拉勾教育
—互联网人实战大学—

归并排序+二分查找：寻找两个正序数组的中位数

困难/归并排序、二分查找

学习目标

拉勾教育
—互联网人实战大学—

- 理解归并排序算法的思想与流程
- 掌握二分查找算法的原理与应用

题目描述

给定两个大小为 m 和 n 的正序（从小到大）数组 nums1 和 nums2，请你找出这两个正序数组的中位数。

进阶：你能设计一个时间复杂度为 O(log(m + n))的算法解决此问题吗？

提示：0<= m、n <=1000　　m+n>=1　　-10^6 <= nums1[i]、nums2[i] <= 10^6

输入：nums1 = [1, 3]，nums2 = [2]
输出：2.0
解释：合并数组=[1,2,3]，因此中位数是 2

输入：nums1 = [1, 2]，nums2 = [3, 4]
输出：2.5
解释：合并数组=[1,2,3,4]，因此中位数是 (2 + 3)/2 = 2.5

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

寻找两个数组的中位数

- 两个数组都是正序数组
- 寻找两个数组所有元素的中位数

nums1

nums2

进阶要求

- 算法的时间复杂度为O(log(m+n))

宽松限制

- m + n >= 1 ：数组不同时为空（中位数一定存在）

一. Comprehend 理解题意

细节问题

- 某个数组可能为空
- 元素总数为偶数时，中位数是中间两数平均值（不一定是整数）

二. Choose 数据结构及算法思维选择

数据结构选择

- 输入的数据类型为两个整形数组
- 输出的数据类型为一个浮点数
- 因为需要处理的是两个有序的整数集合，我们采用数组作为数据结构

二. Choose 数据结构及算法思维选择

算法思维选择

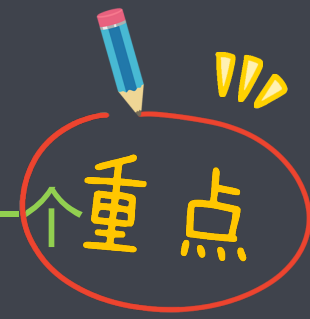
- 题目要求寻找中位数，我们可以对两个数组进行合并和排序
- 在排序后的数组中我们可以很容易地找到中位数
- 合并、排序我们有没有好的方案选择呢？

