```
/*

Please complete the following problems for Chapter 7.  Please call all three
functions from one main function below.  When completed, the output should
look like:

Problem 1:
locations:
           5   -4    3
           4    3   -2
          -4   -3   -1
          -9    8    6

masses:
           2
           5
           2
           1

number of data points:
n = 4

center of mass:
(1.3, 0.9, 0.0)

Problem 2:
0 1 2 3 4 5 6 7 8 9
6 was found at index 6

Problem 3:
9 8 7 6 5 4 3 2 1 0
0 1 2 3 4 5 6 7 8 9

*/

// Import standard input/output library
#include <stdio.h>
#define NUMBER_OF_DIMESIONS 3

// Function prototypes go here
void center_grav(int locationMatrix[4][3], int massVector[4], int
numberOfDataPoints);
void print_array(int myVector[], int vectorLength);
int binary_srch(int myArray[], int lengthOfArray, int elementToFind);
void bubble_sort(int arrayToSort[], int arrayLength);

// Main function goes here
int main()
{
    //Problem #1
    printf("\n Center of Gravity Problem:");
    int location_matrix[4][3] = {{5, -4, 3}, {4, 3, -2}, {-4, -3, -1}, {-9, 8, 6}};
    int mass_vector[4] = {2, 5, 2, 1};
    int number_of_data_points = 4;

    center_grav(location_matrix, mass_vector, number_of_data_points);

    //Problem 2
    int myArray[] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
    int numElements = sizeof(myArray)/sizeof(int);


    print_array(myArray, numElements);
    //Binary Search
    int ElementToSearch = 5;
    int newArray[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int newArrayLength = sizeof(newArray)/sizeof(int);
    int elementIndex = binary_srch(newArray, newArrayLength, ElementToSearch);
    if (elementIndex >= 0)
    {
        printf("Element %d was found at index %d \n", ElementToSearch,
elementIndex);
    }
    else
    {
        printf("Element %d was not found in the array. \n", ElementToSearch);
    }

    //Problem 3
    print_array(myArray, numElements);
    printf("\n");
    bubble_sort(myArray, numElements);
    printf("\n");
    print_array(myArray, numElements);

    return 0;
}


/*
Problem 1:

A point mass consists of a 3-D location and an associated mass, such as

    Location: (6, 0, -2) Mass: 3g

In a system of point masses, let p1, p2, ..., pn be the n 3-D points and m1,
m2, ..., mn be their associated masses. If m is the sum of the masses, the center
of gravity C is calculated as

    C = 1/m x (m1 x p1 + m2 x p2 + ... + mn x pn)

Write a program that calculates a center of gravity system data from a matrix
(points)
and a vector (masses). Display the location matrix, the mass vector, n, and the
center
of gravity.  The data set includes a location matrix (a matrix in which each row is
a
point), a one-dimensional array of masses, and the number of point masses, n .

Number of data points:
n = 4

Sample point data matrix
5 -4 3
4 3 -2
-4 -3 -1
-9 8 6
```

```
Sample mass data vector
2
5
2
1

The sample output should be:

locations:
    5 -4 3
    4 3 -2
    -4 -3 -1
    -9 8 6
masses:
    2
    5
    2
    1
number of data points:
n = 4
center of mass:
(1.3, 0.9, 0.0)

Implement the following function:

center_grav : Takes a location matrix, mass vector, and n value as parameters,
and calculates and returns as the function value the center of gravity
of the system (as an output parameter). The main function should call the
center_grav function, passing necessary arguments. For the point-mass system
data set, display the location matrix, the mass vector, n, and the center of
gravity.

