2019.11.17-2019.11.18 两数之和

1.C++：

向量（Vector）是一个封装了动态大小数组的顺序容器（Sequence Container）。跟任意其它类型容器一样，它能够存放各种类型的对象。可以简单的认为，向量是一个能够存放任意类型的动态数组。

参考博客: https://www.runoob.com/w3cnote/cpp-vector-container-analysis.html

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        int sum2=0;
        vector<int> a;
        for(int i=0;i<nums.size();i++)
        {
            for(int j=i+1;j<nums.size();j++)
            {
                sum2=nums[i]+nums[j];
                if(sum2==target)
                {
                    a.push_back(i);
                    a.push_back(j);

                    //cout<<"["<<i<<","<<j<<"]"<<endl;
                }
                else
                continue;
            }
        }
        return a;
    }
};
```

哈希表对应的容器是 unordered_map，哈希表是根据关键码值(key value)而直接进行访问的数据结构。它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度，这个映射函数叫做散列函数。

2.Java：

```
class Solution {
    public int[] twoSum(int[] nums, int target) {
        int result_list[] = new int[2];
        for(int i = 0;i < nums.length;i++)
        {
            for(int j = i+1;j < nums.length;j++)
            {
                if(nums[i]+nums[j]==target)
                {
                    //result_list = {i,j};
                    result_list[0] = i;
                    result_list[1] = j;

                }
            }
        }
        return result_list;
    }
}
```

java.lang.IllegalArgumentException

方法的参数错误
比如 g.setColor(int red,int green,int blue)这个方法中的三个值，如果有超过 2 5 5 的也会出现这个异常，因此一旦发现这个异常，我们要做的，就是赶紧去检查一下方法调用中的参数传递是不是出现了错误。

哈希表使用参考博客: https://blog.csdn.net/wxgxgp/article/details/79194360

HashMap 和 HashTable 到底哪不同？
https://juejin.im/post/5add97a46fb9a07aa212f4c0

利用 hashmap 实现，注意此处不是 key-value 而是巧妙的变换为 value-key，更加方便的调用函数

```
1.  class Solution {
2.      public int[] twoSum(int[] nums, int target) {
3.          HashMap<Integer, Integer> test = new HashMap();
4.               for(int i = 0;i < nums.length;i++){
5.                   test.put(nums[i],i);
6.                   }
7.              for(int j = 0;j < nums.length;j++){
8.                  int difference = target - nums[j];
9.                  if(test.containsKey(difference)&&test.get(difference)!=j){
10.                     return new int[] { j, test.get(difference)};
11.                 }
12.         }
13.         throw new IllegalArgumentException("No two sum solution");
14.     }
15. }
```

3.Python

```
1.  class Solution(object):
2.      def twoSum(self, nums, target):
3.          """
4.          :type nums: List[int]
5.          :type target: int
6.          :rtype: List[int]
7.          """
8.          list_array=[]
9.          for i in range(len(nums)):
10.             for j in range(i+1,len(nums)):
11.                 if(nums[i]+nums[j]==target):
12.                     list_array.append(i)
13.                     list_array.append(j)
14.                 else:
15.                     continue
16.         return list_array
```

python 中通过哈希表，此处是字典形式

```
1.  def twoSum(nums, target):
2.      hashmap={}
3.      for ind,num in enumerate(nums):
4.          hashmap[num] = ind
5.      for i,num in enumerate(nums):
6.          j = hashmap.get(target - num)
```

```
            if j is not None and i!=j:
                return [i,j]
```

enumerate() 函数用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。get 是返回 key 值的。
事例如：
seasons = ['Spring', 'Summer', 'Fall', 'Winter']
>>> list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]

2019.11.19-2019.11.20 链表实现两数相加
1.C++
基于 C++实现单向链表的基本功能，弄清楚相关的定义，如模板，类的定义，类中成员函数引用等，参考博客 https://www.cnblogs.com/QG-whz/p/5170147.html，进行总结。

未做出 我的出错代码：

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode* newNode;
        ListNode* newNode1 = newNode;
        int flag = 0;
        while(l1->next!=NULL&&l2->next!=NULL){
            if(l1!=NULL&&l2!=NULL){
                newNode->val = l1->val + l2->val+flag;
            }

            if (newNode->val>=10){
                flag = 1;
            }
            else
                flag = 0;

            if(l1!=NULL&&l2!=NULL){
                l1 = l1->next;
                l2 = l2->next;
                newNode = newNode->next;
            }

        }
        return newNode1;
    }
};
```

