|  |
| --- |
| **NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES**  **CS 201–DATA STRUCTURES LAB**  **Lab Session 12** |
| **Instructors:** Mr. Irfan Ayub, Ms. Safia, |

## Objective:

"Searching algorithms in Data structure"

There are basically two types of searches, Sequential search and Binary search. These searching can efficiently work with different types of data structures such as arrays and linked list. We can find sequential and binary search performs better for arrays while Hash Table works better in linked list. Sequential Or Linear Search: Sequential search is used whenever the data list is not ordered. Generally, it is used only for small data list. The Sequential search algorithm calls two things; first did it find the data it was looking for? Second, if it did, at what index (location) are the target data found? Considering algorithm requirement, we need four parameters

1: The list we are searching (array),

2: An index to the last element in the list, 3: The target data,

4: And the address where they found element's location is to be stored.



Consider the following example:

|  |
| --- |
| int seq\_search( int array[], int last, int target, int index)  {  int i=0;  while (i <last && target != array[i]){ i++; }  index=i;  if(target==array[i]) return index;  else  return index;  return index;  } |

The linear search algorithm efficiency is O(n).

Case Best Worst Average found 1 n n/2

Not found n n n

### Tasks:

1: Write a program that creates an array of 100 random integers in the 1 to 200 and apply sequential search and search array 100 times using randomly generated targets by same range and display the number of searches

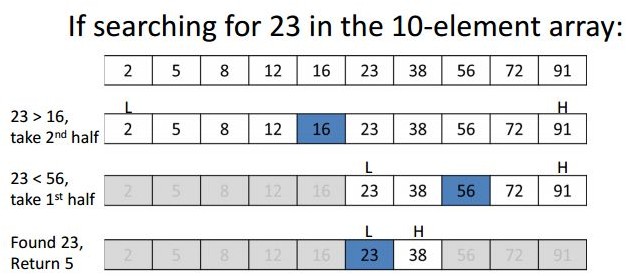
Completed and number of successful searches.

2: Implement the given program, complete it and check the result from external file.

## Binary Search:

The sequential search is very slow. If we have unsorted array then we have only sequential sort as solution. However if the array is sorted, we can use a more efficient algorithm called the Binary Search. The binary search starts by testing the data in the element at the middle of the array to determine if the target is in the first or second half of the list. If it is in first half then we don't need to test the second half, and if it is in second half, we don't need to test it in first half. In simple words we eliminate half the array from further test. First, we need three variables to get the middle of array, first index, middle index and Last index.

We test only two cases, 1: The target is in the array, 2: The target is not in the array.



The binary search finds an item by dividing array repeatedly. As the loop is dividing therefore the efficiency is O(log2n). Thus disregarding the time required to sort the array, the binary search is more efficient for any significant size of array.

Consider the following example: mid= (first+last)/2

int Bin\_search( int array[], int end, int target)

{

int first=0,last=end,mid,index; while (first<=last){

mid =(first+last)/2; if(target<array[mid])

//look in first half last = mid-1;

else if(target>array[mid])

//look in second half first= mid+1;

else break;

}

index=mid; return index;

}

### Task:

1: Implement Binary search recursively.

2: Apply Binary search, generate a sequence of an array of numbers starting with 1 and then add 2 to create the next numbers in the series, as shown below

1,3,4,6,7,9,10.....147,148,150

For the search arguments, generate numbers in the range of 1 to 150.