1.Write a C program to implement following operations

a) Transverse

Code:

```
#include <stdio.h>
int main() {
   int array[] = {1, 2, 3, 4, 5};
   int size = sizeof(array) / sizeof(array[0]);

   printf("The elements of the array are: \n");
   for (int i = 0; i < size; i++) {
      printf("%d\n", array[i]);
   }

   return 0;
}
```

Output: The elements of the array are:
1 2 3 4 5

b) Search

Code:

```
#include <stdio.h>

int main() {
   int array[] = {1, 2, 3, 4, 5};
   int size = sizeof(array) / sizeof(array[0]);
   int searchElement = 3;
   int found = 0;

   for (int i = 0; i < size; i++) {
      if (array[i] == searchElement) {
         found = 1;
         printf("Element found at index %d\n", i);
         break;
      }
   }

   if (!found) {
      printf("Element not found in the array\n");
   }

   return 0;
```

```
}
```

Output: Element found at index 2

c) Insert

Code:

```
#include <stdio.h>
int main() {
    int array[10] = {1, 2, 3, 4, 5};
    int size = 5;
    int position = 3;
    int newValue = 10;

    if (position < 0 || position > size)
 {
        printf("Invalid position for insertion\n");
    } else
   {
    for (int i = size; i >= position; i--)
 {
            array[i] = array[i - 1];
        }
        array[position - 1] = newValue;
        size++;

        printf("Array after insertion: \n");
        for (int i = 0; i < size; i++) {
            printf("%d ", array[i]);
        }
        printf("\n");
    }

    return 0;
}
```

Output: Array after insertion:
1 2 10 3 4 5

d) Delete

Code:

```
#include <stdio.h>
```

```
int main() {
    int array[10] = {1, 2, 3, 4, 5};
    int size = 5;
    int position = 3;

    if (position < 0 || position >= size)
 {
        printf("Invalid position for deletion\n");
    }
 else {
        for (int i = position - 1; i < size - 1; i++)
{
            array[i] = array[i + 1];
        }
        size--;

        printf("Array after deletion: \n");
        for (int i = 0; i < size; i++)
{
            printf("%d ", array[i]);
        }
        printf("\n");
    }

    return 0;
}
```

Output: Array after deletion:
1 2 4 5

e) Update

Code:

```
#include <stdio.h>

int main() {
    int array[] = {1, 2, 3, 4, 5};
    int position = 2;
    int newValue = 10;

    if (position < 0 || position >= sizeof(array) / sizeof(array[0])) {
        printf("Invalid position for update\n");
    } else {
        array[position] = newValue;
```

```
        printf("Array after update: \n");
        for (int i = 0; i < sizeof(array) / sizeof(array[0]); i++) {
            printf("%d ", array[i]);
        }
        printf("\n");
    }

    return 0;
}
```
Output: Array after update:
1 2 10 4 5

2.Writing a recursive function to calculate the factorial of a number.

Code:
```
#include<stdio.h>
long int multiplyNumbers(int n);
int main()
{
     int main()
{
     int n;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    printf("Factorial of %d = %ld", n, multiplyNumbers(n));
   return 0;
}
long int multiplyNumbers(int n)
{
    if (n>=1)
 return n*multiplyNumbers(n-1);
    else
   return 1;
}
```
Output: Enter a positive integer: 5
Factorial of 5 = 120

3.Write a C program to find duplicate elements in an array.

Code:
```
#include <stdio.h>
```

```
void findDuplicates(int arr[], int size)
{
   printf("Duplicate elements in the array are: ");
   for (int i = 0; i < size; i++)
{
      for (int j = i + 1; j < size; j++)
{
         if (arr[i] == arr[j])
{
            printf("%d ", arr[i]);
            break;
         }
      }
   }
}

int main() {
   int arr[] = {1, 2, 3, 4, 2, 7, 8, 8, 3};
   int size = sizeof(arr) / sizeof(arr[0]);

   findDuplicates(arr, size);

   return 0;
}
```
Output: Duplicate elements in the array are: 2 3 8

4.Write a C program to find max and min from an array of elements.

