Long Questions

1. What is an Algorithm?
2. Define Space Complexity in Algorithm Analysis
3. Explain Time Complexity in Algorithm Analysis
4. What are Asymptotic Notations in Algorithm Analysis?
5. Define Big O Notation
6. Explain Omega Notation
7. Define Theta Notation
8. What is Little O Notation Used For?
9. Explain the Divide and Conquer Algorithm Design Paradigm
10. Provide an Example of a Problem Where the Divide and Conquer Technique is Commonly Used
11. Describe the Binary Search Algorithm and Its Time Complexity
12. Explain the Quick Sort Algorithm and Its Time Complexity
13. Describe the Merge Sort Algorithm and Its Time Complexity
14. What is Strassen's Matrix Multiplication?
15. Explain the Time Complexity of Strassen's Matrix Multiplication
16. How does the choice of algorithmic notation (e.g., Big O vs. Theta) impact the analysis of algorithms?

17. Compare and contrast Space Complexity and Time Complexity in algorithm analysis.
18. Give an example of an algorithm with high time complexity and low space complexity.
19. Explain how Divide and Conquer algorithms can benefit from parallel processing.
20. Provide an example of a problem where Little O notation is applicable.
21. How does the choice of pivot element affect the performance of the Quick Sort algorithm?
22. What is the primary advantage of Merge Sort over Quick Sort in terms of stability?
23. How does the choice of algorithm affect real-world applications in terms of performance?
24. Explain the concept of "problem size" in the context of algorithm analysis.
25. What are some common challenges in designing Divide and Conquer algorithms?
26. How do you determine whether an algorithm is suitable for a specific problem in real-world applications?
27. Explain the concept of "in-place" sorting algorithms and provide an example.
28. What are the potential drawbacks of using Strassen's Matrix Multiplication in practice?
29. Describe a scenario where choosing an algorithm with a higher time complexity may be justified.
30. How does the choice of asymptotic notation affect algorithmic analysis for real-world applications?
31. What are Disjoint Sets?

32. What are the key operations on Disjoint Sets?

33. What are the Union and Find algorithms in Disjoint Sets?

34. How does the Union operation work in Disjoint Sets?

35. What is the Find operation's significance in Disjoint Sets?

36. How do Disjoint Sets find applications in real-world scenarios?

37. Discuss the efficiency considerations in Disjoint Set operations.

38. How does Disjoint Set Union by Rank optimize the Union operation?

39. Explain the process of path compression in Disjoint Sets.

40. How do Disjoint Sets facilitate cycle detection in graphs?

41. Discuss the time complexity of basic operations in Disjoint Sets.

42. How are Disjoint Sets employed in image processing for segmentation tasks?

43. Explain the concept of Union by Size in Disjoint Sets.

44. How are Disjoint Sets utilized in Kruskal's algorithm for finding minimum spanning trees?

45. What is Backtracking and how does it work as a general method?

46. Discuss some common applications of Backtracking.

47. Explain the n-Queen's problem and how it is solved using Backtracking.

48. Discuss the Sum of Subsets problem and its solution using Backtracking.

49. Explain how Backtracking is used in graph colouring problems.

50. Discuss the time complexity of Backtracking algorithms.

51. Explain how Backtracking is used in solving Sudoku puzzles.

52. Discuss the role of pruning in improving the efficiency of Backtracking algorithms.

53. Explain how Backtracking is applied to solve the Traveling Salesman Problem (TSP).

54. Discuss the importance of backtracking order in solving combinatorial problems.

55. Explain how Backtracking is utilized in generating all permutations of a given set.

56. Discuss the trade-offs between recursion and iteration in implementing Backtracking algorithms.

57. Explain how Backtracking can be applied to solve the Knight's Tour problem.

58. Discuss the challenges of implementing Backtracking algorithms for large problem instances.

59. Explain how Backtracking can be applied to solve the Subset Sum problem.

60. Explore the trade-offs between using recursive backtracking and iterative approaches for solving combinatorial optimization problems.

61. What is Dynamic Programming and how is it different from brute force?

62. Explain the concept of memorization in Dynamic Programming.

63. What are some common applications of Dynamic Programming?

64. How does Dynamic Programming help in solving the Optimal Binary Search Tree problem?

65. What is the 0/1 knapsack problem, and how can Dynamic Programming solve it?

66. Explain how Dynamic Programming can be applied to the All-Pairs Shortest Path problem.

67. How is Dynamic Programming used to tackle the Traveling Salesperson problem?

68. In the context of Dynamic Programming, what is Reliability Design?

69. What are the key steps involved in solving a problem using Dynamic Programming?

70. How can you determine the time complexity of a Dynamic Programming solution?

71. What is the principle of optimality, and how does it relate to Dynamic Programming?

72. Can Dynamic Programming be applied to problems with overlapping subproblems but without optimal substructure?

73. How do you decide whether to use top-down (memorization) or bottom-up (tabulation) Dynamic Programming?

74. Explain the concept of state transition in Dynamic Programming.

75. What is the principle of optimality, and how does it relate to Dynamic Programming?