CS301
Data Structures

Important subjective

Lec 1 - Introduction to Data Structures

1. **What is a data structure?** Answer: A data structure is a way of organizing and storing data in a computer memory so that it can be accessed and manipulated efficiently.
2. **What is the difference between an array and a linked list?** Answer: An array is a fixed-size data structure that stores elements in contiguous memory locations, while a linked list is a dynamic data structure that stores elements in nodes, with each node pointing to the next one.
3. **What is a stack?** Answer: A stack is a data structure that follows the "last-in-first-out" (LIFO) principle, where the last element added is the first one to be removed.
4. **What is a queue?** Answer: A queue is a data structure that follows the "first-in-first-out" (FIFO) principle, where the first element added is the first one to be removed.
5. **What is a tree?** Answer: A tree is a hierarchical data structure that consists of nodes connected by edges, with one node at the top called the root node.
6. **What is a graph?** Answer: A graph is a non-linear data structure that consists of nodes connected by edges, where the edges may be directed or undirected.
7. **What is a hash table?** Answer: A hash table is a data structure that uses a hash function to map keys to their corresponding values, allowing for efficient insertion, deletion, and retrieval operations.
8. **What is a binary search tree?** Answer: A binary search tree is a binary tree where the left subtree of each node contains only elements smaller than the node, and the right subtree contains only elements larger than the node.
9. **What is a heap?** Answer: A heap is a binary tree where each parent node has a value that is greater than or equal to (for a max heap) or less than or equal to (for a min heap) its children.
10. **What is a priority queue?** Answer: A priority queue is a data structure that stores elements with associated priorities, where elements with higher priorities are dequeued first.

Lec 2 - List Implementation

1. **What is an array-based list?**
   Answer: An array-based list is a data structure that stores a collection of elements in a contiguous block of memory, where each element can be accessed using an index.

2. **What is a linked-list based list?**
   Answer: A linked-list based list is a data structure that stores a collection of elements as nodes, where each node contains an element and a reference to the next node in the list.

3. **What is the difference between an array-based list and a linked-list based list?**
   Answer: An array-based list uses a fixed-size array to store the elements, while a linked-list based list uses a dynamic data structure composed of nodes. The main difference is that arrays offer efficient random access to elements, while linked lists offer efficient insertion and deletion operations.

4. **What is the time complexity of accessing an element in an array-based list?**
   Answer: The time complexity of accessing an element in an array-based list is O(1).

5. **What is the time complexity of accessing an element in a linked-list based list?**
   Answer: The time complexity of accessing an element in a linked-list based list is O(n).

6. **What is the advantage of using a linked-list based list over an array-based list?**
   Answer: The main advantage of using a linked-list based list is that it offers efficient insertion and deletion operations, which can be expensive in an array-based list.

7. **What is the disadvantage of using a linked-list based list over an array-based list?**
   Answer: The main disadvantage of using a linked-list based list is that it offers inefficient random access to elements, which can be expensive in an array-based list.

8. **What is dynamic resizing of a list?**
   Answer: Dynamic resizing of a list refers to the ability of a list to grow or shrink in size as elements are added or removed.

9. **How is dynamic resizing achieved in an array-based list?**
   Answer: Dynamic resizing in an array-based list is achieved by allocating a new, larger array when the existing array becomes full, and copying the elements from the old array to the new array.

10. **How is dynamic resizing achieved in a linked-list based list?**
    Answer: Dynamic resizing in a linked-list based list is achieved by allocating new nodes as needed and updating the references between nodes.

Lec 3 - Linked List inside Computer Memory

1. **What is a linked list and how is it different from an array?**
   Answer: A linked list is a data structure where each element (node) contains a value and a reference to the next node. The first node in the list is called the head, and each subsequent node is linked to the previous node. In contrast, an array stores a fixed number of elements of the same type in contiguous memory locations.

2. **How are nodes in a linked list allocated in memory?**
   Answer: Each node in a linked list is typically represented as a block of memory that contains the value and a pointer to the next node. The head node is stored in a variable, and each subsequent node is allocated dynamically as needed.

3. **What is the time complexity of inserting a node at the beginning of a linked list?**
   Answer: The time complexity of inserting a node at the beginning of a linked list is O(1), as it involves updating the head node pointer to point to the new node.

4. **How do you traverse a linked list?**
   Answer: To traverse a linked list, start at the head node and follow the next node pointers until the end of the list is reached.

5. **What is the difference between a singly linked list and a doubly linked list?**
   Answer: In a singly linked list, each node contains a reference to the next node, while in a doubly linked list, each node contains references to both the next and previous nodes.

6. **What is the time complexity of inserting a node at the end of a linked list?**
   Answer: The time complexity of inserting a node at the end of a linked list is O(n), as it involves traversing the list to find the last node and updating its next node pointer to point to the new node.

7. **How do you delete a node from a linked list?**
   Answer: To delete a node from a linked list, update the previous node's next node pointer to point to the next node, effectively removing the node from the list.

8. **What is a circular linked list?**
   Answer: A circular linked list is a linked list where the last node's next node pointer points to the head node, creating a circular structure.

