/\*

There are two sorted arrays nums1 and nums2 of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

Example 1:

nums1 = [1, 3]

nums2 = [2]

The median is 2.0

Example 2:

nums1 = [1, 2]

nums2 = [3, 4]

The median is (2 + 3)/2 = 2.5

方法一：归并排序 O(m+n)

方法二：

该方法的核心是将原问题转变成一个寻找第k小数的问题（假设两个原序列升序排列），

这样中位数实际上是第(m+n)/2小的数。所以只要解决了第k小数的问题，原问题也得以解决。

首先假设数组A和B的元素个数都大于k/2，我们比较A[k/2-1]和B[k/2-1]两个元素，这两个元素分别表示A的第k/2小的元素和B的第k/2小的元素。这两个元素比较共有三种情况：>、<和=。如果A[k/2-1] < B[k/2-1]，这表示A[0]到A[k/2-1]的元素都在A和B合并之后的前k小的元素中。换句话说，A[k/2-1]不可能大于两数组合并之后的第k小值，所以我们可以将其抛弃。

当A[k/2-1]>B[k/2-1]时存在类似的结论。

当A[k/2-1]=B[k/2-1]时，我们已经找到了第k小的数，也即这个相等的元素，我们将其记为m。由于在A和B中分别有k/2-1个元素小于m，所以m即是第k小的数。(这里可能有人会有疑问，如果k为奇数，则m不是中位数。这里是进行了理想化考虑，在实际代码中略有不同，是先求k/2，然后利用k-k/2获得另一个数。)

通过上面的分析，我们即可以采用递归的方式实现寻找第k小的数。此外我们还需要考虑几个边界条件：

如果A或者B为空，则直接返回B[k-1]或者A[k-1]；

如果k为1，我们只需要返回A[0]和B[0]中的较小值；

如果A[k/2-1]=B[k/2-1]，返回其中一个；

\*/

class Solution {

public:

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2)

{

//way-1

/\*

int l1=nums1.size();

int l2=nums2.size();

int k1=0;

int k2=0;

vector<int> contain;

while(k1<l1 && k2<l2)

{

if(nums1[k1]<nums2[k2])

{

contain.push\_back(nums1[k1]);

k1++;

}

else

{

contain.push\_back(nums2[k2]);

k2++;

}

}

if(k1==l1)

{

for(int i=k2;i<l2;i++)

contain.push\_back(nums2[i]);

}

if(k2==l2)

{

for(int i=k1;i<l1;i++)

contain.push\_back(nums1[i]);

}

int k=contain.size();

for(int i=0;i<k;i++)

cout<<contain[i]<<" ";

cout<<endl;

if(k%2==1)

return contain[(k-1)/2];

else

return (double(contain[k/2]+contain[k/2-1])/2);

\*/

//way-2

//未实现……边界条件比较麻烦,下面是网上标准代码

/\*

double findKthSortedArrays(int A[], int m, int B[], int n, int k)

{//k starts from 1

//make sure A is shorter than B

if(m > n)

return findKthSortedArrays(B, n, A, m, k);

//special case1: A empty

if(m == 0)

return B[k-1];

//special case2: k==1 (m>0 is guaranteed)

if(k == 1)

return min(A[0], B[0]);

int Acandi, Bcandi;

if((k-1)/2 >= m)

{//A[(k-1)/2] out of range

Acandi = A[m-1];

Bcandi = B[k-m-1];

if(Acandi == Bcandi)

return Acandi;

else if(Acandi > Bcandi)

//for A: no skip

//for B: skip the k-m smaller elements (including Bcandi)

return findKthSortedArrays(A, m, B+k-m, n-(k-m), k-(k-m));

else

//for A: skip the m smaller elements

//for B: skip the k-m larger elements

return findKthSortedArrays(A+m, 0, B, n-(k-m), k-m);

}

else

{

//1,2->index0; 3,4->index1; ...

Acandi = A[(k-1)/2];

Bcandi = B[(k-1)/2];

if(Acandi == Bcandi)

return Acandi;

else if(Acandi > Bcandi)

{

//for A: skip the larger elements

//for B: skip the smaller elements

if(k%2 == 1)

//keep the smaller candidate, skip the larger

return findKthSortedArrays(A, (k-1)/2, B+(k-1)/2, n-(k-1)/2, k-(k-1)/2);

else

//keep the larger candidate, skip the smaller

return findKthSortedArrays(A, (k-1)/2+1, B+(k-1)/2+1, n-((k-1)/2+1), k-((k-1)/2+1));

}

else

{

//for A: skip the smaller elements

//for B: skip the larger elements

if(k%2 == 1)

//keep the smaller candidate, skip the larger

return findKthSortedArrays(A+(k-1)/2, m-(k-1)/2, B, (k-1)/2, k-(k-1)/2);

else

//keep the larger candidate, skip the smaller

return findKthSortedArrays(A+(k-1)/2+1, m-((k-1)/2+1), B, (k-1)/2+1, k-((k-1)/2+1));

}

}

}

double findMedianSortedArrays(int A[], int m, int B[], int n)

{

if((m+n)%2 == 0)

{//average of two medians

return (findKthSortedArrays(A, m, B, n, (m+n)/2) + findKthSortedArrays(A, m, B, n, (m+n)/2+1))/2;

}

else

{

return findKthSortedArrays(A, m, B, n, (m+n)/2+1);

}

}

\*/

};