Question 1:

Q: Explain the fundamental property of a Binary Search Tree (BST).

A: The fundamental property of a Binary Search Tree is that for each node:

* All nodes in its left subtree have keys less than the node's key.
* All nodes in its right subtree have keys greater than the node's key.

Question 2:

Q: Why is the average time complexity of search, insertion, and deletion operations in a balanced Binary Search Tree O(log n)?

A: In a balanced BST, the height is logarithmic with respect to the number of nodes (n). The average time complexity of operations is O(log n) because each level in the tree reduces the search space by a constant factor.

Question 3:

Q: What is the purpose of balancing a Binary Search Tree?

A: Balancing a BST ensures that its height remains logarithmic, maintaining efficient search, insertion, and deletion operations. Unbalanced trees may lead to worst-case scenarios with a linear height, resulting in O(n) time complexity for operations.

Question 4:

Q: How does the In-order Traversal of a Binary Search Tree provide sorted keys?

A: In-order traversal of a BST visits nodes in ascending order. By visiting left subtree, root, and right subtree in that order, it ensures that smaller keys are encountered before larger keys.

Graph:

Question 5:

Q: Define a graph in terms of vertices and edges.

A: A graph is a collection of vertices (nodes) and edges that connect pairs of vertices. Edges may be directed or undirected, and they represent relationships between vertices.

Question 6:

Q: What is the difference between a directed graph and an undirected graph?

A: In a directed graph, edges have a direction, indicating a one-way relationship between vertices. In an undirected graph, edges have no direction, representing a mutual relationship between connected vertices.

Question 7:

Q: Explain the term "adjacency matrix" in the context of graph representation.

A: An adjacency matrix is a square matrix used to represent a graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not. For an undirected graph, the matrix is symmetric.

Question 8:

Q: What is the purpose of the depth-first search (DFS) algorithm in graph traversal?

A: The depth-first search algorithm is used for graph traversal to explore as far as possible along each branch before backtracking. It is often employed for tasks like topological sorting, connected component identification, and pathfinding.

Question 9:

Q: Differentiate between a cyclic and an acyclic graph.

A: A cyclic graph contains at least one cycle (a path that starts and ends at the same vertex). An acyclic graph has no cycles; it is a graph without closed loops.

# Question 10:

Q: How does Dijkstra's algorithm find the shortest path in a weighted graph?

A: Dijkstra's algorithm works by iteratively selecting the vertex with the smallest known distance, updating the distances of its neighbors, and continuing until all vertices are explored. It guarantees finding the shortest path in a weighted, non-negative graph.

# Question 11:

Q: Explain the concept of an "Eulerian path" in a graph.

A: An Eulerian path is a path in a graph that visits every edge exactly once. If a graph has an Eulerian path, it must have at most two vertices with an odd degree (an odd number of edges incident on them).

# Question 12:

Q: What is the significance of the "adjacency list" in graph representation?

A: An adjacency list is a data structure used to represent a graph. It provides a compact way to store the connections between vertices by listing the neighbors of each vertex. This representation is often more memory-efficient for sparse graphs.

# Question 13:

Q: Define the term "connected components" in the context of a graph.

A: Connected components in a graph are subgraphs where any two vertices are connected by paths. A graph may have multiple connected components, and each component is disconnected from the others.

# Question 14:

Q: How does the Breadth-First Search (BFS) algorithm explore a graph?

A: The Breadth-First Search algorithm explores a graph level by level. It starts at a given vertex, visits all its neighbors, then moves to their neighbors, and so on. It ensures that all vertices at distance k from the starting vertex are visited before vertices at distance k+1.

# Question 15:

Q: In the context of graphs, what is the significance of a "spanning tree"?

A: A spanning tree of a connected graph is a subgraph that includes all the vertices of the original graph and forms a tree without any cycles. It ensures connectivity while minimizing the number of edges.

# Asymptotic Notations Question 1:

Q: What does Big O notation (O) represent in the context of algorithms?

A: Big O notation (O) represents an upper bound on the growth rate of a function, providing an estimate of the worst-case scenario in terms of time or space complexity.

# Question 2:

Q: Define Omega notation (Ω) and its significance.

A: Omega notation (Ω) represents a lower bound on the growth rate of a function, indicating the best-case scenario. It provides insight into the lower limits of algorithm efficiency.

# Question 3:

Q: What does Theta notation (Θ) express about the growth rate of a function?

A: Theta notation (Θ) provides a tight bound on the growth rate of a function, indicating both upper and lower bounds. It describes the average-case behavior of an algorithm.

# Question 4:

Q: What information does Omega notation provide about the growth rate of a function?

A: Omega notation provides a lower bound on the growth rate of a function, indicating the best-case behavior and the lower limits of efficiency.

# Question 5:

Q: If a function (f(n)) is (Theta(g(n))), what can be inferred about its growth rate?

A: If (f(n)) is (Theta(g(n))), it means that (f(n)) grows at the same rate as (g(n)) within constant factors. It describes the average-case complexity of an algorithm.

# Question 6:

Q: In practical terms, how are asymptotic notations useful in algorithm analysis?

A: Asymptotic notations provide a concise way to express the efficiency of algorithms. They help analyze the behavior of algorithms as input sizes approach infinity, aiding in comparing and classifying algorithms based on their time or space complexity.

# Question 7:

Q: Can a function have multiple asymptotic notations? Explain.

A: Yes, a function can have multiple asymptotic notations. For example, if \(f(n)\) is both

\(O(g(n))\) and \(\Omega(g(n))\), then it is also \(\Theta(g(n))\). This indicates both upper and lower bounds on the growth rate.

# Question 8:

Q: Why is Big O notation commonly used to describe the time complexity of algorithms?

A: Big O notation is commonly used because it provides an upper bound on the worst-case time complexity of algorithms, allowing for a simplified and standardized way to express and compare algorithmic efficiency.

# Question 9:

Q: How does Omega notation (Omega) differ from Big O notation (O)?

A: Omega notation (Omega) represents a lower bound, indicating the best-case scenario. In contrast, Big O notation (O) represents an upper bound, indicating the worst-case scenario. Both notations describe the growth rate of a function.