**Section 1: Multiple Choice Questions (10 Questions)**

**Recursion:**

1. **Which of the following is true about recursion?**
   * a) Recursion always consumes more memory than iteration
   * b) Recursion always terminates
   * c) The base case is essential to prevent infinite recursion
   * d) Recursion cannot be used for problems that involve backtracking

**Searching and Sorting:**

1. **Which of the following algorithms is best suited for searching in a sorted array?**
   * a) Linear Search
   * b) Binary Search
   * c) Hash Search
   * d) Selection Sort
2. **Which of the following sorting algorithms has the best average-case time complexity?**
   * a) Bubble Sort
   * b) Merge Sort
   * c) Quick Sort
   * d) Insertion Sort
3. **In Quick Sort, what is the role of the 'pivot'?**
   * a) The pivot is always the first element
   * b) The pivot helps partition the array into two sub-arrays
   * c) The pivot is used to select the largest element
   * d) The pivot is used for finalizing the sorted array

**Tables and Hashing:**

1. **Which technique is used to resolve collisions in hash tables?**
   * a) Linear Probing
   * b) Backtracking
   * c) Bubble Sort
   * d) None of the above
2. **In which scenario would you prefer linear probing over chaining in hash tables?**
   * a) When hash table is sparsely populated
   * b) When hash table is fully populated
   * c) When there are frequent deletions
   * d) None of the above

**Graphs and Algorithms:**

1. **What is the time complexity of Dijkstra's algorithm when using a priority queue with an adjacency list?**
   * a) O(V^2)
   * b) O(V log V + E log V)
   * c) O(E log V)
   * d) O(V + E)
2. **Which graph representation is more efficient for checking whether two nodes are connected?**
   * a) Adjacency List
   * b) Adjacency Matrix
   * c) Hash Table
   * d) Linked List
3. **What is the primary disadvantage of using an adjacency matrix to represent a graph?**
   * a) High memory usage for sparse graphs
   * b) Cannot represent directed graphs
   * c) Slower to find adjacent nodes
   * d) Not suitable for weighted graphs
4. **Which of the following statements is true about the time complexity of merge sort?**

* a) O(n log n) for average and worst-case
* b) O(n^2) for average and worst-case
* c) O(n) for average and worst-case
* d) O(log n) for worst-case

**Section 2: Short Answer Questions (10 Questions)**

1. **What is the difference between binary search and linear search in terms of time complexity?**
2. **How does the merge step work in the merge sort algorithm?**
3. **What is the base case in recursion, and why is it important?**
4. **Describe how linear probing works when inserting into a hash table.**
5. **What is the difference between chaining and linear probing in hash tables?**
6. **What is the purpose of the pivot in Quick Sort? How does it help with sorting?**
7. **How can you detect a cycle in a graph using DFS?**
8. **Explain how Dijkstra’s algorithm guarantees the shortest path from the source node to other nodes.**
9. **In a graph, how would you differentiate between a directed and an undirected edge?**
10. **Why is the adjacency list representation of a graph better for sparse graphs than the adjacency matrix?**

**Section 3: Long Answer Questions (5 Questions)**

1. **Explain the difference between a recursive and an iterative solution to finding the factorial of a number. Provide an example of both approaches in Python.**
2. **Describe the steps of the Quick Sort algorithm and explain why its average case is more efficient than other simple sorts like bubble sort or selection sort.**
3. **Write a detailed explanation of how hash tables work, discussing how collisions are handled with linear probing and chaining.**
4. **Discuss the steps and rationale behind using Dijkstra’s algorithm for finding the shortest path in a graph. Include a simple example.**
5. **Compare and contrast BFS and DFS in terms of their applications, advantages, and when you would use one over the other. Provide examples where each would be useful.**

**Section 4: Diagrams (4 Questions)**

1. **Draw the adjacency matrix representation of the following graph:**

* Vertices: {A, B, C, D, E}
* Edges: {A → B, A → D, B → C, C → D, D → E}

1. **Draw a hash table using linear probing for collision resolution. Use the following keys: {20, 35, 17, 50, 55}. Assume a table size of 7.**
2. **Draw a diagram illustrating the recursion stack for a recursive implementation of the factorial function for factorial(4).**
3. **Illustrate the partition process of Quick Sort when choosing the last element as the pivot in an array [3, 7, 2, 9, 4].**

**Section 5: Code Implementation (5 Questions)**

1. **Write a recursive Python function to find the factorial of a number.**
2. **Implement a binary search algorithm in Python that returns the index of the target element in a sorted list.**
3. **Write the Python code for a hash table using chaining for collision resolution. Implement insertion and search functions.**
4. **Write a Python function that performs merge sort on an unsorted list of integers.**
5. **Write a Python implementation of Dijkstra’s algorithm that finds the shortest path from a source vertex in a graph. Assume the graph is represented by an adjacency list.**

**Answers:**

**Section 1: Multiple Choice**

1. b) Adjacency List
2. b) Binary Search
3. b) Merge Sort
4. b) The pivot helps partition the array into two sub-arrays
5. a) Linear Probing
6. a) When hash table is sparsely populated
7. b) O(V log V + E log V)
8. b) Adjacency Matrix
9. a) High memory usage for sparse graphs
10. a) O(n log n) for average and worst-case

**Section 2: Short Answers**

1. Binary search is O(log n) for sorted arrays, while linear search is O(n) for unsorted arrays.
2. In merge sort, the merge step combines two sorted halves into one sorted array.
3. The base case in recursion is a condition that stops further recursive calls. It prevents infinite recursion.
4. Linear probing checks consecutive table slots for an empty spot if the desired index is already occupied.
5. Chaining stores multiple elements in the same slot using a linked list, while linear probing checks sequentially in the array.
6. The pivot is used to partition the array into smaller and larger elements, helping with sorting.
7. DFS can detect cycles by marking nodes as visited and checking if a back edge leads to an already visited node.
8. Dijkstra’s algorithm iteratively picks the closest unvisited node, updating distances to its neighbors until all nodes are processed.
9. A directed edge has a direction (from one vertex to another), while an undirected edge connects two vertices bidirectionally.
10. Adjacency lists are space-efficient for sparse graphs since they only store actual edges, unlike adjacency matrices which store all possible edges.

**Section 3: Long Answer**

1. **Recursion vs Iteration in Factorial**: Recursive solutions call themselves, while iterative ones use loops. Recursive solutions are elegant but might consume more memory. Example code in Python:

python

Copy code

# Recursive factorial

def factorial\_recursive(n):

if n == 0:

return 1

else:

return n \* factorial\_recursive(n - 1)

1. **Quick Sort**: Quick Sort selects a pivot, partitions the array around the pivot, and recursively sorts the subarrays. It is O(n log n) on average because the partitioning is efficient.
2. **Hash Tables**: Hash tables store key-value pairs. Collisions can be resolved using chaining (linked lists) or linear probing (sequential checking).
3. **Dijkstra's Algorithm**: It processes nodes by selecting the unvisited node with the smallest tentative distance and updates the shortest path to its neighbors.
4. **BFS vs DFS**: BFS explores nodes level-by-level, ideal for shortest-path search. DFS explores as deep as possible, used for finding connected components or solving puzzles.