二分查找

完全有序

二分查找

```
int[ ]  nums = {1,3,4,5,6,8,12,14,16}; target = 8
```

- 一般写法

```
public static int binarySearch(int[] nums,int target,int left, int right)
{
    //这里需要注意，循环条件
    while (left <= right) {
        //这里需要注意，计算mid
        int mid = left + ((right - left) >> 1);
        if (nums[mid] == target) {
            return mid;
        }else if (nums[mid] < target) {
            //这里需要注意，移动左指针
            left  = mid + 1;
        }else if (nums[mid] > target) {
            //这里需要注意，移动右指针
            right = mid - 1;
        }
    }
    //没有找到该元素，返回 -1
    return -1;
}
```

- 递归写法

```
public static int binarySearch(int[] nums,int target,int left, int right)
{
    if (left <= right) {
        int mid = left + ((right - left) >> 1);
        if (nums[mid] == target) {
            //查找成功
            return  mid;
        }else if (nums[mid] > target) {
            //新的区间,左半区间
            return binarySearch(nums,target,left,mid-1);
        }else if (nums[mid] < target) {
            //新的区间，右半区间
            return binarySearch(nums,target,mid+1,right);
        }
    }
```

```
    //不存在返回-1
    return -1;
}
```

搜索插入位置

Leetcode 35

```
//输入: [1,3,5,6], 7
//输出: 4
class Solution {
    public int searchInsert(int[] nums, int target) {

        int left = 0, right = nums.length-1;
        //注意循环条件
        while (left <= right) {
            //求mid
            int mid = left + ((right - left ) >> 1);
            //查询成功
            if (target == nums[mid]) {
                return mid;
            //右区间
            } else if (nums[mid] < target) {
                left = mid + 1;
            //左区间
            } else if (nums[mid] > target) {
                right = mid - 1;
            }
        }
        //返回插入位置
        return left;
    }
}
```

查找第一个元素

leetcode 34

```
class Solution {
    public int[] searchRange (int[] nums, int target) {
         int upper = upperBound(nums,target);
         int low = lowerBound(nums,target);
         //不存在情况
         if (upper < low) {
             return new int[]{-1,-1};
         }
         return new int[]{low,upper};
    }
    //计算下边界
```

```
    int lowerBound(int[] nums, int target) {
        int left = 0, right = nums.length - 1;
        while (left <= right) {
            //这里需要注意，计算mid
            int mid = left + ((right - left) >> 1);
            if (target <= nums[mid]) {
                //当目标值小于等于nums[mid]时，继续在左区间检索，找到第一个数
                right = mid - 1;

            }else if (target > nums[mid]) {
                //目标值大于nums[mid]时，则在右区间继续检索，找到第一个等于目标值的
数
                left = mid + 1;

            }
        }
        return left;
    }
    //计算上边界
    int upperBound(int[] nums, int target) {
        int left = 0, right = nums.length - 1;
        while (left <= right) {
            int mid = left + ((right - left) >> 1);
            if (target >= nums[mid]) {
                left = mid + 1;
            }else if (target < nums[mid]) {
                right = mid - 1;
            }
        }
        return right;
    }
}
```

查找最后一个元素

同上 leetcode 34

找出第一个大于目标元素的索引

```
public static int lowBoundnum(int[] nums,int target,int left, int right) {
  while (left <= right) {
      //求中间值
      int mid = left + ((right - left) >> 1);
      //大于目标值的情况
      if (nums[mid] > target) {
            //返回 mid
          if (mid == 0 || nums[mid-1] <= target) {
              return mid;
          }
          else{
              right = mid -1;
```

```
            }

        } else if (nums[mid] <= target){
            left = mid + 1;
        }
    }
    //所有元素都小于目标元素
    return -1;
}
```