nums1

nums2

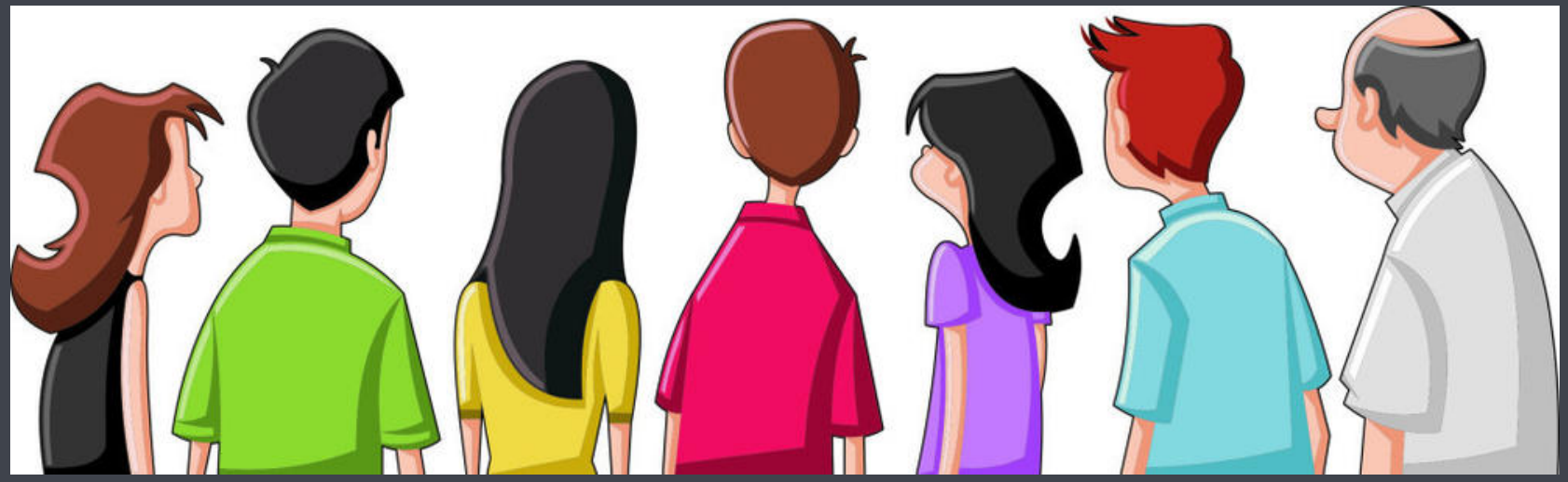
二. Choose 数据结构及算法思维选择

知识点：归并排序（Merge Sort）



将序列拆成两个或两个以上分别排序，然后再合并成一个

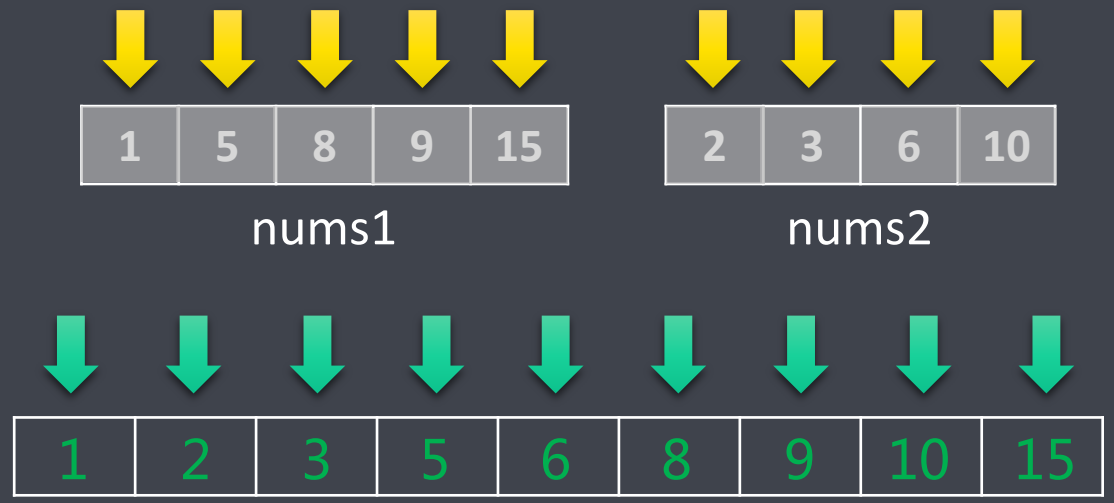
- 采用分治思想，基于归并操作的一种稳定排序算法
- 稳定排序：对于关键字相等的元素在排序前后相对顺序保持不变



二. Choose 数据结构及算法思维选择

知识点：归并排序

- 寻找两个有序数组的最小值
  - 即两个数组最小值中较小的那个
  - 放入合并后数组的第一位
- 寻找两个数组第2小的值
- 寻找两个数组第3小的值
- 寻找两个数组第4小的值
- 依次进行直到结束



