***Introduction:***

What is Searching?

* Searching is the process of finding a given value position in a list of values.
* It decides whether a search key is present in the data or not.
* It is the algorithmic process of finding a particular item in a collection of items.
* It can be done on internal data structure or on external data structure.

**To search an element in a given array, it can be done in following ways:**  
1. Sequential Search  
2. Binary Search

***Theory:***

Sequential Search (Linear Search):

* Sequential search is also called as Linear Search.
* Sequential search starts at the beginning of the list and checks every element of the list.
* It is a basic and simple search algorithm.
* Sequential search compares the element with all the other elements given in the list. If the element is matched, it returns the value index, else it returns -1.

Binary Search:

* Binary Search is used for searching an element in a sorted array.
* It is a fast search algorithm with run-time complexity of O(log n).
* Binary search works on the principle of divide and conquer.
* This searching technique looks for a particular element by comparing the middle most element of the collection.
* It is useful when there are large number of elements in an array.

***Objective:***

At the end of this experiment you will be able to perform :

* Perform Searching using Linear search Algorithm.
* Perform Searching using Binary search Algorithm.

***Experiment:***

Sequential Search (Linear Search):

#include <stdio.h>

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(void)

{

int arr[] = { 2, 3, 4, 10, 40 };

int x = 10;

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

int result = search(arr, n, x);

(result == -1) ? printf("Element is not present in array"): printf("Element is present at index %d",result);

return 0;

}

Binary Search:

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

{

if (r >= l)

{

int mid = l + (r - l) / 2;

if (arr[mid] == x)

return mid;

if (arr[mid] > x)

return binarySearch(arr, l, mid - 1, x);

return binarySearch(arr, mid + 1, r, x);

}

return -1;

}

int main(void)

{

int arr[] = { 2, 3, 4, 10, 40 };

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

int x = 10;

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

(result == -1) ? printf("Element is not present in array")

: printf("Element is present at index %d",

result);

return 0;

}

***Algorithm:***

Sequential Search (Linear Search):

Linear search is a very basic and simple search algorithm. In Linear search, we search an element or value in a given array by traversing the array from the starting, till the desired element or value is found.

It compares the element to be searched with all the elements present in the array and when the element is **matched** successfully, it returns the index of the element in the array, else it returns -1.

Linear Search is applied on unsorted or unordered lists, when there are fewer elements in a list.

Binary Search:

Binary Search is used with sorted array or list. In binary search, we follow the following steps:

1. We start by comparing the element to be searched with the element in the middle of the list/array.
2. If we get a match, we return the index of the middle element.
3. If we do not get a match, we check whether the element to be searched is less or greater than in value than the middle element.
4. If the element/number to be searched is greater in value than the middle number, then we pick the elements on the right side of the middle element(as the list/array is sorted, hence on the right, we will have all the numbers greater than the middle number), and start again from the step 1.
5. If the element/number to be searched is lesser in value than the middle number, then we pick the elements on the left side of the middle element, and start again from the step 1.

Binary Search is useful when there are large number of elements in an array and they are sorted.

So a necessary condition for Binary search to work is that the list/array should be sorted.