Searching and Sorting I

- **Searching and sorting are two of the most common operations in computer science. Searching refers to finding the position of a value in a collection of values.**
- **Sorting refers to arranging data in a certain order. The two commonly used orders are numerical order and alphabetical order.**

- 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 as sequential search, is a very simple method used for searching an array for a particular value.**

- **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,**

Algorithm for 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]
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**
- **the best case of linear search is when VAL is equal to the first element of the array**
- **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**

Binary Search

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

- 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

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

- **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

Algorithm for 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
                         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
```

Complexity of Binary Search Algorithm

- **The complexity of the binary search algorithm can be expressed as f(n), where n is the number of elements in the array.**

- **Thus, we can say that, in order to locate a particular value in the array, the total number of comparisons that will be made is given as**

$$2^{f(n)} > n \text{ or } f(n) = \log_2 n$$

Interpolation Search

Jump Search