基于二分查找--剑指Offer\_53\_在排序数组中查找数字

# 题目一：数字在排序数组中出现的次数

## 题目介绍

统计一个数字在排序数组中出现的次数。

## 思路介绍

两个方法：

方法1：寻找到一个与target相等的值，然后利用while循环左右扫描；

方法2：利用**二分查找**分别找到做边界和右边界的索引，然后相减加1即可。

## Java代码

### 方法1：

/\*\*

\* 方法1：**二叉查找基础上**

\* 若中间位置的值与target相等，向两边顺序扫描，直到向左到low或非target，向右到high或非target。

\* 这种算法的在全是target的情况下时间复杂度为O(n)，与顺序扫描一样。

\*/

public int GetNumberOfK(int [] nums , int target) {

int low = 0,high = nums.length-1;

while(low <= high){

int mid = (low+high)>>1;

if(target < nums[mid]){

high = mid-1;

}else if(target > nums[mid]){

low = mid + 1;

}else{//在二分查找的基础上，改变了相等的情况

**int mid1 = mid;**

**while(mid1 >= low&&nums[mid1] == target) mid1--;**

**while(mid <= high&&nums[mid] == target) mid++;**

**return mid-mid1-1;**

}

}

return 0;//不存在情况下，返回0

}

### 方法2：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*方法二\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* 方法2：二分查找基础上

\* 由于方法1的时间复杂度为O(n)，而二分查找的复杂度为O(logn);

\* 如果只是寻找target连续的第一个位置和最后一个位置，作差加1就是次数，

\* 这样时间复杂度就是O(logn)

\* 二分查找的改进：

\* nums[mid]>target和nums[mid]<target都不变，

\* nums[mid]==target需要好好考虑下条件：

\* 需要返回mid的两种情况：

\* **mid==low或mid==high；或者nums[mid+1]!=target或nums[mid-1]!=target**

\* 其他情况需要递归：high=mid-1或low=mid+1

\*/

public int getNumberOfK(int [] nums , int target) {

if(nums==null||nums.length == 0) return 0;

int firstIndex = getFirstIndexOfK(nums,target);

int lastIndex = getLastIndexOfK(nums,target);

System.out.println(firstIndex+","+lastIndex);

return **(firstIndex != -1 && lastIndex != -1) ? (lastIndex-firstIndex +1) : 0;**

}

/\*\*

\* 获取第一个索引位置

\* @param nums

\* @param target

\* @return

\*/

public int getFirstIndexOfK(int[] nums,int target){

int low = 0,high = nums.length-1;

while(low <= high){

int mid = (low+high)>>1;//1. 位移效率高；2. 求mid一定在while循环内，前几次在while循环外，导致无限循环

if(target < nums[mid]){

high = mid-1;

}else if(target > nums[mid]){

low = mid + 1;

}else{

//找到第一个两种情况：1.mid==low，到达边界；2. 下一个元素不是target；其他情况需要继续循环

// if(mid == low) return mid;//保证后续mid>low

// if(nums[mid-1] != target) return mid;

**if(mid == low||nums[mid-1]!=target) return mid;**

**high = mid-1;**

}

}

return -1;

}

/\*\*

\* 获取最后一个索引位置

\* @param nums

\* @param target

\* @return

\*/

public int getLastIndexOfK(int[] nums,int target){

int low = 0,high = nums.length-1;

while(low <= high){

int mid = (low+high)>>1;

if(target < nums[mid]){

high = mid-1;

}else if(target > nums[mid]){

low = mid + 1;

}else{

// if(mid == high) return mid;//保证后续mid<high

// if(nums[mid+1] != target) return mid;

**if(mid == high||nums[mid+1]!=target) return mid;**

**low = mid+1;**

}

}

return -1;

}

# 题目二：找出0 -- n-1中缺失的数字

## 题目介绍

题目描述：找出0 -- n-1中缺失的数字

长度为n-1的递增排序数组中所有数字都是唯一的，且都在0 -- n-1范围内。

有且只有一个数字不在该数组中，找出这个数字。

## 思路介绍

分析：方法1：求和判断，时间复杂度为O(n);

方法2：二分查找，第一个索引和元素值不相等的索引即可。

## Java代码

//测试

@Test

public void test(){

int[] nums1 = {0,1,2,3,4,5,7,8,9,10};

int[] nums = new int[100];

for(int k = 1;k<100;k+=3){

int num = 0;

for(int i = 0;i < 100;i++){

if(i == k) nums[i] = ++num;

else nums[i] = num;

num++;

}

int missingNumber = getMissingNumber(nums);

System.out.println("missingNumber = " + missingNumber);

}

}

/\*\*

\* 利用二分法查找确实的数字

\* 0 -- n-1共n个数字，有一个长度为n-1的排序数组，每个数字都是惟一的，

\* 找出那个没有出现的数字。

\* @param nums

\* @return

\*/

public int **getMissingNumber**(int[] nums){

if(nums == null || nums.length == 0) return -1;

int low = 0,high = nums.length-1;

while(low <= high){

int mid = (low + high)>>1;

if(nums[mid] == mid) low = mid+1;

else{//只可能nums[mid]<mid；不可能nums[mid]>mid

**if(mid == low) return low;**

**if(nums[mid-1]== mid-1) return mid;**

**high = mid-1;**

}

}

return -1;

}

# 题目三：

## 题目介绍

题目描述：数组中数值和下标相等的元素

\* 数组是单调递增的，每个元素都是整数且是唯一的。

\* 找出任意一个元素值与下标相等的元素，没有则返回-1。

## 思路介绍

基于二分查找，将nums[mid]与mid先比较，分成三种情况。

## Java代码

/\*\*

\* 测试

\*/

@Test

public void test(){

int[] nums = {-9,-2,1,2,3,5,7,8,10};

int number = getNumberSameAsIndex(nums);

System.out.println("number = " + number);

}

/\*\*

\*

\* 方法：利用二分查找

\* 相对于二分查找，只是更改了判断条件。

\* 基本的二分查找是nums[mid]与target作比较，

\* 这里是nums[mid]与mid作比较。

\*/

public int getNumberSameAsIndex(int[] nums){

if(nums == null||nums.length == 0) return -1;

int low = 0,high = nums.length-1;

while(low <= high){

int mid = (low+high)>>1;

if(nums[mid] > mid){

high = mid - 1;

}else if(nums[mid]<mid){

low = mid + 1;

}else{

return mid;

}

}

return -1;

}