**1. Linear Search Algorithm**

Write a program that:

1. Takes a list of integers and a target number as input.
2. Implements **linear search** to find the index of the target number in the list.
3. If the target is not found, print “Not Found.”  
   **Concepts:** Lists, loops, searching algorithms.

**2. Bubble Sort**

Write a program that:

1. Takes a list of integers as input from the user.
2. Implements **bubble sort** to sort the list in ascending order.
3. Print the sorted list.  
   **Concepts:** Lists, nested loops, sorting algorithms.

**3. Remove Duplicates from a List**

Write a program that:

1. Takes a list of integers with duplicates as input.
2. Removes duplicate elements using a **set** and prints the unique elements.  
   **Concepts:** Lists, sets, loops.

**4. Merge Two Sorted Lists**

Write a program that:

1. Takes two sorted lists as input.
2. Merges them into a single sorted list without using the sort() function.  
   **Concepts:** Lists, sorting algorithms, loops.

**5. Stack Implementation**

Implement a **stack** using a list:

1. Provide the following operations: push, pop, peek, and is\_empty.
2. Allow the user to perform these operations interactively.  
   **Concepts:** Lists, stack data structure.

**6. Queue Implementation**

Implement a **queue** using a list:

1. Provide the following operations: enqueue, dequeue, peek, and is\_empty.
2. Allow the user to interactively perform these operations.  
   **Concepts:** Lists, queue data structure.

**7. Find the Intersection of Two Lists**

Write a program that:

1. Takes two lists as input.
2. Finds the common elements (intersection) between the two lists using sets.  
   **Concepts:** Lists, sets, set operations.

**8. Implement Binary Search**

Write a program that:

1. Takes a sorted list of integers and a target number as input.
2. Implements binary search to find the target number in the list.
3. Prints the index of the target or -1 if it’s not found.  
   **Concepts:** Lists, binary search, recursion (optional).

**9. Sort a List of Strings by Length**

Write a program that:

1. Takes a list of strings as input.
2. Sorts the strings in ascending order of their lengths.
3. Print the sorted list.  
   **Concepts:** Lists, sorting, lambda functions.

**10. Find All Pairs with a Given Sum**

Write a program that:

1. Takes a list of integers and a target sum as input.
2. Finds all unique pairs of numbers in the list that add up to the target sum.
3. Print the pairs.  
   **Concepts:** Lists, dictionaries, loops.

**11. Find the Median of a List**

Write a program that:

1. Takes a list of integers as input.
2. Finds and prints the **median** of the list.
   * If the list has an odd number of elements, the median is the middle element.
   * If the list has an even number of elements, the median is the average of the two middle elements.  
     **Concepts:** Lists, sorting.

**12. Find Intersection of Two Arrays**

Write a program that:

1. Takes two lists of integers as input.
2. Finds the intersection of the two lists (common elements).
3. Print the result.  
   **Concepts:** Sets, lists.

**13. Find Duplicates in a List**

Write a program that:

1. Takes a list of integers as input.
2. Finds and prints all duplicate elements in the list.  
   **Concepts:** Lists, dictionaries, sets.

**14. Implement Depth-First Search (DFS)**

Write a program that:

1. Takes a graph (adjacency list) as input.
2. Performs a depth-first search starting from a given node.
3. Prints the traversal order.  
   **Concepts:** Graphs, recursion, stack.

**15. Implement Breadth-First Search (BFS)**

Write a program that:

1. Takes a graph (adjacency list) as input.
2. Performs a breadth-first search starting from a given node.
3. Prints the traversal order.  
   **Concepts:** Graphs, queue.

**38. Implement a Hash Map**

Create a basic **hash map** using lists:

1. Use modulo-based hashing for keys.
2. Handle collisions using separate chaining (linked lists).
3. Provide operations like insert, delete, and search.  
   **Concepts:** Lists, dictionaries.

**39. Count Inversions in an Array**

Write a program that:

1. Takes a list of integers as input.
2. Counts the number of inversions (i.e., pairs (i, j) such that i < j and arr[i] > arr[j]).
3. Use a divide-and-conquer approach to count inversions efficiently.  
   **Concepts:** Lists, recursion, sorting.

**63. Evaluate a Postfix Expression**

Write a program that:

1. Takes a postfix expression (e.g., "2 3 + 5 \*") as input.
2. Evaluates and prints the result.  
   **Concepts:** Stacks, strings.

**71. Implement a Snake Game**

Simulate a basic **snake game**:

1. Use a grid where the snake moves according to user input (w, a, s, d).
2. Allow the snake to grow when it eats food and end the game if it collides with itself or the wall.  
   **Concepts:** Lists, matrices, loops.