二. Choose 数据结构及算法思维选择

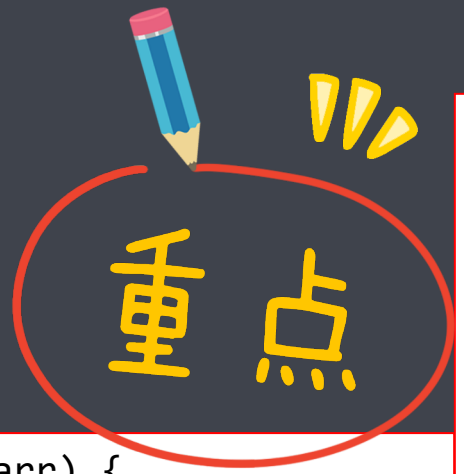
知识点：归并排序

- 首先将数组拆分成两部分
- 对这两部分分别递归排序
  - 元素个数大于1，继续拆分
  - 只有一个元素时无需排序，结束递归
- 在对有序数组进行两两合并

二. Choose 数据结构及算法思维选择

知识点：归并排序

- Java代码实现




```java
public static int[] mergeSort(int[] arr) {
    if (arr.length < 2) return arr;
    //计算中间位置
    int mid = arr.length / 2;
    //分解为左右两部分,分别排序
    int[] left = Arrays.copyOfRange(arr, 0, mid);
    left = mergeSort(left);
    int[] right=Arrays.copyOfRange(arr,mid,arr.length);
    right = mergeSort(right);
    //合并两个排序后的数组为一个数组
    return merge(left, right);
}
```

```java
private static int[] merge(int[] l, int[] r) {
    int[] result = new int[l.length + r.length];
    int lIndex = 0;
    int rIndex = 0;
    for (int i = 0; i < result.length; i++) {
        if (lIndex < l.length && rIndex < r.length) {
            if (l[lIndex] <= r[rIndex]) {
                result[i] = l[lIndex++];
            } else {
                result[i] = r[rIndex++];
            }
        } else if (lIndex >= l.length) {
            result[i] = r[rIndex++];
        } else {
            result[i] = l[lIndex++];
        }
    }
    return result;
}
```

二. Choose 数据结构及算法思维选择

知识点：归并排序

- 时间复杂度
  - 需要递归的将数组切割logn次，然后进行两两归并，时间复杂度为O(nlogn)
- 空间复杂度
  - 递归深度是O(logn)
  - 每次递归在合并时需额外辅助空间，长度与待排序的数组长度相等
  - 每次递归都会释放掉所占的辅助空间，最大辅助空间为O(n)
  - 所以空间复杂度为O(n+logn)=O(n)

二. Choose 数据结构及算法思维选择

在本题中运用归并排序

- 解题思路
  - 题目中给定两个有序数组
  - 直接使用归并排序的最后一步对两个数组进行合并即可
- 时间复杂度
  - 需要对两个数组各浏览一遍，进行m+n-1次比较，时间复杂度为O(m+n)
- 空间复杂度
  - 我们需要额外的空间存储合并排序后的数组，复杂度为 O(m+n)

二. Choose 数据结构及算法思维选择

算法思维优化

- 题目要求寻找中位数，其实我们并不需要得到合并后的有序数组
- 归并排序进行到一半的时候，我们其实已经找到了中位数
- 那么算法可以被改进为遍历两个有序数组num1和nums2：
  - 若|nums1|+|nums2| 为奇数，寻找第$\frac{|nums1|+|nums2|+1}{2}$小的数
  - 若|nums1|+|nums2| 为偶数，寻找第$\frac{|nums1|+|nums2|}{2}$以及$\frac{|nums1|+|nums2|+2}{2}$小的数

