**Basic Concepts:**

**1. Write a program to find the factorial of a number.**

def factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n-1)

number = int(input("Enter a number: "))

result = factorial(number)

print(f"The factorial of {number} is {result}")

**2. Implement a function to check if a number is prime.**

**3. Write a program to reverse a string.**

**4. Create a function to check if a string is a palindrome.**

**5. Implement a binary search algorithm.**

**6. Create a function to find the nth Fibonacci number.**

**7. Write a program to sort an array of integers.**

**8. Create a function to find the largest element in an array.**

**9. Implement depth-first search (DFS) and breadth-first search (BFS) for a graph.**

**10. Implement a stack using arrays or linked lists.**

**11. Implement a queue using stacks.**

**12. Write a program to merge two sorted linked lists.**

**13. Create a function to convert a decimal number to binary.**

**14. Write a program to find the intersection of two arrays.**

**15. Write a program to count the number of set bits in a number (Hamming weight).**

**16. Create a function to reverse words in a string.**

**17. Write a program to perform matrix multiplication.**

**18. Create a function to check if a number is a power of two.**

**19. Write a program to serialize and deserialize a binary tree.**

**20. Implement a hash table with collision resolution.**

**Data Structures:**

**21. Implement a binary search tree (BST) and perform insertion, deletion, and search operations.**

**22. Implement a trie data structure.**

**23. Implement a priority queue.**

**24. Implement an algorithm to detect a cycle in a linked list.**

**25. Write a program to detect and remove a loop in a linked list.**

**26. Implement an algorithm to perform in-place quicksort for an array.**

**27. Implement an algorithm to perform binary tree pruning.**

**28. Implement a stack using arrays or linked lists.**

**29. Implement a queue using stacks.**

**30. Create a function to reverse a linked list.**

**31. Implement a binary search algorithm.**

**32. Implement depth-first search (DFS) and breadth-first search (BFS) for a graph.**

**33. Implement an algorithm to find the strongly connected components (SCCs) of a directed graph using Tarjan's or Kosaraju's algorithm.**

**34. Implement a hash table with collision resolution.**

**35. Implement a trie data structure.**

**Algorithms:**

**36. Implement sorting algorithms: quicksort, mergesort, heapsort, etc.**

**37. Implement Dijkstra's algorithm for finding the shortest path in a graph.**

**38. Implement the Sieve of Eratosthenes algorithm to find all prime numbers up to a given limit.**

**39. Implement the Knuth-Morris-Pratt (KMP) algorithm for string searching.**

**40. Implement the Floyd-Warshall algorithm for all pairs shortest paths.**

**41. Implement the Ford-Fulkerson method or Edmonds-Karp algorithm to find the maximum flow in a flow network.**

**42. Implement the Hopcroft-Karp algorithm to find the maximum matching in a bipartite graph.**

**43. Implement dynamic programming algorithms: subset sum problem, longest common subsequence (LCS), longest increasing subsequence (LIS), etc.**

**44. Implement backtracking algorithms: N-Queens problem, subset sum problem, etc.**

**45. Implement the Graham scan algorithm to compute the convex hull of a set of points.**

**46. Implement the Boyer-Moore algorithm for string searching.**

**47. Implement algorithms for finding articulation points, bridges, and cycles in graphs.**

**String Manipulation:**

**48. Create a function to find the longest common prefix in an array of strings.**

**49. Write a program to check if two strings are anagrams of each other.**

**50. Create a function to find all permutations of a string.**

**51. Write a program to find the longest palindromic substring.**

**52. Create a function to reverse words in a string.**

**53. Write a program to find the longest substring without repeating characters.**

**54. Create a function to compute the edit distance (Levenshtein distance) between two strings.**

**55. Create a function to check if a given string is a valid parentheses expression.**

**56. Write a program to reverse the order of words in a sentence.**

**57. Create a function to check if a given string is an interleaving of two other strings.**

**58. Implement an algorithm to solve the longest common substring of two strings using suffix arrays or dynamic programming.**

