Binary Search

It searches for an element in a sorted dataset by repeatedly dividing the search interval in half.

Time Complexity

Best-case : O(1)
target is found in the central index.
Worst-case : O(log n)
target is either found in one of the two ends of the dataset or not found.

```
int binarySearch(int[] array, int target)
    {
        int start = 0;
        int end  = array.length – 1;
        boolean isAscending = array[start] < array[end];
        while(start<=end)
            {
                int mid = start + (end – start)/2;
                if(array[mid]==target)
                    return mid;
                if(isAscending)
                    {
                        if(target<array[mid])
                            {
                                end = mid – 1;
                            }
                        else
                            {
                                start = mid + 1;
                            }
                    }
                else
                    {
                        if(target>array[mid])
                            {
                                end = mid – 1;
                            }
                        else
                            {
                                start = mid + 1;
                            }
                    }
            }
        return -1;
    }
```

Binary Search in a Matrix$_{(n \times m)}$

Row-wise & Column-wise sorted Matrix

Time Complexity

Best-case : O(1)
target is found in the top-right position.
Worst-case : O(n+m)
target is found in the bottom-left position.

```
int[] binarySearch(int[][] matrix, int target)
    {
        int row = 0;
        int column = matrix[0].length – 1;
        while(row<matrix.length && column>=0)
            {
                if(matrix[row][column]==target)
                    return new int[]{row, column};
                else if(matrix[row][column]<target)
                    row++;
                else
                    column--;
            }
        return new int[]{-1,-1};
    }
```

Strictly-sorted Matrix

A sorted Matrix in which the last element in each row is smaller than the first element of the succeeding row.

Time Complexity

Best-case : O(1)

target is found in the middle of the matrix.i.e., in the middle cell of the middle row.

Worst-case : O(log n + log m)

target is found in one of the two ends of the first or last row of the matrix.

```
int[] binarySearchRow(int[][] matrix, int target)
        {
                int rStart = 0;
                int rEnd = matrix.length – 1;
                int cMid = matrix[0].length/2;
                while(rStart+1<rEnd)
                        {
                                int rMid = rStart + (rEnd – rStart)/2;
                                if(matrix[rMid][cMid] == target)
                                        return new int[]{rMid, cMid};
                                else if(matrix[rMid][cMid] < target)
                                        rStart = rMid;
                                else
                                        rEnd = rMid;
                        }
                if(matrix[rStart][cMid] == target)
                        return new int[]{rStart, cMid};
                if(matrix[rEnd][cMid] == target)
                        return new int[]{rEnd, cMid};
                if(target < matrix[rStart][cMid])
                        return binarySearch(matrix, rStart, 0, cMid – 1, target);
                else if(matrix[rStart][cMid] < target &&
                        target <= matrix[rStart][ matrix[rStart].length - 1] )
                        return binarySearch(matrix, rStart,  cMid + 1,
                                matrix[rStart].length – 1, target);
                else if(target < matrix[rEnd][cMid])
                        return binarySearch(matrix, rEnd, 0, cMid – 1, target);
                else
                        return binarySearch(matrix, rEnd, cMid + 1,
                                matrix[rEnd].length – 1, target);
        }
```

```
int[] binarySearch(int[][] matrix, int row, int cStart, int cEnd, int target)
        {
                int start = cStart;
                int end = cEnd;
                while(start <= end)
                        {
                                int mid = start + (end – start)/2;
                                if(matrix[row][mid] == target)
                                        return new int[]{row, mid};
                                else if(target < matrix[row][mid])
                                        end = mid - 1;
                                else
                                        start = mid + 1;
                        }
                return new int[]{-1,-1};
        }
```