三. Code 基本解法及编码实现

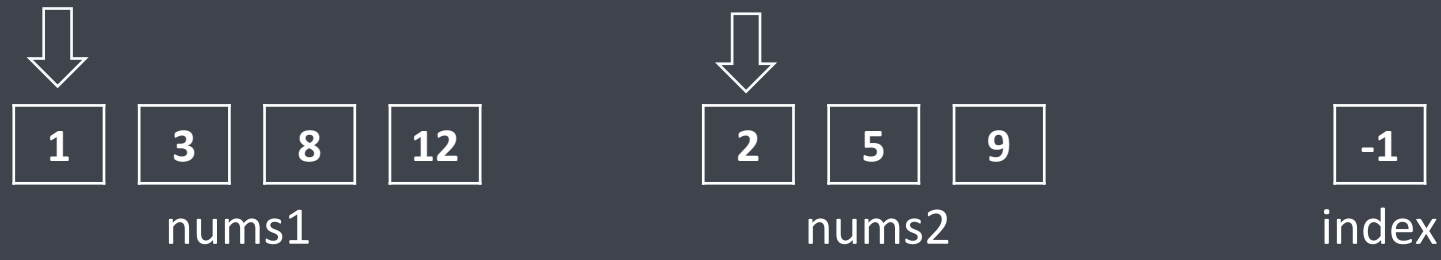
解题思路剖析

- 遍历有序数组nums1和nums2，寻找中位数
- 使用归并排序的思想，但是并不真正构建归并后的数组

三. Code 基本解法及编码实现

解题思路剖析

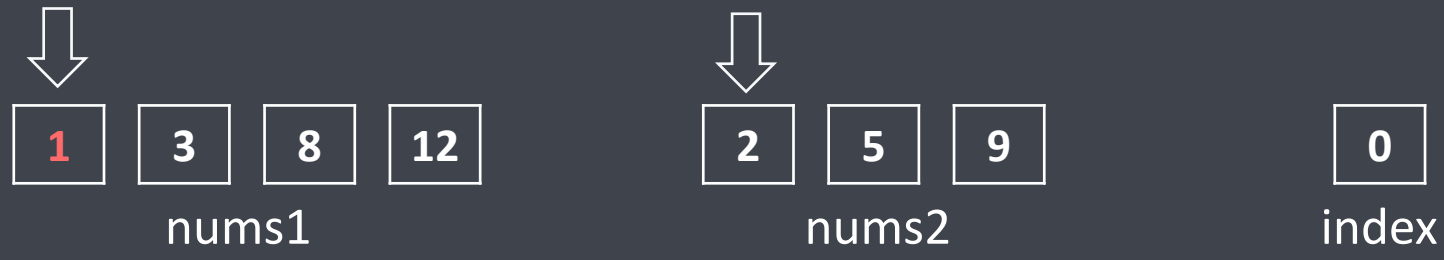
- 遍历有序数组nums1和nums2，寻找中位数
- |nums1|+|nums2|= 4+3 = 7, 中位数为第 (7+1)/2=4 小的数



三. Code 基本解法及编码实现

解题思路剖析

- 遍历有序数组nums1和nums2，寻找中位数
- |nums1|+|nums2|= 4+3 = 7, 中位数为第 (7+1)/2=4 小的数

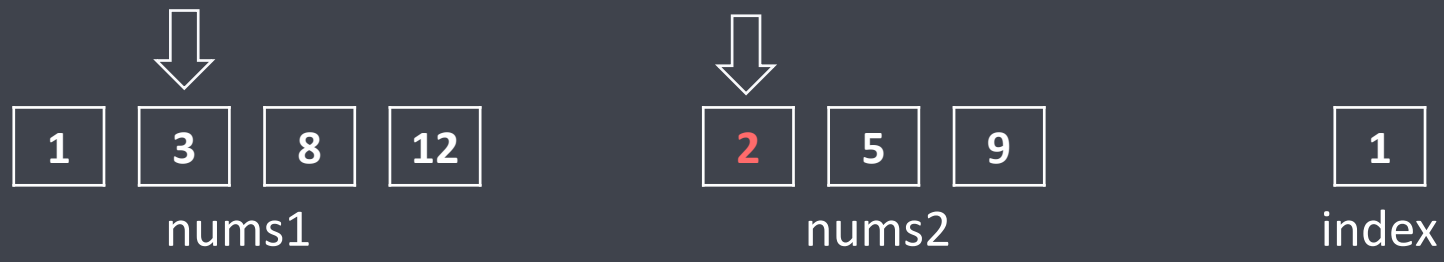


- 1 < 2 最小的数字为1

三. Code 基本解法及编码实现

解题思路剖析

- 遍历有序数组nums1和nums2，寻找中位数
- |nums1|+|nums2|= 4+3 = 7, 中位数为第 (7+1)/2=4 小的数



- 2 < 3 第2小的数字为2

三. Code 基本解法及编码实现

解题思路剖析

- 遍历有序数组nums1和nums2，寻找中位数
- |nums1|+|nums2|= 4+3 = 7, 中位数为第 (7+1)/2=4 小的数

- 3 < 5 第3小的数字为3

三. Code 基本解法及编码实现

解题思路剖析

- 遍历有序数组nums1和nums2，寻找中位数
- |nums1|+|nums2|= 4+3 = 7, 中位数为第 (7+1)/2=4 小的数

- 5 < 8 第4小的数字为5，返回中位数5

三. Code 基本解法及编码实现

复杂度分析

- 时间复杂度
  - 我们需要多次比较两个数组中元素的大小
  - 因为只需要找到中间位置的元素，并不需要完成整个归并，总的比较次数为 $\left\lfloor \frac{m+n}{2} \right\rfloor + 1$
  - 时间复杂度为O(m+n)
- 空间复杂度
  - 空间消耗方面我们只需要使用常数级的变量空间，因此空间复杂度为O(1)