9. **What is a sentinel node in a linked list?**
   Answer: A sentinel node is a special node added to the beginning or end of a linked list that acts as a marker to indicate the start or end of the list.

10. **What is the space complexity of a linked list?**
    Answer: The space complexity of a linked list is O(n), where n is the number of nodes in the list. This is because each node requires its own block of memory.

Lec 4 - Methods of Linked List

1. **What is a Linked List?**
   Answer: A Linked List is a linear data structure that consists of a sequence of nodes, where each node contains data and a pointer to the next (and possibly the previous) node in the list.

2. **What is the difference between a singly linked list and a doubly linked list?**
   Answer: In a singly linked list, each node has a pointer to the next node in the list, while in a doubly linked list, each node has a pointer to both the next and previous nodes in the list.

3. **What is a head pointer and a tail pointer in a Linked List?**
   Answer: The head pointer points to the first node in the list, while the tail pointer points to the last node in the list.

4. **How is a new node inserted at the beginning of a singly linked list?**
   Answer: To insert a new node at the beginning of a singly linked list, a new node is created and its next pointer is set to the current head of the list. The head pointer is then updated to point to the new node.

5. **How is a new node inserted at the end of a singly linked list?**
   Answer: To insert a new node at the end of a singly linked list, a new node is created and its next pointer is set to NULL. The next pointer of the current last node is updated to point to the new node, and the tail pointer is updated to point to the new node.

6. **How is a node deleted from a singly linked list?**
   Answer: To delete a node from a singly linked list, the next pointer of the previous node is updated to point to the next node in the list. The memory occupied by the deleted node is then freed.

7. **What is a sentinel node in a Linked List?**
   Answer: A sentinel node is a dummy node that is added to the beginning or end of a Linked List to simplify certain operations, such as inserting or deleting nodes at the beginning or end of the list.

8. **What is the time complexity of searching for an element in a Linked List?**
   Answer: The time complexity of searching for an element in a Linked List is O(n), where n is the number of nodes in the list.

9. **How is a Linked List traversed recursively?**
   Answer: A Linked List can be traversed recursively by starting at the head of the list and calling a function that takes the current node as an argument and recursively calls itself with the next node in the list.

10. **What is a circular Linked List?**
    Answer: A circular Linked List is a Linked List where the last node points back to the first node, creating a circular structure. This can be either a singly linked or doubly linked list.

Lec 5 - Benefits of using circular list

1. **What is a circular linked list, and how is it different from a linear linked list?**
   Answer: A circular linked list is a type of linked list in which the last node points to the first node, forming a loop. This is different from a linear linked list, where the last node points to NULL.

2. **How can a circular linked list be used to represent a clock?**
   Answer: A circular linked list can be used to represent a clock by having each node represent a minute or an hour. The last node would point back to the first node, creating a circular structure that represents the cyclical nature of time.

3. **What is a circular buffer, and how is it implemented using a circular linked list?**
   Answer: A circular buffer is a data structure that allows for efficient insertion and removal of elements at both ends. It is implemented using a circular linked list by maintaining two pointers, one to the head and one to the tail of the buffer. When an element is inserted, the tail pointer is moved forward, and when an element is removed, the head pointer is moved forward.

4. **What are some common applications of circular linked lists?**
   Answer: Some common applications of circular linked lists include implementing circular buffers, representing circular structures such as clocks or cycles, and implementing certain algorithms that require multiple traversals of the list.

5. **How does a circular linked list conserve memory compared to a linear linked list?**
   Answer: In a linear linked list, the last node points to NULL, which wastes memory. In contrast, in a circular linked list, the last node points to the first node, eliminating the need for a NULL pointer and conserving memory.

6. **How does a circular linked list simplify insertion and deletion at the beginning or end of the list?**
   Answer: A circular linked list simplifies insertion and deletion at the beginning or end of the list by allowing for constant time insertion and deletion operations. This is because the first and last nodes are connected to each other, making it easy to update the pointers when adding or removing nodes.

7. **How can a circular linked list be used in data structures such as hash tables or adjacency lists?**
   Answer: A circular linked list can be used in hash tables or adjacency lists to represent the linked lists associated with each hash table index or vertex, respectively.

8. **What are the advantages of using a circular linked list over a linear linked list?**
   Answer: The advantages of using a circular linked list include efficient implementation of circular structures, faster insertion and deletion operations, efficient implementation of circular buffers, and memory conservation.

9. **How does a circular linked list differ from a doubly linked list?**
   Answer: A circular linked list differs from a doubly linked list in that a circular linked list only has one pointer per node, while a doubly linked list has two pointers per node, one pointing to the previous node and one pointing to the next node.

10. **What are some potential drawbacks of using a circular linked list?**
    Answer: Some potential drawbacks of using a circular linked list include increased complexity in implementation and potential issues with traversing the list indefinitely, leading to an infinite

loop.

Lec 6 - Stack From the Previous Lecture

1. **What is a stack, and how does it follow the LIFO principle?**
   Answer: A stack is an abstract data type that represents a collection of elements with two main operations: push, which adds an element to the top of the stack, and pop, which removes the top element from the stack. It follows the Last-In-First-Out (LIFO) principle, where the last element added is the first to be removed.

