```java
//Classical Binary Search
public class Solution {
  public int binarySearch(int[] array, int target) {
    if (array==null||array.length==0){
      return -1;
    }
    int left = 0;
    int right = array.length -1;

    while( left <= right){ //注意边界条件， 这边是 left小于等于right
      int mid = left + (right - left)/2;
      if (array[mid]==target){
        return mid;
      } else if (array[mid]<target){
        left = mid +1;
      } else {
        right = mid -1;
      }
    }
    return -1;
  }

}
```

```java
public class Solution {
  public int firstOccur(int[] array, int target) {
    if(array==null||array.length==0){
      return -1;
    }
    int left = 0;
    int right = array.length-1;
    while(left < right -1){
      int mid = left + (right -left)/2;
      if (array[mid]<=target){
        left = mid + 1;
      } else {
        right = mid;
      }
    }
    if (array[left]==target){
      return left;
    } else if (array[right]== target){
      return right;
    }
    return -1;
  }
}

```

```java
public class Solution {
  public int lastOccur(int[] array, int target) {
    if (array==null||array.length==0){
      return -1;
    }
    int left = 0;
    int right = array.length -1;
    while (left <right -1){
      int mid = left +(right - left )/2;
      if (array[mid]<= target){
        left = mid;
      } else {
        right = mid;
      }
    }
    if (array[right]== target){ //
      return right;
    } else if (array[left]== target){
      return left;
    }
    return -1;
  }
}

```

```java
public class Solution {
  public int closest(int[] array, int target) {
    if(array==null||array.length==0){
      return -1;
    }
    int left = 0;
    int right = array.length-1;
    while(left <right -1){
      int mid = left + (right - left)/2;
      if(array[mid]==target){
        return mid;
      } else if (array[mid] <target){
        left = mid;
      } else {
        right = mid;
      }
    }
    if(Math.abs(array[left] - target) <= Math.abs(array[right]- target)){
      return left;
    }
    return right;
  }
}

```

```java
// convert the 2D array to 1D array and do binary search
public class Solution {
  public int[] search(int[][] matrix, int target) {
    if(matrix.length==0||matrix[0].length==0){
      return new int[] {-1,-1};
    }
    int rows=matrix.length;
    int cols= matrix[0].length;
    int left = 0;
    //convert the 2D array to 1D array with rows*cols elements
    int right = rows*cols-1;
    while(left <= right){
      int mid = left + (right - left)/2;
      // convert the postion in 1d array back to row and col in 2D array.
      int row = mid /cols;
      int col = mid % cols;
      if(matrix[row][col]==target){
        return new int[]{row,col};
      } else if (matrix[row][col] <  target){
        left = mid +1;
      } else {
        right = mid -1;
      }
    }
    return new int[] {-1,-1};
  }
}
```

// 561. Find the Kth Element in The Matrix

```
Given a matrix, find the Kth index element.




example:

matrix:

1 3 4

5 6 7

8 9 10



k = 4 → return: 6
```

```java
public class Solution {
  public int findElement(int[][] matrix, int k) {
    int i=k/matrix[0].length; //row
    int j=k%matrix[0].length;//columns
    return matrix[i][j];
  }
}
```