三. Code 基本解法及编码实现

Java编码实现

```java
public double findMedianSortedArrays(int[] nums1, int[] nums2) {
    int m = nums1.length;
    int n = nums2.length;
    //定义指针p1,p2 分别指代nums1和nums2的当前元素
    int p1 = 0, p2 = 0;
    //定义中位数m1和m2，指代当前第i-1大的数和第i大的数
    int m1 = 0, m2 = 0;
    for (int i = 0; i <= (m + n) / 2; i++) {
        m1 = m2; //指针右移
        //若nums1未处理完，并且 nums2已处理完 或 p1元素小于p2元素
        //则第i大的元素为nums1当前元素
        if (p1 < m && (p2 >= n || nums1[p1] < nums2[p2])) {
            m2 = nums1[p1++];
        } else { //否则第i大的元素为nums2当前元素
            m2 = nums2[p2++];
        }
    }
    //若m+n为奇数，返回m2，若m+n为偶数，返回(m1+m2)/2.0
    if (((m + n) % 2) == 0)
        return (m1 + m2) / 2.0;
    else
        return m2;
}
```

执行结果： 通过 显示详情 >

执行用时： 3 ms , 在所有 Java 提交中击败了 69.10% 的用户

内存消耗： 41 MB , 在所有 Java 提交中击败了 24.82% 的用户

四. Consider 思考更优解

是否存在无效代码或者无效空间消耗

是否有更好的算法思维

- 当前我们浏览了输入nums1和nums2总数据的一半
- 有没有可能浏览更少的元素就可以确定中位数？

四. Consider 思考更优解

中位数的性质：中位数可以将数组分成2份

- 小于等于中位数的元素
- 大于中位数的元素

这两个数组的元素个数之差小于等于1

那么寻找中位数的问题就可以变为寻找满足下列条件的分界线：

1. 分界线左边的元素都小于等于分界线右边的元素
2. 0 <=分界线左边的元素个数-分界线右边的元素个数<=1

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

中位数的性质：中位数可以将数组分成2份

- 现在我们要寻找两个有序数组的中位数
- 也可以归结为定位该分界线的问题

那么我们该如何寻找该分界线呢？

那么寻找中位数的问题就可以变为寻找满足下列条件的分界线：

1. 分界线左边的元素都小于等于分界线右边的元素
2. 0 <=分界线左边的元素个数-分界线右边的元素个数<=1

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

中位数的性质：中位数可以将数组分成2份

- 现在我们要寻找两个有序数组的中位数
- 也可以归结为定位该分界线的问题

那么寻找中位数的问题就可以变为寻找满足下列条件的分界线：

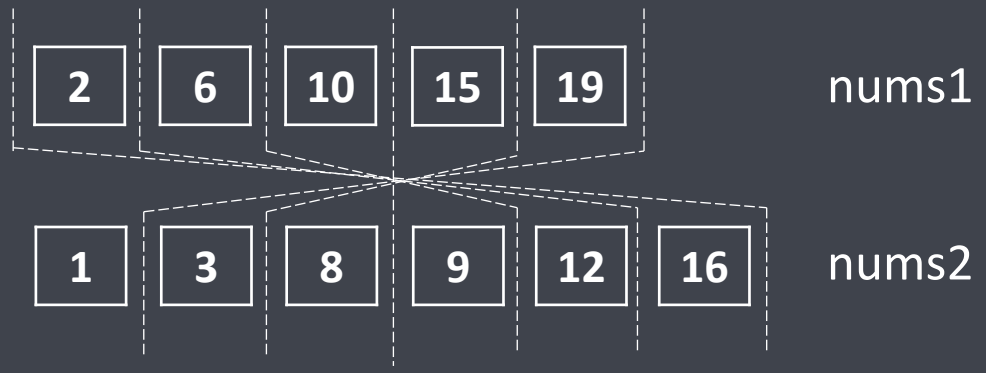
1. 分界线左边的元素都小于等于分界线右边的元素
2. 0 <=分界线左边的元素个数-分界线右边的元素个数<=1

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

中位数的性质：中位数可以将数组分成2份

- 现在我们要寻找两个有序数组的中位数
- 也可以归结为定位该分界线的问题



那么寻找中位数的问题就可以变为寻找满足下列条件的分界线：

1. 分界线左边的元素都小于等于分界线右边的元素
2. 0 <=分界线左边的元素个数-分界线右边的元素个数<=1

四. Consider 思考更优解

关键知识点：二分查找（折半查找）

使用折半的方式在有序数组中查找某一特定元素

- 从数组的中间元素开始查找，如果中间元素等于目标元素，查找结束；
- 如果中间元素小于目标元素，则在右半部分继续查找；
- 如果中间元素大于目标元素，则在左半部分继续查找；
- 如果在某一步骤数组为空，则代表找不到，查找结束；

