DATA TYPES AND STRUCTURES

Section A

1. Define the term data type and list three examples.
2. Explain the difference between integer and real data types.
3. What is a record, and how does it differ from an array?
4. Write pseudocode to declare a record type for a book with title, author, and year.
5. How would you store a sequence of names in a 1D array?
6. Explain what Boolean data types are and give an example of their use.
7. Write an algorithm to find the smallest element in an array of integers.
8. What is a linear search, and how does it work?
9. Describe the purpose of the upper bound in an array.
10. How does bubble sort work in sorting an array?
11. Explain the use of a file in storing data permanently.
12. Describe LIFO and give an example of where it might be used.
13. What is a queue, and how does it operate?
14. Write pseudocode for creating a 2D array.
15. What is a linked list, and how does it differ from an array?
16. Explain the FIFO principle with a real-world example.
17. Define an abstract data type (ADT) and list two examples.
18. Describe the push and pop operations in a stack.
19. What is the role of the front pointer in a queue?
20. Create a flowchart for a linear search algorithm.
21. Write pseudocode to declare an integer array with ten elements.
22. How does a composite data type differ from a basic data type?
23. Define stepwise refinement and give an example.
24. Explain the importance of input validation.
25. Write pseudocode for a bubble sort algorithm.
26. Define array index and explain its purpose.
27. What are pointers, and why are they essential in linked lists?
28. Explain the enqueue and dequeue operations in queues.

29. Describe how to append a line to a text file in pseudocode.
30. What is the end of file (EOF) function used for?
31. Write pseudocode to find if a number exists in a 1D array.
32. How is a 2D array structured differently from a 1D array?
33. Describe the start pointer in a linked list.
34. What is the difference between declaring and initializing an array?
35. Explain why stack overflow might occur.
36. Write pseudocode to check if a stack is empty.
37. Describe the purpose of upper and lower bounds in an array.
38. What is a flag variable and give an example of its use in pseudocode.
39. Write pseudocode to reverse an array of integers.
40. How does a circular queue operate differently from a standard queue?
41. Explain the purpose of the heap in a linked list.
42. Describe how a node is structured in a linked list.
43. Write pseudocode to insert an item at the start of a linked list.
44. What is a null pointer, and how is it used in linked lists?
45. Explain the purpose of the heapStartPointer in a linked list.
46. Write pseudocode for setting up a stack using an array.
47. Describe the top pointer in a stack.
48. Explain why we use abstract data types in computer science.
49. Write pseudocode for inserting a value in a sorted linked list.
50. What is the difference between linear search and binary search?

Section B

1. Write pseudocode to implement a circular queue using an array.
2. Describe how a bubble sort operates and explain its time complexity.
3. Implement a linear search algorithm and count the number of comparisons.
4. Explain how a file can be read line-by-line in pseudocode.
5. Write pseudocode to merge two sorted arrays.
6. Describe a use case for a stack in a real-world application.
7. Implement a stack using an array in pseudocode and handle overflow.
8. Write pseudocode for a binary search in a sorted array.
9. Explain the advantages of using linked lists over arrays.
10. Write pseudocode for a recursive function to calculate the factorial of a number.
11. Implement a selection sort algorithm in pseudocode.
12. Write pseudocode to swap two elements in an array.
13. Describe memory management in the context of stacks.
14. Write a program to check if two linked lists intersect.
15. Explain the heap in a linked list and how it is managed.
16. Write pseudocode to reverse a queue using a stack.
17. Describe stepwise refinement in creating a sorting algorithm.
18. Create an algorithm to search for a value in a 2D array.
19. Explain how EOF can be used in reading a file.
20. Write pseudocode for a queue implemented with pointers.
21. Describe how the heap is used to manage memory in linked lists.
22. Write pseudocode to delete a node from a linked list.
23. Implement a bubble sort in Python or JavaScript.
24. What is Big O notation, and why is it important in sorting?
25. Implement insertion sort on an array of strings.
26. Write pseudocode for a binary search tree (BST) insertion.
27. Implement depth-first search (DFS) for a linked list.
28. Explain how a binary tree is structured and used.
29. Write pseudocode to find the median of an array.
30. Design a function to add a record to a file in pseudocode.