参考别人思路修改：
1.先比较长度不够补零处理
2.每位相加，设置进位位
3.输出

注意：模板中定义的函数用于链表初始化

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode* p = l1;
        ListNode* q = l2;
        int lenth1 = 1,lenth2 = 1;

        while(p->next!=NULL){
            lenth1++;
            p = p->next;
        }

        while(q->next!=NULL){
            lenth2++;
            q = q->next;
        }

        //if l2 is shorter,Zero padding
        if(lenth1>lenth2){
            for (int i = 0;i <lenth1-lenth2;i++){
                q->next = new ListNode(0);
                q = q->next;
            }
        }

        //if l1 is shorter,Zero padding
        if(lenth1<lenth2){
            for (int i = 0;i <lenth2-lenth1;i++){
                p->next = new ListNode(0);
                p = p->next;
            }
        }

        int flag = 0;
        ListNode* newNode = new ListNode(-1);
        ListNode* newNode1 = newNode;
        while(l1!=NULL&&l2!=NULL){
            int mid_value = 0;
            mid_value = l1->val + l2->val+flag;
            newNode1->next = new ListNode(mid_value%10);

            if (mid_value>=10){
                flag = 1;
            }
            else
                flag = 0;

            l1 = l1->next;
            l2 = l2->next;
            newNode1 = newNode1->next;

        }
        if(flag){
            newNode1->next = new ListNode(1);
            newNode1 = newNode1->next;
```

```
            }
            return newNode->next;
        }
    };
```

修改出来了，谈一谈收获吧，按照之前补零的思路进行，此处计算长度过程中，链表需要不断的往后移，如果直接用 L1 或者 L2 操作的话就会破坏原链表，因此此处我们定义了两个滑动链表指针 q 和 p，这一点类似于 IDS JAVA 代码中，我们获得一个 judgeline 之后往往生成一个迭代器，用 getInternet()函数。其次是考虑进位位时，很巧妙。注意链表的赋值操作，最后是还要考虑到最后一位如果需要进位的话，我们还需要在最后继续添加一个节点。

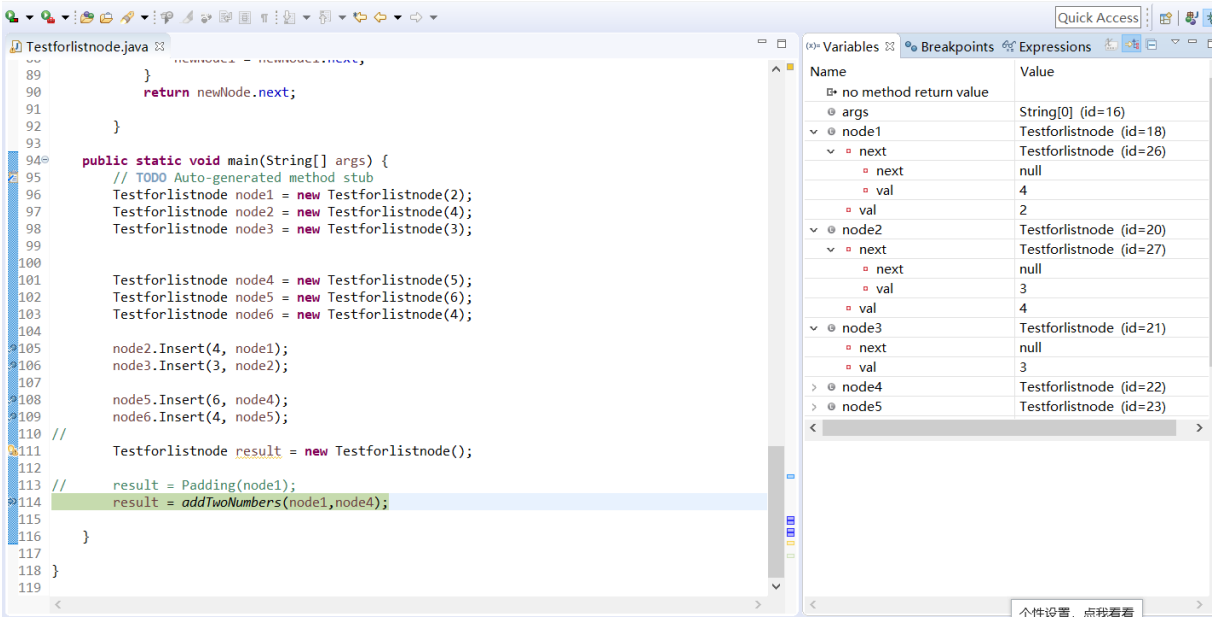
2.JAVA

存在错误

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode p,q;
        int length1 = 1,length2 = 1;

        p = l1;
        q = l2;

        //calculate the length of l1 and l2
        while(p.next!=null){
            length1++;
            p = p.next;
        }

        while(q.next!=null){
            length1++;
            q = q.next;
        }

        if (length1>length2){
            for(int i= 0;i<length1-length2;i++){
                q.next = new ListNode(0);
                q = q.next;
            }
        }
        else{
            for(int i= 0;i<length2-length1;i++){
                p.next = new ListNode(0);
                p = p.next;
            }
        }

        int flag = 0;
        ListNode newNode = new ListNode(-1);
        ListNode newNode1 = newNode;
        while(l1!=null&&l2!=null){
            int mid_value = 0;
            mid_value = l1.val + l2.val +flag;
            newNode1.next = new ListNode(mid_value%10);

