C/C++ Programming Excercise

Programs based on Array:

1. **Array Declaration and Initialization:**
    - How do you declare an array in C?
    - What is the syntax for initializing array elements at the time of declaration?
    - Can you declare an array without specifying its size initially?
2. **Array Operations:**
    - **Sum of Array Elements:**
        1. Write a C program to calculate the sum of elements in an integer array.
        2. How would you modify the program to handle arrays of different sizes?
        3. Can you implement the sum of array elements using recursion?
    - **Maximum Element in Array:**
        1. Describe a method to find the maximum element in an integer array.
        2. Discuss the efficiency of different approaches for finding the maximum element.
        3. Is it possible to find the maximum element in an array without iterating through all elements?
    - **Linear Search in Array:**
        1. Explain the concept of linear search in arrays.
        2. Write a C program to perform linear search for a given element in an array.
        3. What are the advantages and disadvantages of linear search compared to other search algorithms?
3. **Advanced Array Algorithms:**
    - **Sorting Algorithms:**

      (A) Bubble Sort:
        1. What is the Bubble Sort algorithm, and how does it work?
        2. Implement a Bubble Sort algorithm to sort integer elements in an array.
        3. An elementary sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.
        4. Discuss the time complexity of Bubble Sort and possible optimizations.

      (B) Selection Sort:
        1. Sorts an array by repeatedly finding the minimum element from the unsorted part and putting it at the beginning.

      (C) Insertion Sort:

    1. Builds the final sorted array one item at a time by repeatedly taking the next element from the unsorted part and inserting it into its correct position in the sorted part.

  (D) Merge Sort:

    1. Divides the array into two halves, sorts each half separately, and then merges them back together.

  (E) Quick Sort:

    1. Chooses a pivot element and partitions the array into two sub-arrays according to the pivot. It then recursively sorts the sub-arrays.

- **Searching Algorithms:**

  (A) Linear Search:

    1. Iterates through an array sequentially to find the target element.

  (B) Binary Search:

    1. Assumes that the array is sorted and repeatedly divides the search interval in half until the target element is found.

- **Searching and Sorting Efficiency:**
  1. Compare the efficiency of various searching and sorting algorithms (e.g., binary search, merge sort, quick sort) for large datasets.
  2. Discuss the impact of data distribution on algorithm performance.
- **Parallel Array Processing:**
  1. Explore parallel processing techniques to optimize array operations on multi-core processors.
  2. Implement parallel versions of array algorithms and evaluate performance gains.
- **Sparse Arrays and Algorithms:**
  1. Investigate algorithms for handling sparse arrays efficiently.
  2. Implement and analyze the performance of sparse array operations such as addition, multiplication, and dot product.

4. Advanced Array Manipulation:

- **Array Reversal:**
  5. Implement a C program to reverse the elements of an integer array in place without using any additional array.
  6. Discuss the time and space complexity of your solution.
- **Array Rotation:**

1. Write a C function to rotate the elements of an integer array towards the right by a specified number of positions.

2. Can you optimize your solution to achieve the rotation operation in linear time complexity?

5. **Multidimensional Arrays:**
   - **Matrix Operations:**
   1. Develop C functions to perform addition, subtraction, and multiplication of two-dimensional matrices.
   2. Converts rows of a matrix into columns and vice versa.
   3. Consider error handling for matrices of incompatible dimensions.
   - **Sparse Matrix Representation:**
   1. Describe methods for representing sparse matrices efficiently in memory.
   2. Implement a C program to add two sparse matrices using appropriate data structures.

6. **Dynamic Memory Allocation and Arrays:**
   - **Dynamic Array Resizing:**
   1. Write a C program that dynamically resizes an array when it reaches its capacity.
   2. Discuss the trade-offs between static and dynamic arrays.
   - **Dynamic Multidimensional Arrays:**
   1. Implement dynamic allocation of memory for a two-dimensional array and perform matrix operations.
   2. Handle memory deallocation to avoid memory leaks.

7. **Miscellaneous Algorithms:**
   - **Counting Sort:**
     1. A non-comparison-based sorting algorithm that operates by counting the number of occurrences of each unique element in the input array.
   - **Prefix Sum (Cumulative Sum):**
     1. Computes the cumulative sum of elements of an array.
   - **Histogram:**
     1. Computes the frequency distribution of elements in an array.
   - **Sliding Window Technique:**
     1. Efficiently processes a fixed-size contiguous subarray of elements.
   - **Two Pointers Technique**:
     1. Uses two pointers to efficiently solve problems involving arrays, such as finding pairs with a given sum or removing duplicates.