**59. Write a program to generate all valid IP addresses from a given string.**

**Graph Theory:**

**60. Implement depth-first search (DFS) and breadth-first search (BFS) for a graph.**

**61. Write a program to find the shortest path in a maze using BFS or DFS.**

**62. Implement an algorithm to find the maximum flow in a flow network using the Ford-Fulkerson method or Edmonds-Karp algorithm.**

**63. Write a program to check if a given graph is bipartite.**

**64. Implement an algorithm to find the articulation points (cut vertices) and bridges (cut edges) in a graph.**

**65. Write a program to solve the N-Queens problem.**

**66. Implement an algorithm to find the strongly connected components (SCCs) of a directed graph using Tarjan's or Kosaraju's algorithm.**

**67. Write a program to find the longest path in a directed acyclic graph (DAG).**

**68. Implement an algorithm to find the minimum spanning tree (MST) of a weighted graph using Prim's or Kruskal's algorithm.**

**69. Write a program to perform topological sorting of a directed acyclic graph (DAG).**

**70. Implement an algorithm to detect a cycle in a directed graph.**

**71. Implement an algorithm to detect a cycle in an undirected graph.**

**72. Implement an algorithm to find the all-pairs shortest paths using the Floyd-Warshall algorithm with path reconstruction.**

**73. Implement an algorithm to solve the longest path problem in a directed acyclic graph (DAG) using dynamic programming.**

**74. Implement algorithms for finding articulation points, bridges, and cycles in graphs.**

**75. Implement an algorithm to solve the maximum bipartite matching problem using augmenting paths.**

**Dynamic Programming:**

**76. Implement dynamic programming algorithms: subset sum problem, longest common subsequence (LCS), longest increasing subsequence (LIS), etc.**

**77. Implement an algorithm to solve the traveling salesman problem (TSP) using dynamic programming or branch and bound.**

**78. Implement an algorithm to solve the 0/1 knapsack problem using dynamic programming.**

**79. Implement an algorithm to solve the longest common increasing subsequence (LCIS) problem.**

**80. Implement an algorithm to solve the subset sum problem using dynamic programming or backtracking.**

**81. Write a program to find the maximum subarray sum.**

**82. Create a function to find the longest increasing subsequence in an array.**

**83. Write a program to find the maximum product subarray.**

**84. Implement an algorithm to solve the longest path problem in a directed acyclic graph (DAG) using dynamic programming.**

**85. Implement dynamic programming algorithms: subset sum problem, longest common subsequence (LCS), longest increasing subsequence (LIS), etc.**

**86. Implement an algorithm to solve the longest common increasing subsequence (LCIS) problem.**

**87. Implement an algorithm to solve the subset sum problem using dynamic programming or backtracking.**

**88. Write a program to find the maximum sum of a subarray with at most k elements.**

**89. Implement an algorithm to solve the longest common substring of two strings using suffix arrays or dynamic programming.**

**90. Implement an algorithm to solve the longest palindromic substring problem using dynamic programming.**

**Tree and Graph Traversal:**

**91. Implement depth-first search (DFS) and breadth-first search (BFS) for a graph.**

**92. Write a program to solve the N-Queens problem.**

**93. Implement an algorithm to find the shortest path in a maze using BFS or DFS.**

**94. Write a program to find the longest path in a directed acyclic graph (DAG).**

**95. Implement an algorithm to find the maximum matching in a bipartite graph using the Hopcroft-Karp algorithm.**

**96. Write a program to perform topological sorting of a directed acyclic graph (DAG).**

**97. Write a program to find the maximum path sum in a binary tree.**

**98. Create a function to check if a given binary tree is balanced.**

**99. Write a program to find the kth smallest element in a binary search tree.**

**100. Implement an algorithm to solve the maximum independent set in a binary tree.**