**SEARCHING**

**EXPT NO: 9**  **DATE: 30/12/21**

**AIM**

**1)** Write a program to implement Linear Search.

**2)** Write a program to implement Binary Search.

**THEORY**

**Linear Search (Sequential Search)**

Sequential search is performed in a linear way i.e., it starts from the beginning of the list and continues till we find the item or reach the end of list. The item to be search is compared with each element of the list one by one, starting from the first element.

The number of comparisons required to search for an item depends on the position of element inside the array. The best case is when the item is present at the first position and in this case only one comparison is done. The worse case occurs when the item is not present in the array and in this case n comparisons are required where n is the total number of elements. Searching for an item at ith position requires i comparisons. The average number of comparisons required in a successful search assuming that the probability of searching of all elements is same.

(1+2+3+4+....+n)/n = (n+1)/2

So, both is average and worst case the run time complexity of **Linear** **search** is **O(n)**

**Binary Search**

The prerequisite for binary search is that the array should be sorted. Firstly, we compare the item to be searched with the middle element of the array. If the item if found there, our search finishes successfully otherwise the array is divided into two halves, first half contains all elements to the left of the middle element and the other one consists of all elements to the right side of the middle element. Since the array is sorted, all the elements in the left half will be smaller that the middle element and the element in the right will be greater that the middle element. If the item is less than the middle element, it searched in the left half otherwise it is searched in the right half.

Now the search proceeds in the smaller potion of the array and the item is compared with its middle element. If the item is same as the middle element, search finishes otherwise again the subarray is divided into two halves and the search is performed in one of these halves. This process of comparing the item with the middle element and dividing the array continues till we find the required item or get a portion which does not have any element.

To implement this procedure, we will take 3 variables viz. low, up and mid that will keep track of the status of lower limit, upper limit and middle value of that portion of the array, in which we will search the element. If the array contains even number of elements, there will be two middle elements, we’ll take first one as their middle element middle element. The value of mid can be calculated using mid = (low + high)/2

If(item>arr[mid)

Search will resume in the right half which is arr[mid+1]….arr[up].

So low=mid+1, up remains same.

If(item<arr[mid)

Search will resume in the left half which is arr[low]….arr[mid-1].

So up=mid-1, down remains same.

If(item==arr[mid)

Search is successful

Item found at mid position

If(low>up)

Search is unsuccessful

Item is not present in the array.

The best case of binary search is when the item is present in in the middle of the array, and in this case the loop is executed only once. The worst case occurs when the item is not present in the array. In each iteration, the array is divided into half, so if the size of the array is n, there will be maximum log(n) such divisions. Thus, there will be log n comparisons in worst case. The run time complexity of **Binary search** is **O(log(n)).**

Binary search is preferred only when the data is static i.e. very few insertions and deletions are done. This because whenever am insertion or deletion is to be done, may elements have to be moved to keep the data in sorted order. Binary search is not suitable for linked list because it requires direct access to the middle element.

**CODES**

**1)**

#include<stdio.h>

#include<stdlib.h>

void LINEARSEARCH(int arr[], int n, int val)

{

int i;

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

{

if(arr[i]==val)

{

printf("ELEMENT FOUND AT LOACTION %d\n",i+1);

return;

}

}

printf("ELEMENT NOT FOUND\n");

}

int main()

{

int \*arr, n, val, i;

printf("ENTER THE SIZE OF THE ARRAY: ");

scanf("%d",&n);

arr=(int \*)malloc(n\*sizeof(int));

printf("ENTER THE %d VALUES\n",n);

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

{

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

}

while(1)

{

printf("\nENTER THE SEARCH KEY: ");

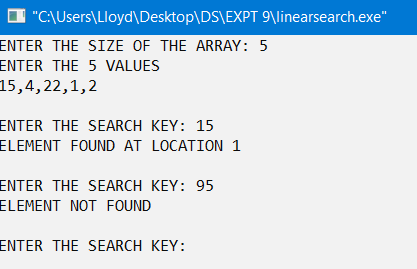
scanf("%d",&val);

LINEARSEARCH(arr,n,val);

}

return 0;

}

**OUTPUT**

**2)**

#include<stdio.h>

#include<stdlib.h>

void BINARYSEARCH(int arr[], int l, int h, int val)

{

if(l<=h)

{

int mid=(l+h)/2;

if(val==arr[mid])

{

printf("ELEMENT FOUND AT POSITION %d\n",mid+1);

return;

}

else if(val < arr[mid])

BINARYSEARCH(arr,l,mid-1,val);

else

BINARYSEARCH(arr,mid+1,h,val);

}

else

printf("ELEMENT NOT FOUND\n");

}

int main()

{

int \*arr, n, val,i;

printf("ENTER THE SIZE OF THE ARRAY: ");

scanf("%d",&n);

arr=(int \*)malloc(n\*sizeof(int));

printf("ENTER THE %d VALUES\n",n);

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

{

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

}

while(1)

{

printf("ENTER THE SEARCH KEY: ");

scanf("%d",&val);

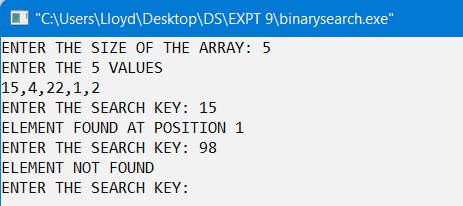
BINARYSEARCH(arr,0,n-1,val);

}

return 0;

}

**OUTPUT**

****

**CONCLUSION**

The given problem statement was successfully compiled and executed.

**LEARNINGS AND FINDINGS**

The experiment demonstrates

1. Concept of Linear & Binary Search
2. Implementation of Linear & Binary Search
3. Time complexity of both the algorithms

Runtime complexity of Binary search is O(log(N)) which makes it the more efficient than the Linear search which has a time complexity of O(N).

|  |  |
| --- | --- |
| **SR. NO.** | **COMPILATION TIME** |
| 1 | 0.19 s |
| 2 | 0.19 s |