```

```
                if(mid_value>=10){
                    flag = 1;
                }
                else{
                    flag = 0;
                }

                l1 = l1.next;
                l2 = l2.next;
                newNode1 = newNode1.next;

            }

            if(flag==1){
                newNode1.next = new ListNode(1);
                newNode1 = newNode1.next;
            }
            return newNode.next;

        }
    }
```

因为 Java 不存在指针，故如果套用上 C++中的先补零方法，其实是无法实现的，我们在 eclipse 中新建了工程用于测试参数的传递，如下图所示：



在图中我们看见 node1 与 node2 并没有连接起来，所以我们放弃了补零的思想，直接判断链表之后若为 null，则直接将其后的值用零代替，修改后的代码如下所示：

```
    class Solution {
        public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
            int flag = 0;
            // ListNode p,q;//if you want to protect l1&l2
            // p = l1;
            // q = l2;
            ListNode newNode = new ListNode(-1);
            ListNode newNode1 = newNode;

            while(l1!=null||l2!=null){
                int mid_value = 0;
                mid_value = ((l1!=null?l1.val:0)) + ((l2!=null?l2.val:0)) +fla
g;
                newNode1.next = new ListNode(mid_value%10);
```

```
            newNode1 = newNode1.next;

            if(mid_value>=10){
                flag = 1;
            }
            else{
                flag = 0;
            }

            if(l1!=null)l1 = l1.next;
            if(l2!=null)l2 = l2.next;

        }

        if(flag==1){
            newNode1.next = new ListNode(1);
            newNode1 = newNode1.next;
        }
        return newNode.next;

    }
}
```

3.Python

```
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def addTwoNumbers(self, l1, l2):
        """
        :type l1: ListNode
        :type l2: ListNode
        :rtype: ListNode
        """
        p,q,carry=l1,l2,0
        l3 = ListNode(-1)
        l4 = l3
        while l1 or l2:
            mid_value = (l1.val if l1 else 0)+(l2.val if l2 else 0)+carry

            l4.next = ListNode(mid_value%10)
            l4 = l4.next

            if mid_value>=10:
                carry = 1
            else:
                carry = 0

            if l1:
                l1 = l1.next
            if l2:
                l2 = l2.next

        if carry:
            l4.next = ListNode(1)
            l4 = l4.next;

        return l3.next
```