*/

// center_grav function goes here
void center_grav(int locationMatrix[4][3], int massVector[4], int
numberOfDataPoints)
{
    //Printing the location
    printf("\n locations: \n");
    for (int iRow = 0; iRow < numberOfDataPoints; iRow++)
    {
        printf("    ");
        for (int iCol = 0; iCol < NUMBER_OF_DIMESIONS; iCol++)
        {
            printf(" %d ", locationMatrix[iRow][iCol]);
        }
        printf("\n");
    }
    //Printing the Masses
    printf("\n");
    printf("masses: \n");
    for (int i = 0; i < numberOfDataPoints; i++)
    {
        printf("    %d\n", massVector[i]);
    }
    //Printing the number of data points
    printf("\nnumber of data points: \n");
    printf("n = %d\n", numberOfDataPoints);

    //Calculate sum of masses
    int sumOfMasses = 0;
    for (int i = 0; i < numberOfDataPoints; i++)
    {
        sumOfMasses += massVector[i];
    }
    //printf("\n %d \n", sumOfMasses);
    printf("\n");
    //Calculate Center of Mass for each dimension: C = 1/m x (m1 x p1 + m2 x p2
+ ... + mn x pn)
    float centerOfMass[NUMBER_OF_DIMESIONS] = {0.0, 0.0, 0.0};
    for (int iDim = 0; iDim < NUMBER_OF_DIMESIONS; iDim++)
    {
        for (int iPoint = 0; iPoint < numberOfDataPoints; iPoint++)
        {
            centerOfMass[iDim] += (massVector[iPoint] * locationMatrix[iPoint]
[iDim]);
        }

        centerOfMass[iDim] /= sumOfMasses;
    }
    printf("\n");
    //Display Sum of Masses
    printf("center of mass: \n");
    printf("(");
    for (int i = 0; i < NUMBER_OF_DIMESIONS; i++)
    {
        printf("%4.2f", centerOfMass[i]);
        if (i < NUMBER_OF_DIMESIONS - 1)
        {
            printf(", ");
        }

    }

}



/*
Problem 2:

The binary search algorithm that follows may be used to search an array
when the elements are in order. This algorithm is analogous to the following
approach for finding a name in a telephone book.

    a. Open the book in the middle, and look at the middle name on the page.
    b. If the middle name isn't the one you're looking for, decide whether it
        comes before or after the name you want.
    c. Take the appropriate half of the section of the book you were looking in
        and repeat these steps until you land on the name.

ALGORITHM FOR BINARY SEARCH
1. Let bottom be the subscript of the initial array element.
2. Let top be the subscript of the last array element.
3. Let found be false.
4. Repeat as long as bottom isn't greater than top and the target has not been
```

```
found
    5. Let middle be the subscript of the element halfway between bottom and top.
    6. if the element at middle is the target
        7. Set found to true and index to middle.
    else if the element at middle is larger than the target
        8. Let top be middle - 1.
    else
        9. Let bottom be middle + 1.

Write a function print_array that accepts a vector and the length of the vector
that
prints an array on one line as:
9 8 7 6 5 4 3 2 1 0

Write and test a function binary_srch that implements this algorithm for an
array of integers.  Make sure to call binary_srch from your main function.

When there is a large number of array elements, Why is this faster than a
linear search?

*/

// print_array function goes here
void print_array(int myVector[], int vectorLength)
{
    for (int i = 0; i < vectorLength; i++)
    {
        printf(" %d ", myVector[i]);
    }
    printf("\n");
}


// binary_srch function goes here
int binary_srch(int myArray[], int lengthOfArray, int elementToFind)
{
    int indexOfElementToFind = -1;
    int bottom = 0;
    int top = lengthOfArray;
    int indexToCheck = 0;

    while (top >= bottom && indexToCheck >= 0 && indexToCheck <= lengthOfArray)
    {
        indexToCheck = (top + bottom) / 2;
        if (elementToFind == myArray[indexToCheck])
        {
            indexOfElementToFind = indexToCheck;
            break;
        }
        else if (elementToFind > myArray[indexToCheck])
        {
            //Searching top half
            bottom = indexToCheck + 1;
        }
        else if (elementToFind < myArray[indexToCheck])
        {
            //Searching lower half
            top = indexToCheck - 1;
        }
    }

    return indexOfElementToFind;
}


/*

The bubble sort is another technique for sorting an array. A bubble sort compares
adjacent array elements and exchanges their values if they are out of
order. In this way, the smaller values "bubble" to the top of the array (toward
element 0), while the larger values sink to the bottom of the array. After the
first pass of a bubble sort, the last array element is in the correct position;
after
the second pass the last two elements are correct, and so on. Thus, after each
pass, the unsorted portion of the array contains one less element.

Write a function print_array to print an array on one line as:
9 8 7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 1 0 9 Iteration 1
7 6 5 4 3 2 1 0 8 9


Write and test a function called bubble_sort that implements this sorting method
and
call it from your main function.  You will need to swap elements to sort the array.

*/

// bubble_sort function goes here
void bubble_sort(int arrayToSort[], int arrayLength)
{
    int temp = 0;
    for (int i = 0; i < arrayLength - 1; i++)
    {
        for (int j = 0; j < arrayLength - 1; j++)
        {
            if (arrayToSort[j] > arrayToSort[j+1])
            {
                temp = arrayToSort[j];
                arrayToSort[j] = arrayToSort[j+1];
                arrayToSort[j+1] = temp;
            }
        }

    }

}
```