说明

二分查找又称折半查找，它是一种效率较高的查找方法，使用前提必须是有序列表

优点是比较次数少，查找速度快，平均性能好；
其缺点是要求待查表为有序表，且插入删除困难。
因此，折半查找方法适用于不经常变动而查找频繁的有序列表。

代码

有两种方式实现： **递归** 和 **迭代**

测试代码

```
public static void main(String[] args) {
    int[] array = new int[100];
    for (int i = 1 ; i <= 100; i++){
        array[i-1] = i;
    }
    int key = 39;
    System.out.println(each(array, key));
    System.out.println(recursive(array, key, array.length - 1, 0));
}
```

迭代

```
/**
 * 查找key，在数组中的下标，没有就返回 -1
 * @param array
 * @param key
 * @return
 */
public static int each(int[] array, int key) {
    int low = 0;
    int high = array.length - 1;

     while (low <= high && key >= array[low] && key <= array[high]) {
         int mid = (low + high) / 2;
         if (key < array[mid]) {
             high = mid - 1;
```

```
        } else if (key > array[mid]) {
            low = mid + 1;
        } else {
            return mid;
        }
    }
    return -1;
}
```

递归

```
/**
 * 查找key，在数组中的下标，没有就返回 -1
 * @param array
 * @param key
 * @param high
 * @param low
 * @return
 */
public static int recursive(int[] array, int key, int high, int low){
    if (low > high || key < array[low] || key > array[high]){
        return -1;
    }
    int mid = (high + low) / 2;
    if (key < array[mid]){
        high = mid - 1;
        return recursive(array, key, high, low);
    }else if (key > array[mid]){
        low = mid + 1;
        return recursive(array, key, high, low);
    }else {
        return mid;
    }
}
```

扩展

另外还有两种算法：**差值查找** 和 **裴波那契查找**

这三种算法本质上是切入点不同，在不同的场景查询效率 各有优劣