注意第一行貌似必须要一行写完？？？否则报错 int 型不能作为迭代器，还有就是 Python 在敲写过程中需要注意行的关系。

2019.11.22-2019.11.24 无重复字符的最长子串

1.C++

方法一暴力法，我的思路一开始是正确的，从头到尾遍历字符串，在类中定义一个函数进行判断是否子串中含有元素，但是依然有出入。以下是我的原始代码。

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        string l1 = s;
        //赋值到 char 型数组中
        // char l1[];
        // for (int i=0;i<s.size();i++){
        //     l1[i]=s[i];
        // }
        //put new element into container
        string con1="";
        string con2="";

        for (int i=0;i<l1.size();i++){
            if (!hasDup(con1,l1[i])||!hasDup(con2,l1[i])){
                con1 = con1+l1[i];
            }
            else{
                if (!hasDup(con2,l1[i])){
                    con2 = con2+l1[i];
                }
            }
        }

    }

    //judge the exsistance of the value in the char array
    bool hasDup(string A, char n){
        if(A.size()==0){
            return false;
        }
        for(int i = 0; i < A.size(); i++){
            if(A[i] == n){
                    return true;
            }
        }

        return false;
    }

};
```

暴力法思想较为简单，先枚举字符串中所有的字符串（通过枚举所有可能的开头 i 和结尾 j）；在定义函数判断字符串中是否有重复的字符，但是此法超过了时间限制。

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int n = s.size();
        int result = 0;

        for (int i=0;i<n;i++){
```

```
                for (int j=i+1;j<=n;j++){
                    if (IsFreshElement(s,i,j)){result = max(result, j-i);}
                }
            }
            return result;
        }

        //judge the exsistance of the value in the char array
        bool IsFreshElement(string A, int start,int end){
            unordered_map<char, int> hash;
            for (int i=start;i<end;i++){
                char character = A[i];
                int index =0;
                if(hash.count(character)!=0) return false;
                hash.insert(pair<char, int>(character,index));
                index++;
            }
            return true;
        }

    };
```

针对此法我们进行优化，此方法是基于滑动窗口，在 C++中使用

```
    class Solution {
    public:
        int lengthOfLongestSubstring(string s) {
            int n = s.size();
            int i = 0,j = 0;
            int result = 0,length = 0;

            unordered_map<char,int> hash;
            while(i<n&&j<n){
                char character = s[j];
                if (hash.find(character)!=hash.end()&&hash[character]>=i){
                    i = hash[character]+1;
                    length = j-i;
                }
                hash[character] = j;

                j++;
                length++;
                result = max(result,length);
            }
            return result;
        }
    };
```

C++中 unordered_map 中哈希表用法，可以参考博客：

因为不同语言中的用法有所差异，因此在 Java 和 Python 中我们对于滑动窗口的定义有所出入，比如上面代码中定义了两个滑动的指针 i 和 j，将 string 中的元素利用尾指针 j 进行指向，放入的元素不删除，我们判断的语句设置为这个 j 指向的新元素是否在区间[i,j)内注意是左开右闭。

2.Java

与 C++中 unordered_map 未重载<不同，在 Java 中我们采用的是不是将所有元素都放入哈希表中，此处我们不是判断是否新元素在区间内，而是不断地加入新元素，并判断新元素是否存在于表中，否则对其中的元素进行 remove，这些函数都是 Java 中的 set 中所含有的，我准备做一个关于不同语言中哈希表的实现实例。

```java
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int n = s.length();
        int i=0,j=0,result=0,length=0;

        Set<Character> set = new HashSet<>();
        while(i<n&&j<n){
            if(!set.contains(s.charAt(j))){
                set.add(s.charAt(j++));
                result = Math.max(result,j-i);
            }
            else{
                set.remove(s.charAt(i++));
            }
        }

        return result;
    }
}
```

3.Python

```python
class Solution(object):
    def lengthOfLongestSubstring(self, s):
        """
        :type s: str
        :rtype: int
        """

        n = len(s)
        lookupwindow = list()
        max_len,cur_len = 0,0

        for i in range(n):
            character = s[i]
            if not character in lookupwindow:
                lookupwindow.append(character)
                cur_len += 1

            else:
                index = lookupwindow.index(character)
                lookupwindow = lookupwindow[index+1:]
                lookupwindow.append(character)
                cur_len = len(lookupwindow)

            if cur_len>max_len:max_len = cur_len

        return max_len
```