2. **How can you implement a stack using an array? What are the advantages and disadvantages of this approach?**
   Answer: A stack can be implemented using an array by maintaining a variable to keep track of the topmost element's index. Advantages of this approach include constant time complexity for push and pop operations and efficient use of memory. Disadvantages include fixed size and difficulty in dynamic resizing.

3. **How can you implement a stack using a linked list? What are the advantages and disadvantages of this approach?**
   Answer: A stack can be implemented using a linked list by using the head of the list to represent the topmost element. Advantages of this approach include dynamic resizing, efficient use of memory, and easy implementation. Disadvantages include slower access time than an array-based implementation.

4. **What is the purpose of the peek operation in a stack?**
   Answer: The peek operation allows you to view the topmost element in the stack without removing it.

5. **What happens when you try to pop an element from an empty stack?**
   Answer: When you try to pop an element from an empty stack, an error message is displayed.

6. **What is the time complexity of push and pop operations in a stack implemented using a linked list?**
   Answer: The time complexity of push and pop operations in a stack implemented using a linked list is O(1).

7. **How can stacks be used to evaluate postfix expressions?**
   Answer: Stacks can be used to evaluate postfix expressions by iterating over the expression and performing operations based on the current element. When an operand is encountered, it is pushed onto the stack. When an operator is encountered, the two most recent operands are popped from the stack, and the operation is performed, with the result being pushed back onto the stack.

8. **What is a potential application of stacks in checking for balanced parentheses in an expression?**
   Answer: A stack can be used to check for balanced parentheses in an expression by pushing opening parentheses onto the stack and popping them off when a closing parenthesis is encountered. If the stack is empty at the end of the expression, then the parentheses are balanced.

9. **What is a potential disadvantage of using an array to implement a stack?**
   Answer: A potential disadvantage of using an array to implement a stack is that the size is fixed and cannot be dynamically resized.

10. **What is the time complexity of searching for an element in a stack implemented using an array?**
    Answer: The time complexity of searching for an element in a stack implemented using an array is O(n), as you need to iterate over each element to find the desired one.

Lec 7 - Evaluating postfix expressions

1. **What is postfix notation, and how is it different from infix notation?**
   Answer: Postfix notation is a mathematical notation where operators are written after their operands. In contrast, infix notation is a notation where operators are written between their operands. The primary difference is that postfix notation eliminates the need for parentheses to indicate the order of operations.

2. **How does a stack data structure help in evaluating postfix expressions?**
   Answer: A stack is used to keep track of operands and operators and perform the necessary calculations. The process involves scanning the expression from left to right, pushing operands onto the stack, and when an operator is encountered, popping the top two operands off the stack, performing the operation, and pushing the result back onto the stack.

3. **How do you evaluate a postfix expression?**
   Answer: The process involves scanning the expression from left to right, pushing operands onto the stack, and when an operator is encountered, popping the top two operands off the stack, performing the operation, and pushing the result back onto the stack. The final result is the top element in the stack after all expressions have been evaluated.

4. **What happens when an operand is encountered in a postfix expression?**
   Answer: It is pushed onto the stack.

5. **What happens when an operator is encountered in a postfix expression?**
   Answer: The top two operands are popped from the stack, and the operation is performed on them. The result is then pushed back onto the stack.

6. **What is the significance of the order of operations in postfix notation?**
   Answer: The order of operations in postfix notation is determined by the order in which the operands and operators are encountered. Operators are applied to the two most recently pushed operands in the stack, so the order of operations is naturally enforced.

7. **How can you detect and handle errors while evaluating a postfix expression?**
   Answer: One common method is to check the stack after evaluating the expression. If there is more than one element left in the stack, it indicates an error. Another method is to check for errors while scanning the expression and handling them as they occur.

8. **Can all mathematical expressions be converted to postfix notation?**
   Answer: Yes, all mathematical expressions can be converted to postfix notation using the algorithm for conversion.

9. **What are some advantages of using postfix notation?**
   Answer: Postfix notation eliminates the need for parentheses to indicate the order of operations and can be evaluated efficiently using a stack data structure.

10. **What are some limitations of using postfix notation?**
    Answer: Postfix notation may be less intuitive to read and write than infix notation, and the conversion process can be time-consuming for complex expressions.

Lec 8 - Conversion from infix to postfix

1. **What is the main advantage of postfix notation over infix notation?**
   Answer: Postfix notation eliminates the need for parentheses to indicate the order of operations.
2. **What is the role of a stack in converting an infix expression to postfix notation?**
   Answer: The stack is used to keep track of operators and their precedence levels.
3. **How do you handle errors while converting an infix expression to postfix notation?**
   Answer: By checking for balanced parentheses and errors during scanning.
4. **What is the first step in converting an infix expression to postfix notation?**
   Answer: Initializing an empty stack and postfix expression.
5. **How do you handle operators with equal precedence levels while converting infix to postfix notation?**
   Answer: Operators with equal precedence levels are added to the postfix expression based on the associativity rules (left to right or right to left).
6. **What happens to a left parenthesis when converting infix to postfix notation?**
   Answer: The left parenthesis is pushed onto the stack.
7. **What is the final step in converting an infix expression to postfix notation?**
   Answer: Pop any remaining operators off the stack and add them to the postfix expression.
8. **What is the difference between infix notation and postfix notation?**
   Answer: In infix notation, operators are written between their operands, while in postfix notation, operators are written after their operands.
9. **What are the advantages of using a postfix notation over infix notation?**
   Answer: Postfix notation eliminates the need for parentheses to indicate the order of operations, and it is easier to evaluate expressions using a stack-based algorithm.
10. **How can you convert an infix expression with nested parentheses to postfix notation?**
    Answer: By using a stack-based algorithm to remove the nested parentheses and convert the expression to postfix notation.

