INTRODUCTION TO SEARCHING

Searching means to find whether a particular value is present in an array or not. If the value is present in the array, then searching is said to be successful and the searching process gives the location of that value in the array. However, if the value is not present in the array, the searching process displays an appropriate message and in this case searching is said to be unsuccessful.There are two popular methods for searching the array elements: linear search and binary search.The algorithm that should be used depends entirely on how the values are organized in the array.For example, if the elements of the array are arranged in ascending order, then binary search should be used, as it is more efficient for sorted lists in terms of complexity.

Linear Search

Linear search, also called 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[] is declared and initialized as,

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

and the value to be searched is VAL = 7 , then searching means to find whether the value ‘7’ is present in the array or not. If yes, then it returns the position of its occurrence. Here, POS = 3 (index starting from 0).

Algorithm: linear search

```
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]
```

Complexity of Linear Search Algorithm

Linear search executes in O(n) time where n is the number of elements in the array. Obviously, the best case of linear search is when VAL is equal to the first element of the array. In this case, only one comparison will be made. Likewise, the worst case will happen when either VAL is not present in the array or it is equal to the last element of the array. In both the cases, n comparisons will have to be made. However, the performance of the linear search algorithm can be improved by using a sorted 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.

Take another analogy. How do we find words in a dictionary? We first open the dictionary somewhere in the middle. Then, we compare the first word on that

page with the desired word whose meaning we are looking for. If the desired word comes before the word on the page, we look in the first half of the dictionary, else we look in the second half. Again, we open a page in the first half of the dictionary and compare the first word on that page with the desired word and repeat the same procedure until we finally get the word. The same mechanism is applied in the binary search.

Now, let us consider how this mechanism is applied to search for a value in a sorted array.
Consider an array A[] that is declared and initialized as
int A[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
and the value to be searched is VAL = 9.
The algorithm will proceed in the following manner.
BEG = 0, END = 10, MID = (0 + 10)/2 = 5
Now, VAL = 9 and A[MID] = A[5] = 5
A[5] is less than VAL , therefore, we now search for the value in the second half of the array. So,we change the values of BEG and MID .
Now, BEG = MID + 1 = 6, END = 10, MID = (6 + 10)/2 =16/2 = 8
VAL = 9 and A[MID] = A[8] = 8
A[8] is less than VAL , therefore, we now search for the value in the second half of the segment.So, again we change the values of BEG and MID .
Now, BEG = MID + 1 = 9, END = 10, MID = (9 + 10)/2 = 9
Now, VAL = 9 and A[MID] = 9 .

In this algorithm, we see that BEG and END are the beginning and ending positions of the segment that we are looking to search for the element. MID is calculated as (BEG + END)/2 . Initially, BEG =lower_bound and END = upper_bound . The algorithm will terminate when A[MID] = VAL . When the algorithm ends, we will set POS = MID . POS is the position at which the value is present in the array.

However, if VAL is not equal to A[MID] , then the values of BEG , END , and MID will be changed depending on whether VAL is smaller or greater than A[MID] .
(a) If VAL < A[MID] , then VAL will be present in the left segment of the array. So, the value of END will be changed as END = MID – 1 .

(b) If VAL > A[MID] , then VAL will be present in the right segment of the array. So, the value of BEG will be changed as BEG = MID + 1 .

Algorithm: binary search

```
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
                           GOTO 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
```