K. J. Somaiya College of Engineering

(A Constituent College of Somaiya Vidyavihar University)

Batch: B4 **Roll No.:** 16010122805

Experiment No.___02___

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of Binary search/Max-Min algorithm

Objective: To learn the divide and conquer strategy of solving the problems of different types

CO to be achieved:

CO 2 Describe various algorithm design strategies to solve different problems and analyse Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://en.wikipedia.org/wiki/Binary_search_algorithm**
4. **https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Binary_search_algorithm.html**
5. **http://video.franklin.edu/Franklin/Math/170/common/mod01/binarySearchAlg.html**
6. **http://xlinux.nist.gov/dads/HTML/binarySearch.html**
7. **https://www.cs.auckland.ac.nz/software/AlgAnim/searching.html**

Pre Lab/ Prior Concepts:

Data structures

Historical Profile:

Finding maximum and minimum or Binary search are few problems those are solved with the divide-and-conquer technique. This is one the simplest strategies which basically works on dividing the problem to the smallest possible level.

Binary Search is an extremely well-known instance of divide-and-conquer paradigm. Given an ordered array of n elements, the basic idea of binary search is that for a given element , "probe" the middle element of the array. Then continue in either the lower or upper segment of the array, depending on the outcome of the probe until the required (given) element is reached.

New Concepts to be learned:
Number of comparisons, Application of algorithmic design strategy to any problem, Classical problem solving Vs Divide-and-Conquer problem solving.

Algorithm IterativeBinarySearch

```
int binary_search(int A[ ], int key, int imin, int imax)
//The algorithm takes as parameters an array A[1.. n] , the search key and lower-higher index pair of the array.
 // Output- The algorithm returns index of the search key in the given array, if it's present.
{
  // continue searching while [imin, imax] is not empty
  WHILE (imax >= imin)
    {
        // calculate the midpoint for roughly equal partition
        int imid = midpoint(imin, imax);
        IF(A[imid] == key)
            // key found at index imid
            return imid;
          // determine which subarray to search
        ELSE If (A[imid] < key)
            // change min index to search upper subarray
            imin = imid + 1;
          ELSE
            // change max index to search lower subarray
            imax = imid - 1;
    }
  // key was not found
  RETURN KEY_NOT_FOUND;
}
```

The Time and space complexity of Iterative Binary Search

- Space Complexity of Iterative Binary Search is O(1).
- Time Complexity in the best case is O(1).
- Time Complexity in the worst case is O(log N).

Algorithm Recursive Binary Search

```
int binary_search(int A[], int key, int imin, int imax)
//The algorithm takes as parameters an array A[1.. n] , the search key and lower-higher index pair
of the array.
 // Output- The algorithm returns index of the search key in the given array, if it's present.
{
  // test if array is empty
  IF (imax < imin)
    // set is empty, so return value showing not found
    RETURN KEY_NOT_FOUND;
  ELSE      {
                // calculate midpoint to cut set in half
                int imid = midpoint(imin, imax);
                 // three-way comparison
                IF (A[imid] > key)
                     // key is in ← lower subset
                     RETURN binary_search(A, key, imin, imid-1);
                ELSE IF (A[imid] < key)
                     // key is in → higher subset
                     RETURN binary_search(A, key, imid+1, imax);
                ELSE
                     // key has been found
                     RETURN imid;
              }
}
```

The Time and space complexity of Recursive Binary Search:

- Space Complexity of Recursive Binary Search is O(log N).
- Time Complexity:

  T(n) = a*T(n/b) + f(n)

  Where a=1,b=2 and f(n)=1

  Therefore, T(n) = 1*T(n/2) + 1

Algorithm StraightMaxMin:

```
 VOID StraightMaxMin (Type a[], int n, Type& max, Type& min)
// Set max to the maximum and min to the minimum of a[1:n].
{   max = min = a[1];

          FOR (int i=2; i<=n; i++)
          {
                    IF (a[i]>max)  then max = a[i];
                              IF (a[i]<min) min = a[i];
          }
}
```

Algorithm: Recursive Max-Min

```
 VOID MaxMin(int i, int j, Type& max, Type& min)
// A[1:n] is a global array. Parameters i and j are integers, 1 <= i <= j <= n.
 //The effect is to set  max and min to the largest and smallest values in a[i:j], respectively.
     {
       IF (i == j)
        max = min = a[i]; // Small(P)
       ELSE IF (i == j-1) { // Another case of Small(P)
              IF (a[i] < a[j])
                    max = a[j];
                   min = a[i];
              ELSE { max = a[i]; min = a[j];
              }
          ELSE {      Type max1, min1;
 // If P is not small  divide P into sub problems.  Find where to split the set.
              int mid=(i+j)/2;
             // solve the sub problems.
              MaxMin(i, mid, max, min);
             MaxMin(mid+1, j, max1, min1);
          // Combine the solutions.
            IF (max < max1) max = max1;
            IF (min > min1) min = min1;
    }
  }
```

The space complexity of Max-Min:

Space Complexity of Max-min is O(n)

Time complexity for Max-Min:

By General Equation (Recurrence):

T(n) = a * T(n/b) + f(n)

Where a=2, b=2, f(n)=2

Therefore,

T(n) = 2 * T(n/2) + 2

CONCLUSION:

We learned how to apply divide and conquer method to perform Binary search and max min algorithm. We also learned how to derived Recurrence relation for the given algorithm.