Short Questions

1. What is the difference between text and binary files in C?
2. How can you create a text file in C? Provide an example code snippet.
3. Explain the steps to read data from a text file in C.
4. What function is used to open a file for writing in C?
5. How do you append data to an existing file in C?
6. Describe the process of creating a binary file in C.
7. Provide an example of reading data from a binary file in C.
8. Explain the purpose of the fseek() function in C file handling.
9. How is the ftell() function used in file operations?
10. What is the role of the rewind() function in C file handling?
11. Discuss the precautions to be taken when working with binary files.
12. How do you write and read structures to/from a binary file in C? Provide a brief example.
13. Explain the concept of a text file and its characteristics.
14. Differentiate between the 'r' and 'w' modes when opening a file in C.
15. How does the fclose() function contribute to proper file handling in C?
16. Discuss the significance of the feof() function in file operations.
17. What is the purpose of the 'a' mode when opening a file in C?
18. Explain the role of the fprintf() function in writing to a text file.
19. How can you check if a file has been successfully opened in C?
20. Discuss the importance of error handling when working with files in C.
21. Explain the use of the fputc() function in writing characters to a file.
22. What is the significance of the 'b' mode in file handling? Provide an example.

23. Discuss the potential issues associated with file operations in C.
24. How do you handle random access in a file using fseek() and ftell()?
25. Provide an example of appending data to an existing text file in C.
26. Explain the importance of designing structured programs.
27. What is the purpose of declaring a function in C?
28. Describe the signature of a function and its components.
29. Discuss the role of parameters and return types in defining a function.
30. Explain the concept of passing parameters to functions in C.
31. Differentiate between call by value and call by reference in function calls.
32. How are arrays passed to functions in C? Provide an example.
33. Discuss the idea of passing pointers to functions in C.
34. What is call by reference, and why is it important in C programming?
35. Provide examples of some standard C functions and libraries.
36. Explain the process of implementing recursion in programming.
37. Provide a simple program using recursion to find the factorial of a number.
38. What is the Fibonacci series, and how is it computed using recursion?
39. Discuss the limitations of recursive functions in C.
40. How can you avoid stack overflow when using recursion?
41. Explain the concept of a recursive function and its characteristics.
42. Describe the role of the base case in recursive functions.
43. What happens during the recursive call in a function?
44. Differentiate between direct and indirect recursion.
45. How does recursion contribute to solving complex problems in programming?
46. Discuss the importance of termination conditions in recursive functions.

47. Provide an example of a recursive program to calculate the power of a number.
48. Explain the significance of tail recursion and its advantages.
49. Discuss scenarios where recursion is preferred over iterative solutions.
50. How can recursion be utilized in searching and sorting algorithms?
51. Explain the concept of dynamic memory allocation in C.
52. What is the significance of allocating memory dynamically?
53. How is memory allocated dynamically using the malloc function?
54. Discuss the role of the free function in dynamic memory allocation.
55. Differentiate between stack and heap memory.
56. What issues can arise from not freeing dynamically allocated memory?
57. Explain the process of allocating memory for an integer dynamically.
58. How can you allocate memory for an array of integers dynamically?
59. Discuss the role of the calloc function in dynamic memory allocation.
60. What is the key difference between malloc and calloc?
61. How do you free memory allocated for an array of structures?
62. Explain the purpose of the realloc function in dynamic memory allocation.
63. Discuss potential memory-related errors in C programming.
64. How is memory deallocated when using the realloc function?
65. Provide an example of dynamically allocating memory for a character array.
66. Discuss scenarios where dynamic memory allocation is preferable.
67. What is the role of the sizeof operator in dynamic memory allocation?
68. Explain the importance of checking the return value of memory allocation functions.
69. How can dynamic memory allocation prevent fixed-size limitations?
70. Discuss the significance of avoiding memory leaks in programming.

71. What challenges may arise from improper use of dynamic memory?
72. Provide an example of allocating memory for a 2D array dynamically.
73. How do pointers play a crucial role in dynamic memory allocation?
74. Explain the concept of freeing memory before program termination.
75. Discuss the role of dynamic memory allocation in creating flexible data structures.
76. Explain the algorithm for finding roots of a quadratic equation.
77. Discuss the steps involved in finding the minimum and maximum numbers in a given set.
78. Provide a basic algorithm to determine if a number is a prime number.
79. What is the significance of searching algorithms in programming?
80. Differentiate between linear and binary search techniques.
81. Explain the basic steps of a linear search algorithm.
82. Discuss the concept of a binary search algorithm.
83. How does a binary search algorithm work on a sorted array?
84. What are the limitations of linear search in large datasets?
85. Provide an example scenario where linear search is more suitable than binary search.
86. Explain the Bubble Sort algorithm for sorting an array.
87. Discuss the steps involved in the Insertion Sort algorithm.
88. Provide a basic overview of the Selection Sort algorithm.
89. What is the significance of sorting algorithms in data processing?
90. Differentiate between stable and unstable sorting algorithms.
91. Explain the order of complexity and its importance in algorithm analysis.
92. Provide an example of an algorithm with constant time complexity.
93. Discuss the concept of linear time complexity in algorithms.

94. How does an algorithm with logarithmic time complexity behave?

95. Explain the role of quadratic time complexity in algorithmic analysis.

96. What is the significance of polynomial time complexity in algorithms?

97. Discuss the characteristics of algorithms with exponential time complexity.

98. Explain the concept of big-O notation in algorithm analysis.

99. Discuss scenarios where quicksort is preferred over mergesort.

100. Explain the divide-and-conquer strategy in algorithm design.

101. What is the role of recursion in certain sorting algorithms?

102. Discuss the space complexity of an algorithm.

103. How does the efficiency of an algorithm impact overall program performance?

104. Provide an example of a real-world scenario where quicksort is advantageous.

105. Explain the concept of stable sorting algorithms.

106. Discuss the steps involved in the merge sort algorithm.

107. Provide an example scenario where selection sort is more efficient than quicksort.

108. Explain the concept of in-place sorting algorithms.

109. Discuss the role of the partitioning step in quicksort.

110. Explain the time complexity of the quicksort algorithm.

111. How does a divide-and-conquer algorithm differ from a dynamic programming approach?

112. Provide an example of a scenario where mergesort is preferable over quicksort.

113. Explain the significance of the pivot element in quicksort.

114. Discuss the advantages and disadvantages of the bubble sort algorithm.

115. What is the importance of stability in sorting algorithms?

116. Explain the role of the selection step in the selection sort algorithm.

117. Provide an example of a scenario where bubble sort is efficient.

118. Discuss the basic steps of the insertion sort algorithm.

119. Explain the concept of time complexity trade-offs in algorithm design.

120. How does the efficiency of a sorting algorithm impact real-time applications?

121. Discuss the steps involved in the radix sort algorithm.

122. Explain the importance of choosing an appropriate sorting algorithm based on input data.

123. Provide an example of a scenario where insertion sort outperforms quicksort.

124. Discuss the role of the heap data structure in heap sort.

125. Explain the concept of adaptive sorting algorithms and their benefits.