**Searching Methods in Array:**

* Linear Search
* Binary Search

Array A with size=10

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 18 | 19 | 14 | 17 | 7 | 6 | 3 | 10 | 5 | 2 |
| i=0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Succesful search= key element is found🡪 key=10

UnSuccesful search= key element not found🡪 key=22.

Linear search—comparing elements while searching the key element—atmost how many comparisons

In minimum, how many comparisons linear search performs.

Best case: Search element at the first location, require one comparison. So time taken by for loop in best case in O(1).

Worst Case: linearsearch performs maximum comparsion, search element at the last location, time taken by for loop is O (n)

In case of unsuccesful search, where element is not found in the array, the time taken is exactly O(n).

Average Case: we sum up the number of time taken in all possible cases divided by total cases

For example, seraching 18 requires 1 comparison, searching 19 requires 2 comparisons and so on

T(n)= 1+2+3+……………………n/ n =

= (n(n+1)/2) /n = n+1/2 = O(n).

Mostly we will perform worst case analysis of algorithm. Usually, average case and worst case analysis of most of the algorithm is same. Average case analysis for some algorithm is hard to find.

Linear\_Search (A, n, key)

{

For (i=0; I < n; i++)

{

If ( key == A[i])

return i;

}

return -1; // invalid index indicates unsuccesful search

}

**Linear Search Java Program**

import java.util.Scanner;

public class LinearSearch {

// This function returns index of element x in arr[]

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

{

for (int i = 0; i < n; i++)

{

// Return the index of the element if the element

// is found

if (arr[i] == x)

return (i+1);

}

// return -1 if the element is not found

return -1;

}

public static void main(String[] args) {

int counter, num, item, array[];

//To capture user input

Scanner input = new Scanner(System.in);

System.out.println("Enter number of elements:");

num = input.nextInt();

//Creating array to store the all the numbers

array = new int[num];

System.out.println("Enter " + num + " integers");

//Loop to store each numbers in array

for (counter = 0; counter < num; counter++)

array[counter] = input.nextInt();

System.out.println("Enter the search value:");

item = input.nextInt();

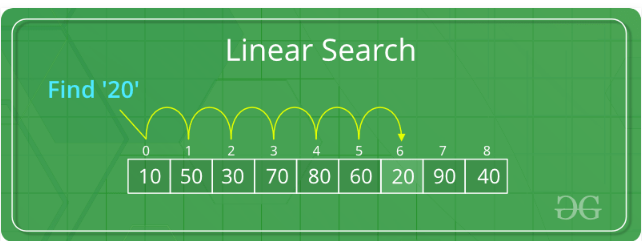
System.out.println("The elemnt lie at location:" +search(array,array.length,item));

}}

**Linear Search Case Analysis**

A simple approach is to do **linear search**, i.e

* Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
* If x matches with an element, return the index.
* If x doesn’t match with any of elements, return -1.



**Worst Case Analysis (Usually Done)**

We must know the case that causes maximum number of operations to be executed. For Linear Search, the worst case happens when the element to be searched (x in the above code) is not present in the array. When x is not present, the search() functions compares it with all the elements of arr[] one by one. Therefore, the worst case time complexity of linear search would be Θ(n).

**Best Case Analysis**

We must know the case that causes minimum number of operations to be executed. In the linear search problem, the best case occurs when x is present at the first location. The number of operations in the best case is constant (not dependent on n).

So time complexity in the best case would be Θ(1)  
Most of the times, we do worst case analysis to analyze algorithms. In the worst analysis, we guarantee an upper bound on the running time of an algorithm which is good information.