Python 的语言表达更加灵活，此处我们定义一个 list 列表，不断地加入新的元素，如果查找到对应的 index 表示已经存在表中，我们将表格之前的元素予以删除，思路和 Java 相同，此处实现更为灵活。总结 Python 中字典，列表等数据结构的基本用法。

2019.11.27-2019.11.30 寻找两个有序数组的中位数

1.Java

在统计中，中位数被用来：将一个集合划分为两个长度相等的子集，其中一个子集中的元素总是大于另一个子集中的元素。可以参考山东大学《数学分析》中附录部分有关于中位数在有理数与无理数概念定义中的阐述。

在计算之前我们先明确几个概念，我们总是假设 A 数组长度为 m，而 B 数组长度为 n，且有 m<n。这样我们便可以将整个数组分为两个部分，一部分作为 leftPart，另一部分为

rightPart，如下所示：

leftPart	rightPart
A[0],A[1],... A[i-1]	A[i],A[i+1],...A[m - 1]
B[0],B[1],... B[j-1]	B[j],B[j+1],...B[n - 1]

由此我们可以可以将求解中位数的问题进行转换成如下的数学表达式：

$$median = (\max(leftPart) + \min(rightPart)) / 2$$

$$s.t: len(leftPart) = len(rightPart) \ \ \max(leftPart) <= \min(rightPart)$$

此处式中有很多不完善的地方，比如如果在 m+n 为奇数时，左右是不等的，但是在计算机中进行除法操作是自动求解最小整数的，这就在编程时方便了我们表达，由此我们推断出 i 和 j 的关系：$i + j = m - i + n - j / m - i + n - j + 1$

事实上在编程中加一得到的结果在奇数和偶数时得到的结果是一致的，所以我们采用加一的式子，得到两者之间的关系：$j = (m + n + 1) / 2$，其中 $i \in [0, m]$，当然这个式子是建立在 $m < n$ 的基础之上的，在代码中你会看到我们一开始就非常巧妙的设置了这个条件。它的具体证明过程如下：

感谢 @Quentin.chen 指出：$i < m \implies j > 0$ 以及 $i > 0 \implies j < n$ 始终成立，这是因为：

$$m \le n,\ i < m \implies j = \frac{m+n+1}{2} - i > \frac{m+n+1}{2} - m \ge \frac{2m+1}{2} - m \ge 0$$

$$m \le n,\ i > 0 \implies j = \frac{m+n+1}{2} - i < \frac{m+n+1}{2} \le \frac{2n+1}{2} \le n$$