找出第一个小于目标元素的索引

```
public static int upperBoundnum(int[] nums,int target,int left, int right)
{
    while (left <= right) {

        int mid = left + ((right - left) >> 1);
          //小于目标值
        if (nums[mid] < target) {
            //看看是不是当前区间的最后一位，如果当前小于，后面一位大于，返回当前值即可
            if (mid == right || nums[mid+1] >= target) {
                return mid;
            }
            else{
                left = mid + 1;
            }

        } else if (nums[mid] >= target){
            right = mid - 1;
        }
    }
    //没有查询到的情况
    return -1;
}
```

不完全有序

查找目标元素（不含重复元素）

leetcode 33

```
class Solution {
    public int search(int[] nums, int target) {
        //左右指针
        int left = 0;
        int right = nums.length - 1;
        while (left <= right) {
            int mid = left+((right-left)>>1);
```

```
            if (nums[mid] == target) {
                return mid;
            }
            //落在同一数组的情况，同时落在数组1 或 数组2
            if (nums[mid] >= nums[left]) {
                //target 落在 left 和 mid 之间，则移动我们的right，完全有序的一个区
间内查找
                if (nums[mid] > target && target >= nums[left]) {
                        right = mid - 1;
                // target 落在right和 mid 之间，有可能在数组1， 也有可能在数组2
                } else if (target > nums[mid] || target < nums[left]) {
                        left = mid + 1;
                }
            //不落在同一数组的情况，left 在数组1， mid 落在 数组2
            }else if (nums[mid] < nums[left]) {
                //有序的一段区间，target 在 mid 和 right 之间
                if (nums[mid] < target && target <= nums[right]) {
                    left = mid + 1;
                // 两种情况，target 在left 和 mid 之间
                } else if (target < nums[mid] || target > nums[right]) {
                    right = mid - 1;
                }
            }
        }
        //没有查找到
        return -1;
    }
}
```