每一次比较都使查找范围缩小一半

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

关键知识点：二分查找

示例

- 在右侧有序数组中寻找元素22
- 初始查找范围为　0~9

0	1	2	3	4	5	6	7	8	9
2	**5**	**6**	**9**	**12**	**22**	**36**	**55**	**72**	**81**

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

关键知识点：二分查找

示例

- 在右侧有序数组中寻找元素22
- 初始查找范围为　0~9
- 首先选取中间元素12与22进行对比，12 < 22
- 所以目标元素所在的位置一定在12右侧，查找范围变为5~9

0	1	2	3	4	5	6	7	8	9
2	5	6	9	12	22	36	55	72	81

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

关键知识点：二分查找

示例

- 在右侧有序数组中寻找元素22
- 当前查找范围为　5~9

0	1	2	3	4	5	6	7	8	9
2	5	6	9	12	22	36	55	72	81

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

关键知识点：二分查找

示例

- 在右侧有序数组中寻找元素22
- 当前查找范围为　5~9
- 选取中间元素55与22进行对比，55 > 22
- 目标元素所在的位置一定在55左侧，查找范围变为5~6

0	1	2	3	4	5	6	7	8	9
2	5	6	9	12	22	36	55	72	81

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

关键知识点：二分查找

示例

- 在右侧有序数组中寻找元素22
- 当前查找范围为　5~6

0	1	2	3	4	5	6	7	8	9
2	5	6	9	12	22	36	55	72	81

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

关键知识点：二分查找

示例

- 在右侧有序数组中寻找元素22
- 当前查找范围为　5~6
- 选取中间元素22与22进行对比，找到目标元素，返回下标5

0	1	2	3	4	5	6	7	8	9
2	5	6	9	12	22	36	55	72	81

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

关键知识点：二分查找


```
//二分查找算法，在a[start] ~ a[end]中查找key
public int binarySearch(int key, int a[], int start, int end) {
    if (start > end) //未找到key，返回-1
        return -1;

    int m = (start + end) / 2;
    if (a[m] == key) //找到key，返回key的id
        return m;

    if (a[m] > key)
        return binarySearch(key, a, start, m - 1);//在m左侧继续查找

    return binarySearch(key, a, m + 1, end);//在m右侧继续查找
}
```

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

二分查找nums1数组中的划分位置

- 首先选择nums1所有潜在划分位置的中间位置3

那么寻找中位数的问题就可以变为寻找满足下列条件的分界线：

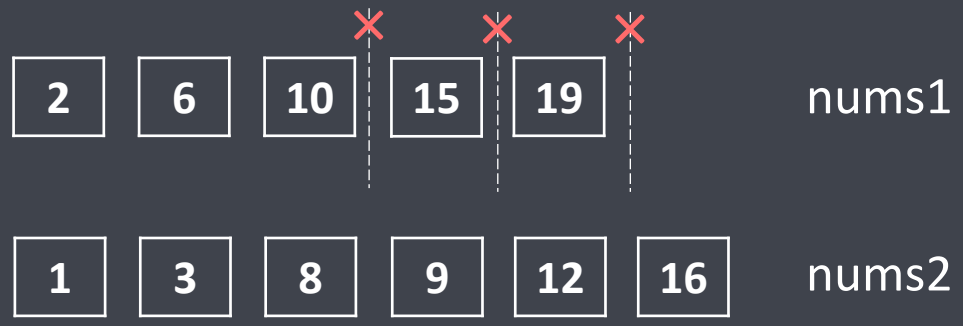
1. 分界线左边的元素都小于等于分界线右边的元素
2. 0 <=分界线左边的元素个数-分界线右边的元素个数<=1

