Queens problem.

**9.**hash table could be better than a vectore: **Use Case:** Storing and retrieving data with non-sequential keys, such as a phonebook or dictionary. Hash tables are better for fast lookups better than a vector.

**10.Why do you need a base case for a recursive function? Prevents Infinite Recursion:** Without a base case, the recursion would never terminate, leading to a stack overflow. For calculating the factorial of a number n!, the function factorial(n) calls factorial(n-1) recursively until it reaches the base case factorial(0) = 1.

**In your own words, explain the differences between Linear Probing and Separate Chaining.**

**Linear Probing:**

* **How It Works:** If a hash collision occurs, the next available slot is checked (in sequence).
* **Advantages:** Simple to implement and avoids additional memory allocation.
* **Disadvantages:** Prone to clustering, which can degrade performance.

**Separate Chaining:**

* **How It Works:** Each index in the hash table maintains a linked list to store all elements that hash to the same index.
* **Advantages:** Reduces clustering and can handle more elements than the table size.
* **Disadvantages:** Additional memory overhead for pointers and potential slower lookups due to traversal of the linked list.

**1. Like-Love-Meh: Maps, Hash Tables, Binary Trees**

Maps (Love):

Maps are great because they allow you to store data in key-value pairs, which makes it easy to quickly retrieve values using a unique key. This structure is really efficient for searching because, unlike lists or arrays that may require scanning through each element, maps use algorithms that provide faster lookups, often in constant time (O(1)). This

makes maps especially useful when you have a large dataset and need to access data quickly. Additionally, they can store different types of values associated with each key, offering flexibility and organization in your data management.

I like hash tables too because they are really fast for searching, inserting, and deleting (O(1) on average). The way they work is that they hash the key to an index, which makes looking up values quick. But, if there are too many collisions, they can slow down, so it’s important to have a good hash function.

Binary Trees (Meh):

I’m not as big a fan of binary trees. They are useful for sorting and searching, but unless they are balanced (like in AVL or Red-Black trees), the performance can get bad (O(n) time complexity in the worst case). For general use, I’d prefer hash tables since they are usually faster for lookups.

**In Your Own Words, Describe the Difference Between Two Data Structures**

• Array vs. Linked List:

An array is a collection of elements stored in contiguous memory locations. It allows for constant-time access to any element by its index (O(1)), but inserting or removing elements can be inefficient (O(n)) due to the need for shifting elements.

• A linked list, on the other hand, consists of nodes where each node contains data and a pointer to the next node. It allows for efficient insertions and deletions (O(1)) because no elements need to be shifted. However, accessing elements requires O(n) time, as you have to traverse the list from the head.

**Benefits of Implementing a Hash Table Compared to a Linked List**

**• Hash Table:**

**Hash tables provide faster access times with an average case of O(1) for insertion, deletion, and lookup operations. This is much faster than the O(n) time complexity of linked lists for searching, since a hash table maps keys to values directly via a hash function.**

**• Linked List:**

**Linked lists are efficient for scenarios where frequent insertions and deletions are required in the middle of the collection. However, they do not provide fast access to individual elements compared to hash tables**

**Given a Singly Linked List, Write Pseudo Code (or C++) to Reverse It**

**Node\* reverseList(Node\* head) {**

**Node\* prev = nullptr; // `prev` will track the previous node, initially null.**

**Node\* current = head; // `current` starts at the head of the list.**

**Node\* next = nullptr; // `next` will temporarily hold the next node.**

**while (current != nullptr) {**

**next = current->next; // Store the next node.**

**current->next = prev; // Reverse the link direction.**

**prev = current; // Move `prev` forward to the current node.**

**current = next; // Move `current` forward to the next node.**

**}**

**return prev; // After the loop, `prev` will be the new head.**

**}**

Linear Probing is a collision resolution technique used in hash tables. When a collision occurs (two keys hash to the same index), linear probing checks the next consecutive index until it finds an empty slot.

Example:

Suppose we have a hash table of size 10 and we insert the values 12, 22, and 32. Using the hash function hash(x) = x % 10, all three values will hash to index 2.

o Insert 12 at index 2.

o Insert 22 at index 2, but it’s occupied, so move to index 3.

o Insert 32 at index 2, but it’s occupied, so move to index 4

Explain How to Fix Zig-Zig and Zig-Zag Imbalance in AVL Trees

Zig-Zig:A zig-zig imbalance occurs when both rotations are in the same direction (either left-left or right-right). To fix it, you perform a single rotation (left or right) at the root of the subtree.

Using the Rule of Three, Make a .h File and Include Two Functions and Two Variables

class MyClass {

private:

int\* data;

public:

MyClass(int value) : data(new int(value)) {}

MyClass(const MyClass& other) : data(new int(\*other.data)) {} // Copy Constructor

MyClass& operator=(const MyClass& other) { // Copy Assignment Operator

if (this != &other) {

delete data;

data = new int(\*other.data);

}

return \*this;

}

~MyClass() { delete data; } // Destructor

void setData(int value) { \*data = value; }

int getData() { return \*data; }

};

Describe the Components that Make Up a Binary Tree

A binary tree consists of:

• Root: The topmost node.

• Nodes: Each node has a data element and two children (left and right).

• Leaf Nodes: Nodes with no children.

• Subtrees: The left and right subtrees rooted at any node.

Who Would Win in a Battle to the Death, Stacks, Deques, or Queues?

Deques would win because they offer the most flexibility and can handle both stack-like and queue-like operations. Whether you need to add/remove from the front or back,

deques can do it all.

Simple Coding Interview Question:

Question:Write a function that takes a string as input and returns the first non-repeating character in the string. If no such character exists, return null.

Data Structure Best for the Solution:

• Hash Map (or Dictionary):A hash map is ideal for this problem because it allows us to store the frequency of each character as we iterate through the string.

Using a hash map, we can quickly check the frequency of each character and find the first one with a frequency of 1.The time complexity of this approach is O(n),

where n is the length of the string, because we only need to traverse the string once to build the frequency map, and checking the map for each character is an O(1) operation.