# Searching Algorithms in C

Define all Searching types using C language programs.

Certainly! Here are examples of different searching algorithms implemented in C. I'll cover Linear Search, Binary Search, and Hashing.

### 1. Linear Search

Linear Search scans each element in the array sequentially until the desired element is found or the end of the array is reached.

```c
#include <stdio.h>

// Function for Linear Search
int linearSearch(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            return i; // Element found, return index
        }
    }
    return -1; // Element not found
}

int main() {
    int arr[] = {2, 4, 6, 8, 10};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 8;

    int result = linearSearch(arr, size, target);
    if (result != -1) {
        printf("Element found at index %d\n", result);
    } else {
        printf("Element not found\n");
    }

    return 0;
}
```

### 2. Binary Search

Binary Search requires the array to be sorted. It repeatedly divides the search interval in half until the target element is found or the interval is empty.

```c
#include <stdio.h>

// Function for Binary Search
int binarySearch(int arr[], int left, int right, int target) {
    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == target) {
            return mid; // Element found, return index
        }
        if (arr[mid] < target) {
            left = mid + 1; // Search in the right half
        } else {
            right = mid - 1; // Search in the left half
        }
    }
    return -1; // Element not found
}

int main() {
    int arr[] = {1, 3, 5, 7, 9, 11};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 7;

    int result = binarySearch(arr, 0, size - 1, target);
    if (result != -1) {
        printf("Element found at index %d\n", result);
    } else {
        printf("Element not found\n");
    }

    return 0;
}
```

## 3. Hashing

Hashing involves using a hash function to map data to a fixed-size array (hash table). This example demonstrates basic hashing using an array and linear probing for collision resolution.

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

#define TABLE_SIZE 10

// Hash function
int hash(int key) {
    return key % TABLE_SIZE;
}

// Function to insert into hash table
void insert(int hashTable[], int key) {
    int index = hash(key);

    // Linear probing in case of collision
    while (hashTable[index] != -1) {
        index = (index + 1) % TABLE_SIZE;
    }
    hashTable[index] = key;
}

// Function to search in hash table
int search(int hashTable[], int key) {
    int index = hash(key);
    int startIndex = index;

    // Linear probing
    while (hashTable[index] != -1) {
        if (hashTable[index] == key) {
            return index; // Element found
        }
        index = (index + 1) % TABLE_SIZE;
        if (index == startIndex) {
            break; // Loop back to the start
        }
    }
    return -1; // Element not found
}

int main() {
    int hashTable[TABLE_SIZE];
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable[i] = -1; // Initialize hash table
    }

    insert(hashTable, 15);
    insert(hashTable, 25);
    insert(hashTable, 35);

    int key = 25;
    int result = search(hashTable, key);

    if (result != -1) {
        printf("Element found at index %d\n", result);
    } else {
        printf("Element not found\n");
    }

    return 0;
}
```

## Summary

- **Linear Search**: Simple and works on unsorted arrays but inefficient for large data sets.
- **Binary Search**: Efficient for sorted arrays, with a time complexity of $O(\log n)$.
- **Hashing**: Provides efficient average-time complexity for insertion and search operations but requires extra space and can suffer from collisions.

Feel free to adapt these examples to suit your needs or expand on them with more advanced features!

ChatGPT can make mistakes. Check important info.