Short Questions

1. What is a data structure?
2. Define an abstract data type (ADT).
3. List four basic types of data structures.
4. What is the difference between primitive and non-primitive data structures?
5. How do data structures improve the efficiency of computer programs?
6. What is a linear list?
7. Explain the concept of a singly linked list.
8. How does a singly linked list differ from an array?
9. What operations can be performed on a linear list?
10. Describe the process of inserting a new element into a singly linked list.
11. How is a node in a singly linked list represented in C?
12. What does the ‘head’ pointer represent in a linked list?
13. How do you search for an element in a singly linked list?
14. Describe the process of deleting a node from a singly linked list.
15. What are the advantages of using a linked list over an array?
16. Explain how to insert a node at the beginning of a linked list.
17. How can a node be inserted at the end of a linked list?
18. Describe how to delete the first node of a linked list.
19. Explain the deletion of the last node in a linked list.
20. How do you perform a search operation in a linked list?
21. What is a stack?
22. List the basic operations performed on a stack.
23. Explain the LIFO principle with an example.
24. How is a stack implemented using an array?
25. Describe how a stack can be implemented using a linked list.
26. What does PUSH operation do in a stack?
27. Explain the POP operation in a stack.
28. How do you check if a stack is full in array implementation?
29. Describe how to check if a stack is empty.
30. What is a stack overflow?
31. Compare array and linked list implementations of stacks.
32. What are the advantages of using a linked list to implement a stack?
33. How is the top element accessed in an array-based stack?
34. Explain the dynamic nature of a linked list implementation of a stack.
35. What is a real-world application of stacks?
36. How are stacks used in function calls in programming languages?
37. Explain how stacks can be used for expression evaluation.

38. What role do stacks play in undo mechanisms in software applications?
39. What is a queue?
40. Describe the FIFO principle.
41. List the basic operations of a queue.
42. How is a queue different from a stack?
43. Explain how a circular queue works.
44. Describe the ENQUEUE operation in a queue.
45. What is the DEQUEUE operation in a queue?
46. How do you check if a queue is full?
47. Explain how to check if a queue is empty.
48. What is queue overflow and underflow?
49. Compare the array and linked list implementations of queues.
50. What are the benefits of implementing queues using linked lists?
51. What is a dictionary in the context of data structures?
52. How is a dictionary implemented using a linear list?
53. Explain the concept of a skip list.
54. How does a skip list improve search efficiency?
55. What are the basic operations performed on dictionaries?
56. Describe the insertion process in a dictionary using a linear list.
57. How is deletion handled in a skip list?
58. What is a hash table?
59. Explain the role of a hash function in a hash table.
60. What is a collision in the context of hash tables?
61. Describe separate chaining as a method for collision resolution.
62. How does open addressing differ from separate chaining?
63. Explain linear probing in hash tables.
64. What is quadratic probing, and how does it work?
65. Describe double hashing as a collision resolution technique.
66. What is rehashing in hash tables?
67. Explain the concept of extendible hashing.
68. Compare linear probing and quadratic probing in terms of efficiency.
69. How does separate chaining handle collisions differently from open addressing?
70. What are the advantages of using a hash table for dictionary operations?
71. How do you choose a good hash function?
72. What are the challenges associated with hashing?
73. How can hash tables be resized, and why is this important?
74. Describe a real-world application of hash tables.
75. How does a hash table perform insertion operations?
76. Explain the deletion process in a hash table.

77. How is searching implemented in a hash table?
78. What is the load factor in the context of hash tables?
79. How does the load factor affect a hash table's performance?
80. Describe how extendible hashing dynamically adjusts to the data set size.
81. What is the significance of choosing the right probing sequence in open addressing?
82. How does double hashing minimize clustering in hash tables?
83. Compare the efficiency of separate chaining and open addressing for different load factors.
84. Why might a skip list be preferred over a traditional linked list for dictionary implementations?
85. Describe an instance where rehashing would be necessary in a hash table.
86. How can extendible hashing be beneficial for databases?
87. What is spatial locality, and why is it important in the context of hashing?
88. How does a hash table contribute to the efficiency of data retrieval?
89. Describe the process of key transformation in hashing.
90. How does a hash table support quick insertion and deletion?
91. What is the impact of hash function selection on collision frequency?
92. Explain the advantage of quadratic probing over linear probing.
93. How do dynamic hashing techniques like extendible hashing work?
94. What are the trade-offs involved in selecting a hash table size?
95. How does separate chaining allow for the direct addressing of collisions?
96. Describe how a dictionary can be implemented in a distributed system.
97. Explain the advantages of using a skip list for dictionary implementations with large data sets.
98. How does hashing facilitate faster search operations compared to other data structures?
99. What strategies can be employed to reduce the impact of collisions in a hash table?
100. Compare the performance implications of different collision resolution techniques.
101. Define a binary search tree (BST).
102. How is a binary search tree implemented?
103. Describe the process of searching for an element in a BST.
104. Explain how insertion is performed in a BST.
105. How is a node deleted from a BST?
106. What is a B-Tree and how does it differ from a BST?

107. Explain the structure of a B+ Tree.
108. What are the advantages of using AVL trees over BSTs?
109. Define the height of an AVL tree.
110. Describe the insertion process in an AVL tree.
111. How does deletion work in an AVL tree?
112. What is a Red-Black Tree?
113. Explain the properties of a Red-Black Tree.
114. How do you insert a node into a Red-Black Tree?
115. Describe the deletion process in a Red-Black Tree.
116. Define what a Splay Tree is.
117. Explain the splaying operation in Splay Trees.
118. How does searching work in a Splay Tree?
119. Compare the insertion process in BSTs and AVL trees.
120. How do B-Trees handle large amounts of data efficiently?
121. Describe the balancing mechanism of AVL trees.
122. What makes Red-Black Trees unique in handling insertions and deletions?
123. How do B+ Trees improve upon B-Trees in terms of disk access?
124. Explain the concept of rotation in AVL tree balancing.
125. What is the difference between AVL trees and Splay Trees in terms of balancing?