Linear Search

a **linear search** or **sequential search** is a method for finding an element in a sequential way

Program

#include <bits/stdc++.h>

using namespace std;

int search(int arr[], int n, int x)

{

    int i;

    for (i = 0; i < n; i++)

        if (arr[i] == x)

            return i;

    return -1;

}

int main()

{

    int arr[] = { 3, 4, 1, 7, 5 };

    int n = sizeof(arr) / sizeof(arr[0]);

    int x = 4;

    int index = search(arr, n, x);

    if (index == -1)

        cout << "Element is not present in the array";

    else

        cout << "Element found at position " << index;

    return 0;

}

In **linear search**, best-case **complexity** is O(1) where the element is found at the first index. Worst-case **complexity** is O(n) where the element is found at the last index or element is not present in the array. In **binary search**, best-case **complexity** is O(1) where the element is found at the middle index.

Binary Search

It works by comparing the middle item of the array with our target, if it matches, it returns true otherwise if the middle term is greater than the target, the search is performed in the left sub-array. If the middle term is less than the target, the search is performed in the right sub-array.

#include <bits/stdc++.h>

using namespace std;

// A iterative binary search function. It returns

// location of x in given array arr[l..r] if present,

// otherwise -1

int binarySearch(int arr[], int l, int r, int x)

{

    while (l <= r) {

        int m = l + (r - l) / 2;

        if (arr[m] == x)

            return m;

        if (arr[m] < x)

            l = m + 1;

        else

            r = m - 1;

    }

    return -1;

}

int main(void)

{

    int arr[] = { 2, 3, 4, 10, 40 };

    int x = 10;

    int n = sizeof(arr) / sizeof(arr[0]);

    int result = binarySearch(arr, 0, n - 1, x);

    (result == -1) ? cout << "Element not found"

                   : cout << "Element is present at index " << result;

    return 0;

}

The **time complexity** of the **binary search** algorithm is O(log n). The best-case **time complexity** would be O(1) when the central index would directly match the desired value. The worst-case scenario could be the values at either extremity of the list or values not in the list.

Interpolation Search

**Interpolation search** is an algorithm for **searching** for a key in an array that has been ordered by numerical values assigned to the keys (key values). It was first described by W. W. Peterson in 1957.

#include<bits/stdc++.h>

using namespace std;

int interpolationSearch(int arr[], int n, int x)

{

    // Find indexes of two corners

    int lo = 0, hi = (n - 1);

    // Since array is sorted, an element present

    // in array must be in range defined by corner

    while (lo <= hi && x >= arr[lo] && x <= arr[hi])

    {

        if (lo == hi)

        {

            if (arr[lo] == x) return lo;

            return -1;

        }

        int pos = lo + (((double)(hi - lo) /

            (arr[hi] - arr[lo])) \* (x - arr[lo]));

        if (arr[pos] == x)

            return pos;

        if (arr[pos] < x)

            lo = pos + 1;

        else

            hi = pos - 1;

    }

    return -1;

}

int main()

{

    int arr[] = {10, 12, 13, 16, 18, 19, 20, 21,

                 22, 23, 24, 33, 35, 42, 47};

    int n = sizeof(arr)/sizeof(arr[0]);

    int x = 18; // Element to be searched

    int index = interpolationSearch(arr, n, x);

    if (index != -1)

        cout << "Element found at index " << index;

    else

        cout << "Element not found.";

    return 0;

}

**Time Complexity**: O(log2(log2 n)) for the average case, and O(n) for the worst case (when items are distributed exponentially) , best case time complexity is O(1).

Other searching algorithms are jump search , ternary search.