Linear Search

Linear search is defined as a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set.

Linear Search Algorithm:

Linear Search (Array Arr, array size, Value a) // Arr is the name of the array, and a is the searched element.

```
Step 1: Set i to 0                          // i is the index of an array which starts from 0
Step 2: if i > n then go to step 7          // n is the number of elements in array
Step 3: if Arr[i] = a then go to step 6
Step 4: Set i to i + 1
Step 5: Goto step 2
Step 6: Print element a found at index i and go to step 8
Step 7: Print element not found
Step 8: Exit
```

Time Complexity of Linear Search:

Best Case: In the best case, the key might be present at the first index. So the best case complexity is O(1)

Worst Case: In the worst case, the key might be present at the last index i.e., opposite to the end from which the search has started in the list. So the worst-case complexity is O(N) where N is the size of the list.

Average Case: O(N)

Advantages of Linear Search:

- Linear search can be used irrespective of whether the array is sorted or not.
- Does not require any additional memory.
- It is a well-suited algorithm for small datasets.

Disadvantage of Linear Search:

- Linear search has a time complexity of O(N), which in turn makes it slow for large datasets.
- Not suitable for large arrays.

Linear Search Program:

```
int linear_search(int arr[], int n, int target) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == target) {
            return i; // Return the index if target is found
        }
    }
    return -1; // Return -1 if target is not found
}

int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int n = sizeof(arr) / sizeof(arr[0]); // Calculate the size of the array
    int target = 30;
    int result = linear_search(arr, n, target);
    if (result != -1) {
        printf("Element found at index: %d\n", result);
    } else {
        printf("Element not found\n");
    }
    return 0;
}
```

20 4

4 8

11 1

hi 11

0

Binary Search

Binary Search is defined as a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to **O(log N).**

Binary Search Algorithm:

```
Binary search (A, n, key)

Step 1: set low=1

Step 2: set high=n

Step 3: while (low<=high) repeat step 4 to 9

Step 4: mid= (low+high)/2

Step 5: If (key==A[mid]) then

                    Step 6: return mid

Step 7: else if (key<A[mid])

                              Step 8: high=mid-1

Step 9: else low=mid+1

[End while step 3]

Step 10: Exit
```

Time Complexity Analysis of Binary Search:

Best Case: O(1)

Average Case: O(log N)

Worst Case: O(log N)

Advantages of Binary Search:

- Binary search is faster than linear search, especially for large arrays.
- Binary search is well-suited for searching large datasets that are stored in external memory, such as on a hard drive or in the cloud.

Drawbacks of Binary Search:

- The array should be sorted.
- Binary search requires that the data structure being searched be stored in contiguous memory locations.

Applications of Binary Search:

- Binary search can be used as a building block for more complex algorithms used in machine learning, such as algorithms for training neural networks or finding the optimal hyperparameters for a model.
- It can be used for searching in computer graphics such as algorithms for ray tracing or texture mapping.
- It can be used for searching a database.