## Long Questions

1. What is the definition of AVL Trees and what distinguishes them from other types of binary search trees?

2. How is the height of an AVL Tree maintained to ensure balanced structure, and why is balancedness important?

3. Walk us through the process of inserting a new node into an AVL Tree while maintaining its AVL property.

4. Explain the steps involved in deleting a node from an AVL Tree while ensuring that the tree remains balanced.

5. How does searching for a specific element in an AVL Tree differ from searching in other types of binary search trees?

6. What are Red-Black Trees, and how do they compare to AVL Trees in terms of balancing criteria and performance?

7. Discuss the advantages and disadvantages of using Red-Black Trees over AVL Trees in certain scenarios.

8. Can you illustrate the rotation operations used in AVL Trees to maintain balance after insertion and deletion?

9. How does the process of rebalancing in AVL Trees impact the time complexity of insertion and deletion operations?

10. Explain the concept of Splay Trees and how they differ from both AVL Trees and Red-Black Trees.

11. What are the primary applications of Splay Trees, and in what scenarios are they most beneficial?

12. Compare and contrast the performance characteristics of AVL Trees, Red-Black Trees, and Splay Trees

13. What are the key differences between adjacency matrix and adjacency list implementations in graph data structures?

14. How does depth-first search (DFS) differ from breadth-first search (BFS) in terms of graph traversal methods?

15. Can you explain the process of heapification and its significance in heap sort?

16. What are the advantages and disadvantages of using heap sort compared to other sorting algorithms like quicksort or mergesort?

17. In the context of external sorting, what challenges does limited primary memory pose, and how do external sorting algorithms address them?

18. How does the two-phase approach in external sorting, involving sorting and merging, contribute to efficiency and scalability?

19. Describe the model for external sorting and its components, such as input buffers, output buffers, and secondary storage.

20. What are the main steps involved in the merge sort algorithm, and how does it achieve its time complexity of O(n log n)?

21. How does merge sort perform in terms of stability compared to other sorting algorithms like quicksort or heapsort?

22. Can you explain the concept of divide and conquer in the context of merge sort, and how it facilitates the sorting process?

23. What are the primary differences between merge sort and quicksort, and under what circumstances would you prefer one over the other?

24. How does merge sort handle scenarios where the data set is too large to fit into memory entirely?

25. What are some strategies for optimising the performance of merge sort, particularly when dealing with large datasets?

26. How does the concept of partitioning contribute to the efficiency of external sorting algorithms?

27. Can you discuss the trade-offs involved in choosing between internal and external sorting methods?

28. What are some common applications of graph traversal algorithms in real-world scenarios, and how do they contribute to solving problems

29. Explain the concept of a min-heap and a max-heap, and discuss their applications in algorithms beyond sorting, such as priority queues and Dijkstra's algorithm.

30. Compare and contrast the time and space complexities of heap sort and merge sort, and discuss how these complexities impact their performance in different scenarios.

31. How does the concept of "in-place" sorting apply to algorithms like heap sort and merge sort?

32. Discuss the role of balanced trees, such as AVL trees or red-black trees, in facilitating efficient external sorting operations?

33. What are some practical considerations when choosing between iterative and recursive implementations of graph traversal algorithms like DFS and BFS?

34. Can you explain the concept of stability in sorting algorithms, and why it is important in certain applications such as sorting by multiple criteria or preserving the original order of equal elements?

35. Discuss the impact of input data distribution on the performance of sorting algorithms, and how algorithms like quicksort and merge sort adapt to different types of input distributions.

36. Can you discuss the trade-offs considering factors such as memory usage, performance, and scalability?

37. What is pattern matching, and why is it important in computer science?

38. Explain the brute force approach to pattern matching. How does it work, and what are its main limitations?

39. Describe the Boyer-Moore pattern matching algorithm. How does it improve upon the brute force approach?

40. What are the key components of the Boyer-Moore algorithm that allow it to skip sections of the text?

41. Outline the Knuth-Morris-Pratt (KMP) algorithm. How does it ensure efficient pattern matching?

42. Complexity: The preprocessing phase has a time complexity of O(m), where m is the length of the pattern, and the search phase has a time complexity of O(n), where n is the length of the text, making the overall time complexity O(n+m).

43. Compare and contrast the Boyer-Moore and Knuth-Morris-Pratt algorithms in terms of time complexity and space complexity.

44. What is a trie, and how is it used in pattern matching?

45. Explain the structure and key features of standard tries. How are they implemented?

46. How do compressed tries differ from standard tries, and what advantages do they offer?

47. Describe the concept of suffix tries. How do they assist in pattern matching and string processing tasks?

48. Provide an example where the Boyer-Moore algorithm would significantly outperform the brute force approach in pattern matching.

49. Discuss the preprocessing steps involved in the Knuth-Morris-Pratt algorithm. How do they contribute to its efficiency?

50. How can suffix tries be used in genome sequencing applications?

51. What are the space complexities of standard tries, compressed tries, and suffix tries? Compare them.

52. How does the Boyer-Moore algorithm handle cases with repetitive patterns?

53. Explain the concept of the prefix function in the KMP algorithm. How is it calculated?

54. Provide an example where compressed tries would be more efficient than standard tries in terms of memory usage.

55. In what situations would the brute force pattern matching algorithm be preferred over the Boyer-Moore or KMP algorithms?

56. How do suffix tries facilitate substring searches within a given string?

57. How do preprocessing steps in pattern matching algorithms like Boyer-Moore and KMP reduce overall search time?

58. What are the challenges in implementing compressed tries for large datasets?

59. Explain how pattern matching algorithms can be optimized for searching in Unicode text.

60. Compare the efficiency of suffix tries against other data structures for string matching tasks.

61. Discuss future trends in pattern matching and tries. How might these algorithms and data structures evolve?

62. How do separate chaining and linear probing differ in resolving collisions in a hash table?

63. What are the key differences between quadratic probing, double hashing, and extendible hashing in managing collisions and table resizing?

64. Compare adjacency matrix vs. adjacency list for graph representation.

65. Contrast DFS and BFS in graph traversal scenarios.

66. Describe heap sort's mechanism and its time complexity.

67. What is external sorting, and what challenges does it address?

68. How is merge sort used in external sorting processes?

69. What is pattern matching, and why is it important in computer science?

70. Explain the brute force approach to pattern matching. How does it work, and what are its main limitations?

71. Explain the structure and key features of standard tries. How are they implemented?

72. Describe the concept of suffix tries. How do they assist in pattern matching and string processing tasks?

73. Provide an example where the Boyer-Moore algorithm would

    significantly outperform the brute force approach in pattern matching.