Lec 9 - Memory Organization

1. **What is memory organization?**
   Answer: Memory organization refers to the arrangement of memory blocks and allocation of data in the computer's memory.

2. **What is the memory hierarchy?**
   Answer: The memory hierarchy is a hierarchy of different types of memory with varying access times and storage capacities.

3. **What is the purpose of memory hierarchy?**
   Answer: The purpose of memory hierarchy is to provide faster access to frequently used data and reduce the average access time.

4. **What is cache memory?**
   Answer: Cache memory is a small, fast memory that is used to temporarily store frequently accessed data and instructions.

5. **What is virtual memory?**
   Answer: Virtual memory is a technique that enables a computer to use more memory than it physically has by temporarily transferring data from the main memory to the hard disk.

6. **What is a memory module?**
   Answer: A memory module is a small circuit board that contains multiple memory chips and is used to expand the memory capacity of a computer.

7. **What is a memory controller?**
   Answer: A memory controller is a device that manages the flow of data between the computer's CPU and memory.

8. **What is the role of the memory controller?**
   Answer: The memory controller is responsible for controlling the access to memory, optimizing data transfer, and managing the memory hierarchy.

9. **What is DRAM?**
   Answer: DRAM (Dynamic Random Access Memory) is a type of memory that is commonly used in computers for main memory.

10. **What is SRAM?**
    Answer: SRAM (Static Random Access Memory) is a type of memory that is faster and more expensive than DRAM and is commonly used in cache memory.

Lec 10 - Queues

1. **What is the difference between a queue and a stack?**
   Answer: A queue is a data structure where the first element added is the first one to be removed (FIFO), while a stack is a data structure where the last element added is the first one to be removed (LIFO).

2. **How is a queue implemented using a linked list?**
   Answer: A queue can be implemented using a linked list by adding elements to the tail of the linked list and removing elements from the head of the linked list.

3. **What is a circular queue, and what are its advantages?**
   Answer: A circular queue is a queue where the last element points to the first element, forming a circle. The advantages of a circular queue are that it can utilize space better than a regular queue and allows efficient use of memory when implementing a buffer.

4. **What is a priority queue, and how is it different from a regular queue?**
   Answer: A priority queue is a data structure where each element has a priority assigned to it, and elements with higher priority are dequeued first. A regular queue, on the other hand, follows the first-in, first-out (FIFO) principle.

5. **How can a queue be used to implement a breadth-first search (BFS) algorithm?**
   Answer: A queue can be used to implement BFS by adding the starting node to the queue and then removing it and adding its adjacent nodes to the queue, repeating this process until the desired node is found.

6. **What is a blocking queue, and how does it work?**
   Answer: A blocking queue is a queue that blocks when attempting to dequeue from an empty queue or enqueue to a full queue. The blocking queue will wait until an element is available or space becomes available to perform the desired operation.

7. **What is a double-ended queue (deque), and what are its advantages?**
   Answer: A double-ended queue, also known as a deque, is a data structure that allows insertion and deletion at both ends. The advantages of a deque are that it can be used as both a stack and a queue and provides more flexibility in implementing certain algorithms.

8. **How can a queue be used to implement a producer-consumer problem?**
   Answer: In a producer-consumer problem, a queue can be used as a buffer between the producer and consumer, where the producer adds items to the queue, and the consumer removes them. The queue ensures that the producer and consumer can work independently and at their own pace.

9. **What is a concurrent queue, and how does it work?**
   Answer: A concurrent queue is a queue that can be accessed by multiple threads simultaneously. It works by using thread-safe operations for enqueueing and dequeueing elements to ensure that the queue remains consistent and free from race conditions.

10. **How can a queue be used to implement a call center waiting system?**
    Answer: In a call center waiting system, a queue can be used to hold calls that are waiting to be answered by the next available representative. As representatives become available, calls are dequeued from the queue and assigned to them. The queue ensures that callers are served in the order they called.

Lec 11 - Implementation of Priority Queue

1. **What is a Priority Queue?**
   Answer: A Priority Queue is an abstract data type that stores a collection of elements with priority values assigned to each element. Elements with higher priority values are given precedence over those with lower priority values.

2. **What is the difference between a Queue and a Priority Queue?**
   Answer: A Queue is a data structure that follows the First-In-First-Out (FIFO) principle, while a Priority Queue follows the priority-based ordering principle. In a Queue, elements are added to the back and removed from the front, while in a Priority Queue, elements are removed based on their priority values.