查找目标元素（含重复元素）

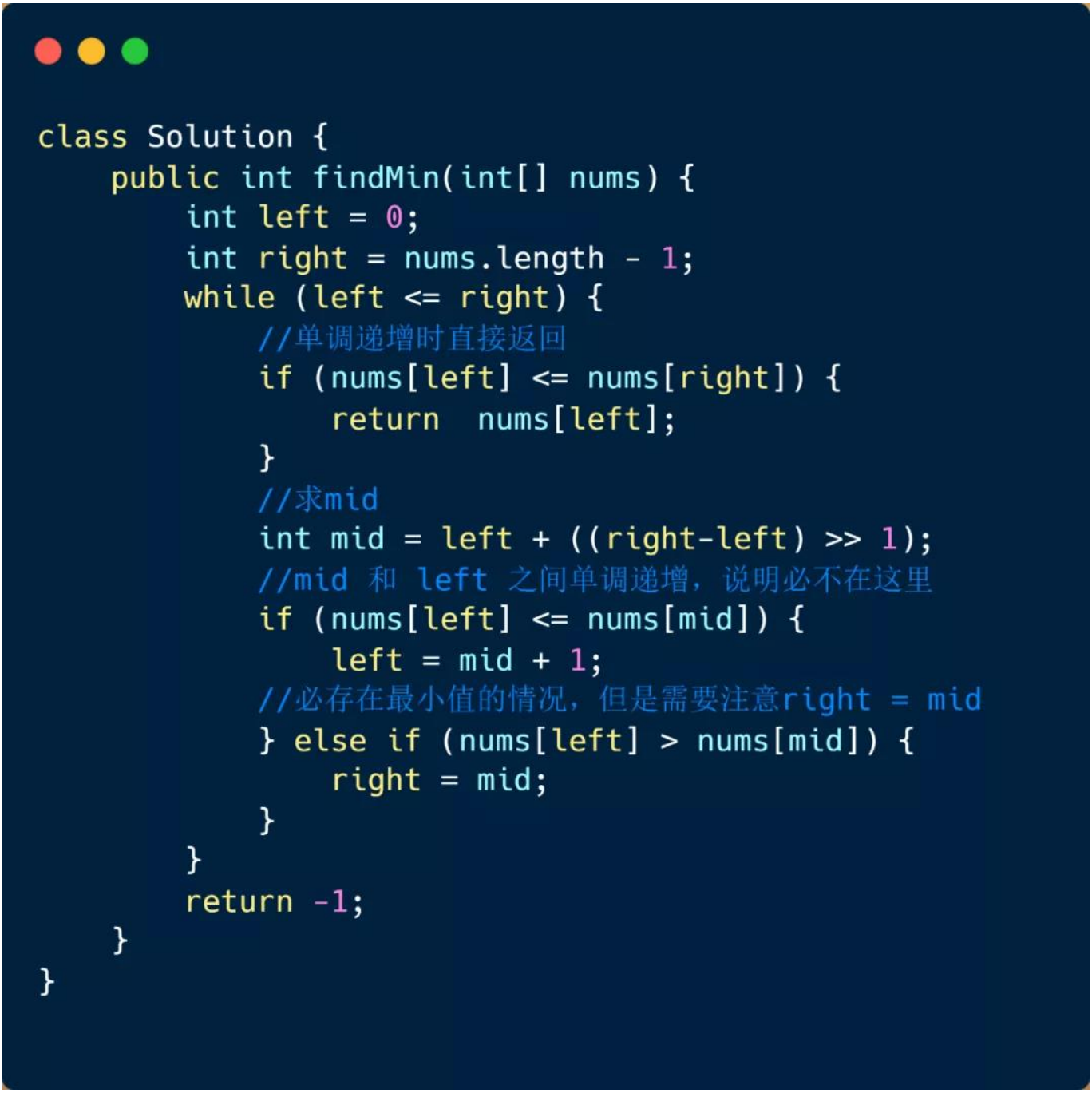
leetcode 81

```java
class Solution {
    public boolean search(int[] nums, int target) {
     int left = 0;
        int right = nums.length - 1;
        while (left <= right) {
            int mid = left+((right-left)>>1);
            if (nums[mid] == target) {
                return true;
            }
            if (nums[mid] == nums[left]) {
                left++;
                continue;
            }
            if (nums[mid] > nums[left]) {
                if (nums[mid] > target && target >= nums[left]) {
                        right = mid - 1;
                } else if (target > nums[mid] || target < nums[left]) {
                        left = mid + 1;
                }

            }else if (nums[mid] < nums[left]) {
                if (nums[mid] < target && target <= nums[right]) {
                    left = mid + 1;
                } else if (target < nums[mid] || target > nums[right]) {
                    right = mid - 1;
                }
            }
        }
        return false;
    }
}
```

寻找最小值

leetcode 153



```
class Solution {
    public int findMin(int[] nums) {
        int left = 0;
        int right = nums.length - 1;
        while (left <= right) {
            //单调递增时直接返回
            if (nums[left] <= nums[right]) {
                return  nums[left];
            }
            //求mid
            int mid = left + ((right-left) >> 1);
            //mid 和 left 之间单调递增，说明必不在这里
            if (nums[left] <= nums[mid]) {
                left = mid + 1;
            //必存在最小值的情况，但是需要注意right = mid
            } else if (nums[left] > nums[mid]) {
                right = mid;
            }
        }
        return -1;
    }
}
```

二维数组

寻找目标元素

```
class Solution {
  public boolean searchMatrix(int[][] matrix, int target) {
    if (matrix.length == 0) {
        return false;
    }
    //行数
    int row = matrix.length;
    //列数
    int col = matrix[0].length;
    int left = 0;
```

```
        //行数乘列数 - 1，右指针
        int right = row * col - 1;
        while (left <= right) {
            int mid = left+ ((right-left) >> 1);
            //将一维坐标变为二维坐标
            int rownum = mid / col;
            int colnum = mid % col;
            if (matrix[rownum][colnum] == target) {
                return true;
            } else if (matrix[rownum][colnum] > target) {
                right = mid - 1;
            } else if (matrix[rownum][colnum] < target) {
                left = mid + 1;
            }
        }
        return false;
    }
}
```