- 1 二分查找的基本用法
  - 1.1 一般用法
  - 1.2 其他的变形
- 2 二分查找的经典题目
  - 2.1 287 寻找重复数

1 二分查找的基本用法

- 二分查找也称为折半查找，每次都能将查找区间减半，这种折半特性的算法时间复杂度为 O(logN)
- 计算mid有两种方法：

```
mid = (lo+high)/2;
mid = lo+(hi-lo)/2;
```

- 一般由于第一种用法很可能会超出整型可表示的最大范围，所以会存在问题，所以一般推荐第二种写法
- 一般的循环条件都是 `while(lo<high)` 或者 `while(lo<=high)` ,这两种条件对应不同的mid取值
  - `while(lo<=high):lo=mid+1;high=mid-1;`
  - `while(lo<high):lo=mid;high=mid-1;`
  - `while(lo<high):lo=mid-1;high=mid;`
  - 否则循环无法退出

1.1 一般用法

```
搜索某个数组nums(不包含重复元素)中是否包含target,如果包含返回索引，否则返回-1
输入:[1,2,3,5,7] 7
输出:4
```

- 经典解法：

```
var binarySearch = function(nums, target) {
    let lo = 0, h = nums.loength - 1;
    while (lo <= h) {
        let mid = lo + (h - lo) / 2;
        if (nums[mid] == target) {
            return m;
        } else if (nums[mid] > key) {
            h = mid - 1;
        } else {
            lo = mid + 1;
        }
    }
    return -1;
}
```

- 变种1：包含重复元素，找到最小的target所在的索引

搜索某个数组nums(不包含重复元素)中是否包含target,如果包含返回target的第一个索引，否则返回-1
输入:[1,2,3,3,5,7,7,7,7] 7
输出:5

- 解法：查找key的最左边位置，返回lo
  - 只有当 nums[mid]<target 时，才改变左边的位置，这样lo始终就指向第一个不小于target的值的索引

```
var binarySearch = function(nums, target) {
    let lo = 0, h = nums.length - 1;
    while (lo <= h) {
        console.log(lo,h);
        let mid = parseInt(lo + (h - lo) / 2);
        // 重点是如果nums[mid]==target,则不改变左边的位置，只改变右边
        // 只有当nums[mid]<target时，才改变左边的位置，这样lo始终就指向第一个不小于target的值的索引
        if (nums[mid] < target) {
            lo = mid + 1;
        } else {
            h = mid - 1;
        }
    }
    return lo;
}
```

- 变种2：包含重复元素，找到最后一个target所在的索引

输入:[1,2,3,3,5,7,7,7,7] 7
输出:8

- 解法：查找key的最右边位置，返回lo-1
  - 当 `nums[mid]<=target` 时，改变lo，这样lo-1始终就指向最后一个target的值的索引
  - 如果是返回第一个大于指定值的元素的元素，则直接返回lo即可

```
var binarySearch = function(nums, target) {
    let lo = 0, h = nums.length - 1;
    while (lo <= h) {
        let mid = parseInt(lo + (h - lo) / 2);
        if (nums[mid] <= target) {
            lo = mid + 1;
        } else {
            h = mid - 1;
        }
    }
    return lo-1;
}
```

1.2 其他的变形

- 简单的
  - 167,35,367,69,744
- 中等
  - 旋转数组：33,153，这两者需要在取得mid之后判断左右是否为有序数组
  - 需要递归的：50,486，难一些，感觉没有看懂
  - 29,34,540
- 困难
  - 04:解法比较不常见

2 二分查找的经典题目

2.1 287 寻找重复数

给定一个包含 n + 1 个整数的数组 nums，其数字都在 1 到 n 之间（包括 1 和 n），可知至少存在一个重复的整数。假设只有一个重复的

示例 1:

输入: [1,3,4,2,2]
输出: 2
示例 2:

输入: [3,1,3,4,2]
输出: 3

说明:

不能更改原数组（假设数组是只读的）。
只能使用额外的 O(1) 的空间。
时间复杂度小于 O(n2) 。
数组中只有一个重复的数字，但它可能不止重复出现一次。

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/find-the-duplicate-number

解法1：双重循环暴力解法

时间复杂度O(n2),不满足条件 空间复杂度O(1)

外层循环依次取出每一个元素，内存循环负责将该元素与其后的所有元素进行比较，一旦发现具有相同的值，则说明该元素是重复的，返回该元素即可

```
function findDuplicate(nums){
    for(let i=0; i<nums.length-1; i++>){
        for(let j = i+1; j<nums.length; j++){
            if(nums[i] === nums[j]){
                return nums[i];
            }
        }
    }
    // 未发现重复值则返回false
    return false;
}
```

解法2：哈希表

时间复杂度O(n) 空间复杂度O(n),不满足条件

依次遍历每个元素，创建一个哈希表保存每个元素出现的次数，然后遍历哈希表，一旦次数大于1，则返回该值

解法3：二分查找

时间复杂度O(nlogn) 空间复杂度O(1)

分析：

- 由题目可知数组的长度为n+1,并且数组内元素的范围为[1,n],则如果1,...,n分别出现一次，还空了一个位置，说明一定有一个元素是重复的
- 令lo=1,hi=n,求得中位数mid,遍历数组得到小于等于mid的元素的数量count，如果mid之前的元素不重复，则小于等于mid的元素最多只有mid个，即
  - 如果count>mid,则说明重复元素在左边，则令hi=mid;
    - 例如[1,3,2,3] lo=1,hi=3,mid=2,count=2
    - 得到 lo=2,hi=3,mid=2,count=2
    - 得到lo=3,hi=3 lo=hi不满足条件则退出，返回3
  - 反之，则重复元素在右边，lo=mid+1;

```
function findDuplicate(nums){
    let lo = 1;
    let hi = nums.length-1;
    while(lo < hi>){
        let mid = parseInt((lo+hi)/2);
        let count = 0;
        nums.forEach((num) => {
            num <= mid ? ++count : count;
        })
        if(count>mid){
            hi = mid;
        }else{
            lo = mid+1;
        }
    }
    return lo;
}
```