1. Linear Search

Problem Statement:
Write a C program to search an element in an array using the Linear Search method.

Objective:
To understand the basic concept of Linear Search and implement it using arrays and loops in C.

Algorithm
1. Start the program.
2. Input the size of the array and its elements.
3. Input the element to search.
4. Traverse the array sequentially.
5. If the element is found, display its position.
6. Otherwise, display that the element is not found.
7. End the program.

Program Code:

```c
#include <stdio.h>
int main() {
    int arr[100], n, i, search, found = 0;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements: \n", n);
    for(i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter element to search: ");
    scanf("%d", &search);

    for(i = 0; i < n; i++) {
        if(arr[i] == search) {
            printf("%d found at position %d\n", search, i+1);
            found = 1;
            break;
        }
    }
    if(!found)
        printf("%d not found in the array.\n", search);
    return 0;
}
```

Sample Output:

```
Enter number of elements: 5
Enter 5 elements:
2 4 6 8 10
Enter element to search: 8
8 found at position 4
```

2. Binary Search

Problem Statement:
Write a C program to search an element in a sorted array using Binary Search.

Objective:
To learn and apply Binary Search algorithm using arrays and loops.

Algorithm:
```
1. Start the program.
2. Input the number of elements and enter them in sorted order.
3. Input the element to search.
4. Initialize `low = 0`, `high = n-1`.
5. Compute `mid = (low + high)/2`.
6. If `arr[mid] == key`, print position.
7. If `arr[mid] > key`, set `high = mid - 1`.
8. If `arr[mid] < key`, set `low = mid + 1`.
9. Repeat until element is found or `low > high`.
10. End the program.
```

Program Code:

```c
#include <stdio.h>
int main() {
    int arr[100], n, i, search, low, high, mid;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d sorted elements: \n", n);
    for(i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter element to search: ");
    scanf("%d", &search);

    low = 0;
    high = n - 1;

    while(low <= high) {
        mid = (low + high) / 2;
        if(arr[mid] == search) {
            printf("%d found at position %d\n", search, mid + 1);
            return 0;
        } else if(arr[mid] < search)
            low = mid + 1;
        else
            high = mid - 1;
    }
    printf("%d not found in the array.\n", search);
    return 0;
}
```

Sample Output:

```
Enter number of elements: 6
Enter 6 sorted elements:
2 4 6 8 10 12
```

```
Enter element to search: 6
6 found at position 3
```

3. Array Insertion

Problem Statement:
Write a C program to insert an element in an array at a specific position.

Objective:
To perform insertion operation on arrays.

Algorithm:
1. Start the program.
2. Input the array size and elements.
3. Input the element and position to insert.
4. Shift elements right from the specified position.
5. Insert the new element.
6. Display the updated array.

Program Code:

```c
#include <stdio.h>
int main() {
    int arr[100], n, pos, val, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements: \n", n);
    for(i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter element to insert: ");
    scanf("%d", &val);
    printf("Enter position: ");
    scanf("%d", &pos);

    for(i = n; i >= pos; i--)
        arr[i] = arr[i - 1];

    arr[pos - 1] = val;
    n++;

    printf("Array after insertion: ");
    for(i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

Sample Output:

```
Enter number of elements: 4
Enter 4 elements:
10 20 30 40
Enter element to insert: 25
Enter position: 3
Array after insertion: 10 20 25 30 40
```

4. Array Deletion

Problem Statement:
Write a C program to delete an element from a given position in an array.

Objective:
To learn how to remove elements from an array by shifting elements.

Algorithm:
```
1. Start the program.
2. Input size and elements of array.
3. Input position to delete.
4. Shift elements left from the position.
5. Reduce array size.
6. Display updated array.
```

Program Code:

```c
#include <stdio.h>
int main() {
    int arr[100], n, pos, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements: \n", n);
    for(i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter position to delete: ");
    scanf("%d", &pos);

    for(i = pos - 1; i < n - 1; i++)
        arr[i] = arr[i + 1];

    n--;

    printf("Array after deletion: ");
    for(i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

Sample Output:

```
Enter number of elements: 5
Enter 5 elements:
10 20 30 40 50
Enter position to delete: 3
Array after deletion: 10 20 40 50
```

5. Bubble Sort

Problem Statement:

Write a C program to sort an array using the Bubble Sort technique.

Objective:
To understand sorting algorithms and implement Bubble Sort.

Algorithm:
1. Start the program.
2. Input size and elements of the array.
3. Compare adjacent elements.
4. Swap them if they are in the wrong order.
5. Repeat until the array is sorted.
6. Display the sorted array.

Program Code:

```c
#include <stdio.h>
int main() {
    int arr[100], n, i, j, temp;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements: \n", n);
    for(i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    for(i = 0; i < n-1; i++) {
        for(j = 0; j < n-i-1; j++) {
            if(arr[j] > arr[j+1]) {
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }

    printf("Sorted array: ");
    for(i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

Sample Output:

```
Enter number of elements: 5
Enter 5 elements:
40 10 30 20 50
Sorted array: 10 20 30 40 50
```