3. **What are the commonly used data structures to implement a Priority Queue?**
   Answer: The commonly used data structures to implement a Priority Queue are binary heap, Fibonacci heap, and sorted array.

4. **What is a binary heap?**
   Answer: A binary heap is a complete binary tree where the parent node has a higher priority value than its children nodes.

5. **What are the operations that can be performed on a Priority Queue?**
   Answer: The operations that can be performed on a Priority Queue are inserting an element, deleting the element with the highest priority, and changing the priority of an element.

6. **What is the time complexity of inserting an element into a Priority Queue?**
   Answer: The time complexity of inserting an element into a Priority Queue depends on the implementation. For a binary heap, the time complexity is O(log n), while for a Fibonacci heap, it is O(1).

7. **What is the time complexity of deleting the element with the highest priority from a Priority Queue?**
   Answer: The time complexity of deleting the element with the highest priority from a Priority Queue depends on the implementation. For a binary heap, the time complexity is O(log n), while for a Fibonacci heap, it is O(log n) amortized.

8. **What is the difference between a Max Heap and a Min Heap?**
   Answer: In a Max Heap, the parent node has a higher priority value than its children nodes, while in a Min Heap, the parent node has a lower priority value than its children nodes.

9. **What is the application of Priority Queues?**
   Answer: Priority Queues are used in various applications such as task scheduling, Dijkstra's shortest path algorithm, Huffman coding, A* search, and many more.

10. **How can a Priority Queue be implemented in C++?**
    Answer: A Priority Queue can be implemented in C++ using the std::priority_queue class from the Standard Template Library (STL).

Lec 12 - Operations on Binary Tree

1. **What is a binary tree?**

Answer: A binary tree is a tree data structure in which each node has at most two children, referred to as the left child and the right child.

2. **What is the height of a binary tree?**

Answer: The height of a binary tree is the length of the longest path from the root node to a leaf node.

3. **What is the difference between a binary tree and a binary search tree?**

Answer: A binary search tree is a binary tree with the property that the value of each node is greater than or equal to the values in its left subtree and less than or equal to the values in its right subtree.

4. **What is the time complexity of inserting a node in a binary tree?**

Answer: The time complexity of inserting a node in a binary tree is O(h), where h is the height of the tree.

5. **What is the time complexity of deleting a node in a binary tree?**

Answer: The time complexity of deleting a node in a binary tree is O(h), where h is the height of the tree.

6. **What is the difference between pre-order traversal and post-order traversal?**

Answer: Pre-order traversal visits the root node first, followed by the left subtree and then the right subtree, while post-order traversal visits the left subtree first, followed by the right subtree and then the root node.

7. **How do you determine if a binary tree is balanced?**

Answer: A binary tree is balanced if the height of its left subtree and the height of its right subtree differ by at most one.

8. **What is the difference between complete binary tree and a full binary tree?**

Answer: A complete binary tree is a binary tree in which every level except possibly the last is completely filled, while a full binary tree is a binary tree in which every node has either two children or zero children.

9. **What is the time complexity of finding the maximum element in a binary tree?**

Answer: The time complexity of finding the maximum element in a binary tree is O(n), where n is the number of nodes in the tree.

10. **What is the time complexity of finding the height of a binary tree using dynamic programming?**

Answer: The time complexity of finding the height of a binary tree using dynamic programming is O(n), where n is the number of nodes in the tree.

Lec 13 - Cost of Search

1. **What is the cost of search and why is it important in computer science?**
   Answer: The cost of search refers to the amount of time, resources, and computational power required to search for a specific item in a data structure. It is important in computer science because efficient search algorithms can significantly reduce search costs and improve overall system performance.

2. **What are the common measures of search cost?**
   Answer: Common measures of search cost include time complexity, space complexity, and worst-case analysis.

3. **What is the time complexity of binary search?**
   Answer: The time complexity of binary search is O(log n).

4. **What is the disadvantage of linear search?**
   Answer: The disadvantage of linear search is that it has a time complexity of O(n), which makes it inefficient for large datasets.

5. **What is the advantage of hash tables for search operations?**
   Answer: Hash tables provide constant time search operations with a good hash function.

6. **What is the time complexity of searching a hash table with a good hash function?**
   Answer: The time complexity of searching a hash table with a good hash function is O(1).

7. **What is the disadvantage of binary search?**
   Answer: The disadvantage of binary search is that it can only be used with sorted arrays.

8. **What is the advantage of binary search over linear search?**
   Answer: The advantage of binary search over linear search is that it has a time complexity of O(log n), which makes it more efficient for large datasets.

9. **What is worst-case analysis for search algorithms?**
   Answer: Worst-case analysis is a measure of the maximum amount of time or space required to perform a specific operation in the worst-case scenario.

10. **How can search costs be reduced in data structures?**
    Answer: Search costs can be reduced in data structures by using efficient search algorithms, such as binary search or hash tables, and by implementing balanced trees or other optimized data structures.

Lec 14 - Recursive Calls

1. **What is recursion?**
   Answer: Recursion is a programming technique where a function calls itself within its own code.

2. **What is the base case in recursion?**
   Answer: The base case is the case where the function returns a value without calling itself, stopping the recursive process.