31. Explain how composite data types can simplify complex data.
32. Describe pass-by-value vs pass-by-reference with pseudocode.
33. Implement a linked list with pointers.
34. Write pseudocode for finding the longest string in an array.
35. Explain recursive data structures with an example.
36. Implement a hash map for storing records.
37. Write a program to sort a 2D array by rows.
38. Implement breadth-first search (BFS) for a queue.
39. Design an algorithm to check if a list is sorted.
40. Write pseudocode to traverse a linked list in reverse order.
41. Explain control structures and their importance.
42. Create a hash function and explain its purpose.
43. Write pseudocode to count occurrences of an item in a list.
44. Implement Fibonacci sequence generation in pseudocode.
45. Write a function to remove duplicates in a sorted array.
46. Explain the difference between mutable and immutable data types.
47. Describe how to manage overflow in a queue.
48. Write pseudocode to generate all permutations of a list.
49. Implement Dijkstra's algorithm for shortest path finding.
50. Explain sorting algorithms' trade-offs in terms of efficiency.

Section C

1. Write pseudocode to implement a priority queue using a linked list.
2. Describe how stacks are used in recursive calls with an example.
3. Write pseudocode for quicksort and analyze its complexity.
4. Implement a binary tree traversal in Python.
5. Explain Big O notation for sorting and give examples.
6. Write a recursive function to check if a linked list is a palindrome.
7. Describe the heap sort algorithm and its efficiency.
8. Implement merge sort on an array.
9. Write pseudocode to find the longest path in a binary tree.
10. Explain garbage collection in linked lists.
11. Implement DFS and BFS for a graph represented by adjacency lists.
12. Write pseudocode to invert a binary tree.
13. Describe graph traversal algorithms and their applications.
14. Implement Prim's algorithm to find minimum spanning trees.
15. Write a program to check if a graph is bipartite.
16. Implement a dynamic array with automatic resizing.
17. Write pseudocode for finding shortest paths in a grid.
18. Describe the Bellman-Ford algorithm for shortest paths.
19. Implement a binary search tree with deletion.
20. Explain time-space trade-offs in recursive algorithms.
21. Write pseudocode for generating random numbers within a range.
22. Implement topological sorting of a directed acyclic graph.
23. Describe hash functions and collision resolution.
24. Write pseudocode for backtracking algorithm to solve n-Queens.
25. Explain divide and conquer with examples.
26. Implement binary heap for a priority queue.
27. Write pseudocode to find connected components in a graph.
28. Describe Kruskal's algorithm for minimum spanning trees.
29. Implement queue with linked list and handle underflow.
30. Write pseudocode to merge k sorted lists.
31. Explain the importance of memory management in data structures.

32. Implement a stack to reverse a string using recursion.
33. Write pseudocode to solve a maze using DFS.
34. Implement longest common subsequence (LCS) algorithm.
35. Describe convex hull algorithms.
36. Implement a cycle detection algorithm in a graph.
37. Write pseudocode for dynamic programming for knapsack problem.
38. Explain probabilistic data structures like Bloom filters.
39. Implement a skip list.
40. Write pseudocode to find strongly connected components.
41. Explain the concept of amortized analysis.
42. Implement cache replacement policies (e.g., LRU).
43. Describe parallel algorithms and their applications.
44. Implement a trie data structure for string storage.
45. Write pseudocode for bucket sort.
46. Implement A search algorithm* for pathfinding.
47. Describe sorting network algorithms.
48. Write a function to detect cycles in a directed graph.
49. Implement Radix Sort for integer sorting.
50. Explain computational complexity theory and its significance.