Code:
```
#include <stdio.h>

void findMaxAndMin(int arr[], int size) {
   int max = arr[0];
   int min = arr[0];

   for (int i = 1; i < size; i++) {
      if (arr[i] > max) {
         max = arr[i];
      }
      if (arr[i] < min) {
         min = arr[i];
      }
```

```
    }

    printf("Maximum element in the array: %d\n", max);
    printf("Minimum element in the array: %d\n", min);
}

int main() {
    int arr[] = {3, 7, 2, 9, 1, 5, 4};
    int size = sizeof(arr) / sizeof(arr[0]);

    findMaxAndMin(arr, size);

    return 0;
}
```

Output: Minimum element in the array: 1
Maximum element in the array: 9

5.Give a number n the task is to print the fibonacci series and the sum of the series using recursion.

Code:

```
#include <stdio.h>

int fibonacci(int n)
{
    if (n <= 1)
 {
        return n;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main()
 {
    int n = 10;  // Replace 10 with the desired number n
    int sum = 0;

    printf("Fibonacci Series up to %d terms: ", n);
    for (int i = 0; i < n; i++)
 {
        printf("%d ", fibonacci(i));
        sum += fibonacci(i);
    }
```

```
    printf("\nSum of the Fibonacci Series: %d\n", sum);

    return 0;
}
```

Output: Fibonacci Series up to 10 terms: 0 1 1 2 3 5 8 13 21 34
Sum of the Fibonacci Series: 88

6.You are given an array arr in increasing order.Find the element x from array using binary

Code:

```
#include <stdio.h>

int binarySearch(int arr[], int x, int low, int high)
{
    while (low <= high)
 {
        int mid = low + (high - low) / 2;
        if (arr[mid] == x)
{
            return mid;
        } else if (arr[mid] < x)
{
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;
}

int main() {
    int arr[] = {3, 4, 5, 6, 7, 8, 9};
    int x = 4;
    int n = sizeof(arr) / sizeof(arr[0]);
    int result = binarySearch(arr, x, 0, n - 1);

    if (result != -1)
{
        printf("Element is present at index %d\n", result);
    }
 else
{
```

```
        printf("Element not found\n");
    }

    return 0;
}
```
Output: Element is present at index 2

7.C programming code in linear search

Code:

```
#include <stdio.h>

int linearSearch(int arr[], int n, int x) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == x) {
            return i;
        }
    }
    return -1;
}

int main() {
    int arr[] = {3, 6, 8, 5, 2,
#include <stdio.h>

int linearSearch(int arr[], int n, int x) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == x) {
            return i;
        }
    }
    return -1;
}

int main() {
    int arr[] = {5, 2, 9, 1, 7, 3, 8, 4, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 7;

    int result = linearSearch(arr, n, x);
    if (result == -1) {
        printf("Element not found in the array.\n");
    } else {
        printf("Element %d found at index %d.\n", x, result);
```

```
    }

    return 0;
}
```

8.C programming code in binary search

Code:

```
#include <stdio.h>

int binarySearch(int arr[], int l, int r, int x) {
    while (l <= r) {
        int mid = l + (r - l) / 2;

        // If the element is present at the middle
        if (arr[mid] == x) {
            return mid;
        }

        // If the element is greater, ignore left half
        if (arr[mid] < x) {
            l = mid + 1;
        }
        // If the element is smaller, ignore right half
        else {
            r = mid - 1;
        }
    }
#include <stdio.h>

int binarySearch(int arr[], int n, int x) {
    int left = 0;
    int right = n - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == x) {
```

```
            return mid;
        } else if (arr[mid] < x) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return -1;
}

int main() {
    int arr[] = {1, 3, 5, 7, 9, 11, 13, 15};
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 7;

    int result = binarySearch(arr, n, x);
    if (result == -1) {
        printf("Element not found in the array.\n");
    } else {
        printf("Element %d found at index %d.\n",
```