3. **What is the difference between direct and indirect recursion?**
   Answer: Direct recursion occurs when a function calls itself, while indirect recursion occurs when two or more functions call each other.

4. **What is tail recursion?**
   Answer: Tail recursion is a type of recursion where the recursive call is the last operation performed by the function.

5. **What is the maximum number of recursive calls that can be made?**
   Answer: The maximum number of recursive calls that can be made depends on the available memory.

6. **What is a stack overflow error?**
   Answer: A stack overflow error occurs when the maximum stack size is exceeded due to too many recursive calls.

7. **What are the advantages of recursion?**
   Answer: Recursion can simplify complex problems by breaking them down into smaller subproblems, and can be more readable and concise than iterative solutions.

8. **What are the disadvantages of recursion?**
   Answer: Recursion may cause stack overflow errors and can be less efficient than iterative solutions.

9. **What is the role of the base case in recursion?**
   Answer: The base case provides a stopping condition for the recursive process.

10. **What are some common algorithms that use recursion?**
    Answer: Quick sort, merge sort, and binary search are common algorithms that use recursion.

Lec 15 - Level-order Traversal of a Binary Tree

1. **What is level-order traversal of a binary tree?**
   Answer: Level-order traversal is a technique used to traverse a binary tree in which nodes are visited level by level, from top to bottom and from left to right.

2. **How can level-order traversal be implemented?**
   Answer: Level-order traversal can be implemented using a queue data structure.

3. **What is the time complexity of level-order traversal?**
   Answer: The time complexity of level-order traversal is O(n), where n is the number of nodes in the binary tree.

4. **What is the space complexity of level-order traversal?**
   Answer: The space complexity of level-order traversal is O(n), where n is the number of nodes in the binary tree.

5. **Can level-order traversal be used to find the minimum depth of a binary tree?**
   Answer: Yes, level-order traversal can be used to find the minimum depth of a binary tree.

6. **Can level-order traversal be used to find the maximum depth of a binary tree?**
   Answer: Yes, level-order traversal can be used to find the maximum depth of a binary tree.

7. **Can level-order traversal be used to sort the nodes in a binary tree?**
   Answer: No, level-order traversal cannot be used to sort the nodes in a binary tree.

8. **Can level-order traversal be used to find the lowest common ancestor of two nodes in a binary tree?**
   Answer: Yes, level-order traversal can be used to find the lowest common ancestor of two nodes in a binary tree.

9. **What is the advantage of level-order traversal?**
   Answer: The advantage of level-order traversal is that it can be used to find the shortest path between two nodes.

10. **What is the disadvantage of level-order traversal?**
    Answer: The disadvantage of level-order traversal is that it requires more memory than other traversal techniques.

Lec 16 - Deleting a node in BST

1. **What is your favorite book and why?**
   Answer: My favorite book is "To Kill a Mockingbird" by Harper Lee. I love the way it explores themes of racism, justice, and morality through the eyes of a child. The characters are well-developed and the story is both heartwarming and heart-wrenching.

2. **What do you think is the most important social issue facing us today?**
   Answer: In my opinion, the most important social issue facing us today is climate change. It is a global problem that affects everyone and everything on our planet. We need to take action to reduce greenhouse gas emissions and transition to a more sustainable way of living.

3. **What is your favorite way to relax?**
   Answer: My favorite way to relax is to read a good book or listen to music. I find that both activities allow me to escape from the stresses of everyday life and recharge my batteries.

4. **What do you think is the biggest challenge facing young people today?**
   Answer: I believe that the biggest challenge facing young people today is finding meaningful work in a rapidly changing economy. With automation and globalization, many traditional jobs are disappearing and it can be difficult to find a career path that offers stability and fulfillment.

5. **What is your favorite place to travel to and why?**
   Answer: My favorite place to travel to is Japan. I love the mix of ancient traditions and modern technology, the delicious food, and the beautiful landscapes. Plus, the people are incredibly friendly and welcoming.

6. **What is your favorite hobby and why?**
   Answer: My favorite hobby is playing music. I love the creative outlet it provides, the way it connects me with others, and the sense of accomplishment I feel when I master a new song.

7. **What is your favorite quote and why?**
   Answer: My favorite quote is "Be the change you wish to see in the world" by Mahatma Gandhi. I love the message of personal responsibility and the idea that each of us has the power to make a difference in the world.

8. **What is the most important lesson you have learned in life so far?**
   Answer: The most important lesson I have learned in life so far is that relationships are the most valuable thing we have. Whether it's family, friends, or coworkers, the people in our lives are what make life worth living.

9. **What is your biggest fear and how do you deal with it?**
   Answer: My biggest fear is failure. To deal with it, I try to focus on the process rather than the outcome. I remind myself that it's okay to make mistakes and that every failure is an opportunity to learn and grow.

10. **What is your favorite memory and why?**
    Answer: My favorite memory is the day I got married. It was a beautiful day filled with love, laughter, and joy. I felt surrounded by the people I care about most and I knew that I was embarking on a new chapter of my life with my best friend by my side.