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

二分查找nums1数组中的划分位置

- 排除3个潜在的位置
- 目标范围缩小到0~2



那么寻找中位数的问题就可以变为寻找满足下列条件的分界线：

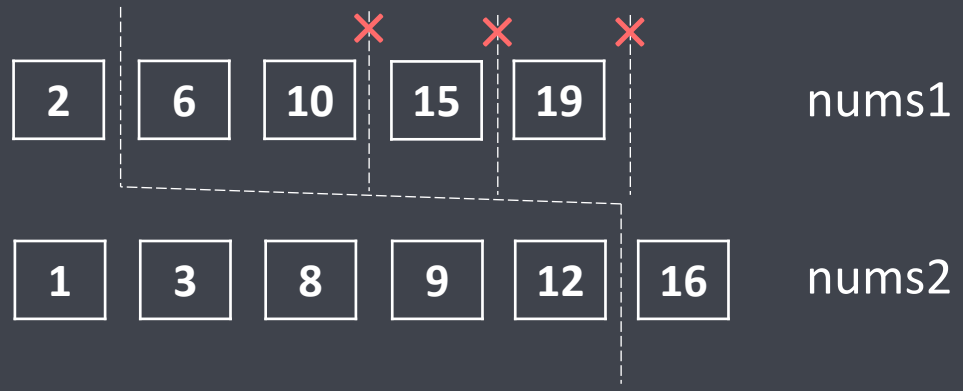
1. 分界线左边的元素都小于等于分界线右边的元素
2. 0 <=分界线左边的元素个数-分界线右边的元素个数<=1

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

二分查找nums1数组中的划分位置

- 选择nums1当前潜在划分位置的中间位置1



那么寻找中位数的问题就可以变为寻找满足下列条件的分界线：

1. 分界线左边的元素都小于等于分界线右边的元素
2. 0 <=分界线左边的元素个数-分界线右边的元素个数<=1

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

二分查找nums1数组中的划分位置

- 排除另外2个潜在的划分位置
- 潜在可行的划分位置只剩下2

那么寻找中位数的问题就可以变为寻找满足下列条件的分界线：

1. 分界线左边的元素都小于等于分界线右边的元素
2. 0 <=分界线左边的元素个数-分界线右边的元素个数<=1

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

二分查找nums1数组中的划分位置

- 经过测试划分位置2满足要求
- 由于|nums1|+|nums2|=11
- 中位数是第6小的数
- 即左边部分的最大值 max(6,9)=9

那么寻找中位数的问题就可以变为寻找满足下列条件的分界线：

1. 分界线左边的元素都小于等于分界线右边的元素
2. 0 <=分界线左边的元素个数-分界线右边的元素个数<=1

五. Code 最优解思路及编码实现

解题思路剖析

- 找到nums1和nums2中的分界线，将所有元素划分为两部分，使得:
  - 分界线左边的元素都小于等于分界线右边的元素
  - 0 <=分界线左边的元素个数-分界线右边的元素个数<=1
- 在nums1和nums2中元素较少的数组进行二分查找分界线
- 若|nums1|+|nums2|为奇数，返回分界线左侧元素的最大值
- 若|nums1|+|nums2|为偶数，返回分界线左侧元素最大值与右侧元素最小值的平均

五. Code 最优解思路及编码实现

复杂度分析

- 时间复杂度：
  - 只需要对nums1和nums2中较短数组进行二分查找
  - 二分查找的时间复杂度为O(log(min(m,n)))
- 空间复杂度：
  - 空间消耗方面我们只需要使用常数级的变量空间，因此空间复杂度为O(1)

