Lab 8

1. **Explain Linear search and binary search with examples.**

**Linear Search**

Linear search, also called as sequential search, is a very simple method used for searching an array for a particular value. It works by comparing the value to be searched with every element of the array one by one in a sequence until a match is found.

Linear search is mostly used to search an unordered list of elements (array in which data elements are not sorted). For example, if an array A[10] is declared and initialized as,

int A[10] = {10, 8, 2, 7, 3, 4, 9, 1, 6, 5};

Val = 7 then Pos = 3

**Linear Search Algorithm**

LINEAR\_SEARCH(A, N, VAL)

Step 1: [INITIALIZE] SET POS = -1

Step 2: [INITIALIZE] SET I = 1

Step 3: Repeat Step 4 while I<=N

Step 4: IF A[I] = VAL

SET POS = I

PRINT POS

Go to Step 6

[END OF IF]

SET I = I + 1

[END OF LOOP]

Step 5: IF POS = –1

PRINT VALUE IS NOT PRESENT IN THE ARRAY

[END OF IF]

Step 6: EXIT

**Complexity of Linear Search Algorithm**

Linear search executes in O(n) time where n is the number of elements in the array.

**Binary Search**

Binary search is a searching algorithm that works efficiently with a sorted list.

The mechanism of binary search can be better understood by an analogy of a telephone directory. When we are searching for a particular name in a directory, we first open the directory from the middle and then decide whether to look for the name in the first part of the directory or in the second part of the directory. Again, we open some page in the middle and the whole process is repeated until we finally find the right name.

**Binary search example:**

Consider an array A[11] that is declared and initialized as,

int A[11] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

and

value to be searched is VAL = 9

**Binary search Algorithm**

BINARY\_SEARCH(A, lower\_bound, upper\_bound, VAL)

Step 1: [INITIALIZE] SET BEG = lower\_bound

END = upper\_bound, POS = - 1

Step 2: Repeat Steps 3 and 4 while BEG <= END

Step 3: SET MID = (BEG + END)/2

Step 4: IF A[MID] = VAL

SET POS = MID

PRINT POS

Go to Step 6

ELSE IF A[MID] > VAL

SET END = MID - 1

ELSE

SET BEG = MID + 1

[END OF IF]

[END OF LOOP]

Step 5: IF POS = -1

PRINT “VALUE IS NOT PRESENT IN THE ARRAY”

[END OF IF]

Step 6: EXIT

**2. Write a complete C program to implement the following searching algorithms:**

**Using an array of size 10, populated with random integers, e.g. 0 – 100 to test your program. Don’t forget to print initial array, and output (for binary search, print also each sub-array during the searching process).**

**a. Linear Search**

**a. The function should receive an integer array, a search key and the size of the array as arguments.**

**b. If the search key is found, return the location in the array where the search key is found; otherwise, return -1.**

**Code:**

#include <stdio.h>

int main()

{

int a[10], i, item,n;

printf("\nEnter number of elements of an array:\n");

scanf("%d",&n);

printf("\nEnter elements: \n");

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

scanf("%d", &a[i]);

printf("\nEnter item to search: ");

scanf("%d", &item);

for (i=0; i<=9; i++)

if (item == a[i])

{

printf("\nItem found at location %d", i+1);

break;

}

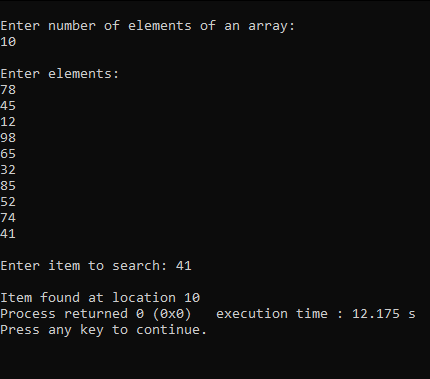
if (i > 9)

return -1;

return 0;

}

**Output:**



**b. Binary Search**

**a. The function should receive an integer array, a search key, the starting subscript and ending subscript as arguments.**

**b. If the search key is found, return the location in the array where the search key is found; otherwise, return -1.**

**Code:**

#include <stdio.h>

int main()

{

int c, first, last, middle, n, search, array[10];

printf("Enter number of elements of an array:\n");

scanf("%d", &n);

printf("Enter %d elements:\n", n);

for (c = 0; c < n; c++)

scanf("%d", &array[c]);

printf("Enter item to search\n");

scanf("%d", &search);

first = 0;

last = n - 1;

middle = (first+last)/2;

while (first <= last) {

if (array[middle] < search)

first = middle + 1;

else if (array[middle] == search) {

printf("Item found at location %d.\n", search, middle+1);

break;

}

else

last = middle - 1;

middle = (first + last)/2;

}

if (first > last) {

return -1;

}

return 0;

}

**Output:**