Lec 17 - Reference Variables

1. **What is a reference variable in Java?**
   Answer: A reference variable in Java is a variable that holds the memory address of an object.

2. **How is a reference variable different from a primitive variable in Java?**
   Answer: A reference variable holds a reference to an object, while a primitive variable holds the actual value of a data type.

3. **What is the default value of a reference variable in Java?**
   Answer: The default value of a reference variable in Java is null.

4. **Can a reference variable be reassigned to a different object in Java?**
   Answer: Yes, a reference variable can be reassigned to a different object in Java.

5. **How does Java handle passing a reference variable as a parameter to a method?**
   Answer: Java passes the reference to the object held by the reference variable as a parameter to the method.

6. **Can a reference variable be used to access static methods in Java?**
   Answer: Yes, a reference variable can be used to access static methods in Java.

7. **How does Java handle garbage collection for objects referenced by reference variables?**
   Answer: Java's garbage collector automatically frees the memory allocated to objects that are no longer referenced by any reference variable.

8. **What is the difference between an instance variable and a reference variable in Java?**
   Answer: An instance variable is a variable declared in a class, while a reference variable is a variable declared in a method or block that holds a reference to an object.

9. **Can a reference variable be used to access private members of a class in Java?**
   Answer: No, a reference variable cannot be used to access private members of a class in Java.

10. **How can we check if a reference variable is referring to an object or is null in Java?**
    Answer: We can check if a reference variable is referring to an object or is null in Java by using the null check operator (==).

Lec 18 - Reference Variables

1. **What is a reference variable in Java?**
   Answer: A reference variable is a variable that holds the memory address of an object in Java.

2. **Can multiple reference variables refer to the same object in Java?**
   Answer: Yes, multiple reference variables can refer to the same object in Java.

3. **What is the default value of a reference variable in Java?**
   Answer: The default value of a reference variable in Java is null.

4. **How does Java handle garbage collection for objects referenced by reference variables?**
   Answer: Java's garbage collector automatically frees up memory allocated to objects that are no longer being referenced by any reference variable.

5. **Can a reference variable be null in Java?**
   Answer: Yes, a reference variable can be null in Java.

6. **Can a reference variable be reassigned to a different object in Java?**
   Answer: Yes, a reference variable can be reassigned to a different object in Java.

7. **Can a reference variable be used to access static methods in Java?**
   Answer: Yes, a reference variable can be used to access static methods in Java.

8. **What is the difference between a reference variable and a primitive variable in Java?**
   Answer: A reference variable holds the memory address of an object, while a primitive variable holds the actual value of a data type.

9. **Can a reference variable be used to access private members of a class in Java?**
   Answer: No, a reference variable cannot be used to access private members of a class in Java.

10. **How can we check if a reference variable is null in Java?**
    Answer: We can use the == operator to check if a reference variable is null in Java.

Lec 19 - Usage of const keyword

1. **What is the purpose of the const keyword in programming?**
   Answer: The const keyword is used to declare variables that cannot be modified once they are initialized.
2. **Why is using const variables beneficial in a program?**
   Answer: Using const variables can help prevent bugs and improve program stability.
3. **Can const variables be modified after they are initialized?**
   Answer: No, const variables cannot be modified after they are initialized.
4. **What happens if you try to modify a const variable in C++?**
   Answer: The compiler generates an error.
5. **Is the const keyword required when passing a variable by reference in C++?**
   Answer: Yes, the const keyword is required when passing a variable by reference in C++.
6. **Can a member function of a C++ class be declared as const?**
   Answer: Yes, a member function of a C++ class can be declared as const.
7. **What is the difference between a const pointer and a pointer to a const variable in C++?**
   Answer: A const pointer is a pointer that cannot be modified to point to a different memory address, while a pointer to a const variable is a pointer that cannot be used to modify the value of the variable it points to.
8. **Can a const variable be initialized with a value at runtime in C++?**
   Answer: No, a const variable must be initialized with a value at compile-time in C++.
9. **What are some examples of variables that are commonly declared as const in C++?**
   Answer: Constants used in mathematical calculations, physical constants, and program-specific constants are all examples of variables that are commonly declared as const in C++.
10. **Is the const keyword used in other programming languages besides C++?**
    Answer: Yes, the const keyword is used in many programming languages besides C++.

Lec 20 - AVL Tree

1. **What is AVL Tree and what is its purpose?**
   Answer: AVL Tree is a self-balancing binary search tree where the difference between the height of the left and right subtrees cannot be more than one for all nodes. Its purpose is to provide faster operations for search, insertion, and deletion compared to other self-balancing trees.

2. **What is the difference between AVL Tree and Red-Black Tree?**
   Answer: Both AVL Tree and Red-Black Tree are self-balancing binary search trees. The main difference between them is that AVL Tree guarantees that the difference between the heights of left and right subtrees is at most one, whereas Red-Black Tree guarantees that the longest path from the root to a leaf is no more than twice as long as the shortest path.

3. **How does rotation help in balancing the AVL Tree?**
   Answer: Rotation is the operation performed to balance the AVL Tree. It involves changing the structure of the tree by rotating a node to a new position, which helps to maintain the balance of the tree by keeping the height difference of the left and right subtrees at most one.