五. Code 最优解思路及编码实现

拉勾教育
—互联网人实战大学—

Java编码实现

```java
public double findMedianSortedArrays(int[] nums1, int[] nums2) {
    int m = nums1.length, n = nums2.length;
    //较短的数组在前，确保m <= n
    if (m > n) return findMedianSortedArrays(nums2, nums1);
    //定义p、q为nums1分界线范围，共m+1个可能的划分位置
    int p = 0, q = m;
    int i=0,j=0;   //nums1、num2的分界位置
    while (p <= q) {
        i = (p + q) / 2; //二分确定nums1当前分界位置
        //根据i确定nums2的分界位置，使得左侧元素数-右侧元素数为0或1
        j = (m + n + 1) / 2 - i;
        //nums1右侧最小值小于nums2左侧最大值，nums1划分位置在[i+1, q]之间
        if (j != 0 && i != m && nums1[i] < nums2[j - 1])
            p = i + 1;
        else if (i != 0 && j != n && nums1[i - 1] > nums2[j])
            //nums1左侧最大值大于nums2右侧最小值，nums1划分位置在[p, i-1]之间
            q = i - 1;
        else {//当前划分位置左侧的最大值小于右侧的最小值，满足要求
            break;
        }
    }
    //m+n为奇数，返回左侧的最大值    左侧最大值：三种情况 nums1为空、nums2为空、都不为空
    int maxLeft = i==0?nums2[j - 1]:(j==0?nums1[i - 1]:Math.max(nums1[i - 1], nums2[j -
1]));
    if ((m + n) % 2 == 1) return maxLeft;
    //m+n为偶数，返回左侧最大值与右侧最小值的平均   右侧最小值：三种情况 nums1为空、nums2为空、都
不为空
    int minRight = i==m?nums2[j]:(j==n?nums1[i]:Math.min(nums1[i ], nums2[j]));
    return  (maxLeft + minRight) / 2.0;
}
```

执行结果：通过 显示详情 >

执行用时：2 ms，在所有 Java 提交中击败了 100.00% 的用户

内存消耗：40.9 MB，在所有 Java 提交中击败了 35.53% 的用户

六. Change 变形延伸

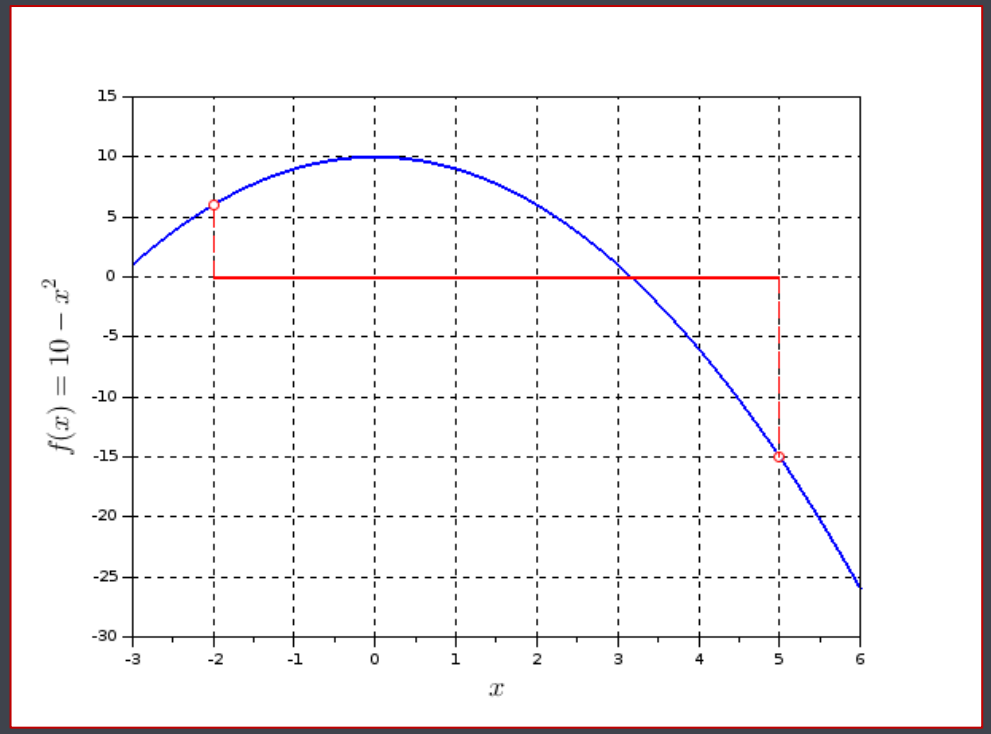
拉勾教育
—互联网人实战大学—

题目变形

1. 如果输入nums1和nums2变为逆序数组该如何实现？

延伸扩展

- 二分查找是一种基础高效的算法思维
- 实际工作中应用非常广泛
  - 数据库中使用最频繁的算法
  - 数学方程求根



总结

- 理解归并排序算法的思想与流程
- 掌握二分查找算法的原理与应用

课后练习

1. 二分查找（leetcode 704 /简单）
2. 查找二维矩阵（leetcode 74/中等）
3. 查找旋转排序数组（leetcode 33 /中等）
4. 查找旋转排序数组 II (leetcode 81 /中等)

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容