Searching Techniques

Searching algorithms are essential components in programming used to locate specific items within a collection of data. Depending on the input array, there are two primary searching techniques:

1. **Linear Search**
2. **Binary Search**

Linear Search

- **Usage**: Applied to unsorted arrays.
- **Process**: Sequentially checks each element of the array from the beginning until the key is found. Returns the index of the key if present.
- **Time Complexity**: (O(n)).

Pseudocode

```
int linearSearch(int arr[], int key) {
    for(int i = 0; i < arr.length; i++) {
        if(arr[i] == key) {
            return i; // Return the index of the key
        }
    }
    return -1; // Return -1 if the key is not found
}
```

Binary Search

- **Usage**: Applied to sorted arrays.
- **Process**: Compares the middle element of the array with the key. If they match, the index is returned. Otherwise, the search continues in the left or right half of the array by updating the `low` or `high` pointers accordingly.
- **Time Complexity**: (O(\log n)).

Key Steps

1. Compare the key with the middle element.
2. If equal, return the index.
3. If the key is smaller, search the left half.
4. If the key is larger, search the right half.

Pseudocode

```
int binarySearch(int arr[], int key) {
    int low = 0;
    int high = arr.length - 1;
    while(low <= high) {
        int mid = low + (high - low) / 2;
        if(arr[mid] == key) {
            return mid;
```

```
        } else if(arr[mid] < key) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1; // Return -1 if the key is not found
}
```

Summary

- **Linear Search**: Simple but inefficient for large datasets ((O(n))).
- **Binary Search**: Efficient ((O(\log n))) but requires the array to be sorted beforehand.