4. **What is the height of an AVL Tree with one node?**
   Answer: The height of an AVL Tree with one node is 0.

5. **Can AVL Tree have duplicate keys?**
   Answer: No, AVL Tree cannot have duplicate keys.

6. **What is the time complexity of search operation in AVL Tree?**
   Answer: The time complexity of search operation in AVL Tree is O(log n).

7. **What is the worst-case time complexity of insertion operation in AVL Tree?**
   Answer: The worst-case time complexity of insertion operation in AVL Tree is O(log n).

8. **How does AVL Tree maintain balance after a node is inserted or deleted?**
   Answer: AVL Tree maintains balance by performing rotations after a node is inserted or deleted to ensure that the height difference of the left and right subtrees is at most one.

9. **What is the height of a perfectly balanced AVL Tree with 15 nodes?**
   Answer: The height of a perfectly balanced AVL Tree with 15 nodes is 3.

10. **What is the purpose of AVL Tree being self-balancing?**
    Answer: The purpose of AVL Tree being self-balancing is to ensure that the worst-case time complexity of operations like search, insertion, and deletion is O(log n), which is much faster than other non-balanced binary search trees.

Lec 21 - AVL Tree Building Example

1. **What is an AVL Tree and what is its significance in computer science?**

Answer: An AVL Tree is a self-balancing binary search tree in which the heights of the left and right subtrees of any node differ by at most one. Its significance in computer science lies in its ability to maintain efficient search, insertion and deletion operations with a guaranteed time complexity of O(log n).

2. **What is the difference between a balanced binary search tree and an unbalanced binary search tree?**

Answer: A balanced binary search tree maintains a balance between the height of the left and right subtrees of any node, ensuring a time complexity of O(log n) for its operations. An unbalanced binary search tree, on the other hand, can have a skewed structure that results in a time complexity of O(n) for its operations.

3. **How does the AVL Tree maintain balance during insertion and deletion operations?**

Answer: The AVL Tree maintains balance during insertion and deletion operations by performing rotations at the nodes where the balance factor (the difference between the heights of the left and right subtrees) is greater than one. The rotations are performed to move the subtrees and maintain the balance factor within the acceptable range.

4. **What is the height of a perfectly balanced AVL Tree with 10 nodes?**

Answer: The height of a perfectly balanced AVL Tree with 10 nodes would be 4.

5. **What is the difference between a single rotation and a double rotation in an AVL Tree?**

Answer: A single rotation is performed to balance an AVL Tree when the balance factor of a node is greater than one, and it involves rotating the node and its subtree once. A double rotation, on the other hand, involves performing two rotations to balance the tree, either in the same or opposite directions.

6. **Can a binary search tree be both height-balanced and weight-balanced?**

Answer: Yes, a binary search tree can be both height-balanced and weight-balanced. AVL Trees are an example of a height-balanced binary search tree, while Red-Black Trees are an example of a weight-balanced binary search tree.

7. **Why is it important to maintain balance in a binary search tree?**

Answer: It is important to maintain balance in a binary search tree to ensure that the search, insertion and deletion operations have a guaranteed time complexity of O(log n), making them efficient for large datasets.

8. **What is the time complexity of searching for a node in an AVL Tree?**

Answer: The time complexity of searching for a node in an AVL Tree is O(log n).

9. **Can an AVL Tree have duplicate nodes?**

Answer: Yes, an AVL Tree can have duplicate nodes, but they would need to be stored in a specific way, such as storing a count for each duplicate node.

10. **What is the root node of an AVL Tree?**

Answer: The root node of an AVL Tree is the topmost node in the tree, from which all other nodes are descended.

Lec 22 - Cases of rotations

1. **What is a rotation in a binary search tree?**
   Answer: A rotation is a manipulation performed on a binary search tree to maintain its balance.

2. **What are the two types of rotations in a binary search tree?**
   Answer: The two types of rotations are single rotations and double rotations.

3. **How do rotations help in balancing a binary search tree?**
   Answer: Rotations help in redistributing the nodes of a binary search tree to maintain balance, which in turn helps in efficient search operations.

4. **When is a single left rotation used in a binary search tree?**
   Answer: A single left rotation is used when the imbalance occurs in the immediate left child of a node.

5. **When is a single right rotation used in a binary search tree?**
   Answer: A single right rotation is used when the imbalance occurs in the immediate right child of a node.

6. **What is the difference between a single rotation and a double rotation in a binary search tree?**
   Answer: A single rotation involves rotating only one node, while a double rotation involves rotating two nodes.

7. **What is the maximum number of rotations required to balance a node in a binary search tree**?
   Answer: The maximum number of rotations required to balance a node in a binary search tree is two.

8. **What is the left-right case in a binary search tree rotation?**
   Answer: The left-right case occurs when the left child of a node has a right child, and the subtree is imbalanced.

9. **What is the right-left case in a binary search tree rotation?**
   Answer: The right-left case occurs when the right child of a node has a left child, and the subtree is imbalanced.

10. **What is the purpose of using rotations in a binary search tree?**
    Answer: The purpose of using rotations in a binary search tree is to maintain its balance and ensure efficient search operations.