而针对 max(*leftPart*) <= min(*rightPart*) 我们可以将其转换为一个交叉的形式，如下所示：

$$\begin{cases} A[i-1] <= B[j] \\ A[i] >= B[j-1] \end{cases}$$

此处我们做好了所有的准备工作，我们开始进行算法的整个过程展示：

输入：数组 A 长度为 m，数组 B 长度 n

过程：

1.初始化 $iMin = 0$ 和 $iMax = m$，其中 i 在 $[iMin, iMax]$ 中搜索

2.定义 $i = (iMin + iMax) / 2$ 和 $j = (m + n + 1) / 2 - i$

3. *while* $iMin <= iMax$

    if $B[j-1] > A[i]$ && $i < iMax$:增大 i，及 $iMin = i + 1$

    else if $A[i-1] > B[j]$ && $i > iMin$:减小 i，及 $iMax = i - 1$

    else ://满足条件，根据边界条件及奇偶数进行输出

        if $i == 0$: $MaxLeft = B[j-1]$ //A 左边为空

        else if $j == 0$: $MaxLeft = A[i-1]$ //B 左边为空

        else : $MaxLeft = \max(A[i-1], B[j-1])$

        if $m + n$ *is odd* : *return MaxLeft*

        if $i == m$: $MinRight = B[j]$ //A 右边为空

        else if $j == n$: $MinRight = A[i]$ //B 右边为空

        else : $MinRight = \min(A[i], B[j])$

        return $(MaxLeft + MinRight)/2.0$

    end

4. *return* 0.0

代码如下：

```
1.    class Solution {
2.       public static double findMedianSortedArrays(int[] nums1, int[] nums2) {

3.            int m = nums1.length;
4.            int n = nums2.length;
5.
6.            //ensure m<=n
7.            if(m>n){
8.                // return findMedianSortedArrays(nums2,nums1);
9.                int[] temp = nums1; nums1 = nums2; nums2 = temp;
10.               int tmp = m; m = n; n = tmp;
11.           }
12.
13.           int iMin = 0,iMax = m,halfLen = (m+n+1)/2;
14.           while(iMin<=iMax){
15.               int i =(iMin+iMax)/2;
16.               int j =halfLen - i;
17.               if(i<iMax&&nums2[j-1]>nums1[i]){
18.                   iMin = i+1;// i is too small
19.               }
20.               else if(i>iMin&&nums1[i-1]>nums2[j]){
21.                   iMax = i-1;// i is too big
22.               }
23.               else{
24.                   int maxLeft = 0;
25.                   if(i==0){
26.                       maxLeft = nums2[j-1];
27.                   }
28.                   else if(j==0){
29.                       maxLeft = nums1[i-1];
30.                   }
31.                   else{
32.                       maxLeft = Math.max(nums1[i-1],nums2[j-1]);
33.                   }
34.                   if((m+n)%2==1){
35.                       return maxLeft;//奇数个中位数为 maxLeft
36.                   }
37.
38.                   int minRight = 0;
39.                   if(i==m){
40.                       minRight = nums2[j];
41.                   }
42.                   else if(j==n){
43.                       minRight = nums1[i];
44.                   }
45.                   else{
46.                       minRight = Math.min(nums2[j],nums1[i]);
47.                   }
48.                   return (maxLeft+minRight)/2.0;
49.               }
50.           }
51.           return 0.0;
52.       }
53.   }
```

2.C++

C++中我们采用了虚拟加#的方法，使得 $m \to 2m+1$ 和 $n \to 2n+1$，如下所示

之前	len	之后	len
[1 4 7 9]	4	[# 1 # 4 # 7 # 9 #]	9
[2 3 5]	3	[# 2 # 3 # 5 #]	7

则 $LMaxi = (i-1)/2$ 和 $RMini = i/2$，注意此时 $i \in [0, 2m]$，其代码如下：

```
#define max(a,b) ((a) > (b) ? (a) : (b))
#define min(a,b) ((a) < (b) ? (a) : (b))
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2)
{
        int m = nums1.size();
        int n = nums2.size();

        if(m>n){
            return findMedianSortedArrays(nums2,nums1);
        }

        int LMax1,LMax2,RMin1,RMin2,i,j;
        int iMin = 0,iMax = 2*m;
        while(iMin<=iMax){
            i = (iMax+iMin)/2;
            j = m+n-i;

            LMax1 = (i==0)?INT_MIN:nums1[(i-1)/2];
            RMin1 = (i==2*m)?INT_MAX:nums1[i/2];
            LMax2 = (j==0)?INT_MIN:nums2[(j-1)/2];
            RMin2 = (j==2*n)?INT_MAX:nums2[j/2];

            if(LMax1>RMin2)
                iMax = i - 1;
            else if(LMax2>RMin1)
                iMin = i + 1;
            else
                break;
        }
        return (max(LMax1,LMax2)+min(RMin1,RMin2))/2.0;
    }
};
```

3.Python

注意一点就是 Python 中交换赋值很简单

```
class Solution(object):
    def findMedianSortedArrays(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: float
        """
        m,n = len(nums1),len(nums2)
        if m>n:
            m,n,nums1,nums2 = n,m,nums2,nums1

        if n==0:
            raise ValueError

        iMin,iMax,halfLen = 0,m,(m+n+1)/2
        while iMin<=iMax:
            i = (iMin+iMax)/2
            j = halfLen - i

            if i<m and nums2[j-1]>nums1[i]:
                iMin = i + 1
            elif i>0 and nums1[i-1]>nums2[j]:
```

```
                iMax = i - 1
            else:
                if i==0:maxleft = nums2[j-1];
                elif j==0:maxleft = nums1[i-1]
                else:maxleft=max(nums1[i-1],nums2[j-1])
                if (m+n)%2==1:
                    return maxleft

                if i==m:minright = nums2[j]
                elif j==n:minright = nums1[i]
                else:minright=min(nums1[i],nums2[j])

                return (maxleft + minright)/2.0
```