**①题目描述:假设两个数组的长度分别为m和n，两个数组都是有序递增的.求这两个数组的第k大的数.**

**代码:**

int FindKthElm(int A[], int aBeg, int aEnd, int B[], int bBeg, int bEnd, int k){

if (aBeg > aEnd)return B[bBeg + k - 1];

if (bBeg > bEnd)return A[aBeg + k - 1];

int aMid = aBeg + (aEnd - aBeg) / 2;

int bMid = bBeg + (bEnd - bBeg) / 2; **//包括了76和85这两个数**

int halfLen = aMid - aBeg + bMid - bBeg + 2;

if (A[aMid] < B[bMid]){

if (halfLen > k)return FindKthElm(A, aBeg, aEnd, B, bBeg, bMid - 1, k);

else return FindKthElm(A, **aMid + 1**, aEnd, B, bMid, bEnd, k - (aMid - aBeg + 1+bMid-bBeg));

}

else{

if (halfLen > k)return FindKthElm(A, aBeg, aMid - 1, B, bBeg, bEnd, k);

else return FindKthElm(A, **aMid**, aEnd, B, bMid + 1, bEnd, k - (bMid - bBeg + 1+aMid-aBeg));

}

}

分析: 假设有两个数组，如下表所示，

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Array a | 3 | 21 | 34 | **56** | 69 | 78 | 92 |  |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Array b | 5 | 32 | 45 | **76** | **85** | 96 | 126 | 142 |

A、数组a和数组b的中位数分别为56和80.5，首先我们比较这两个数组的中位数，然后统计在两个中位数之前的元素的总位数halflen，如果要寻找的第k个数小于halflen则说明第k个数和**中位数隐形比较之后**存在于中位数的前面，那么此时我们可以缩小范围(将中位数较大的数组的mid以后的元素cut掉)，若第k个数大于halflen则说明第k个数在中位数之后，那么按照中位数的寻找方式将范围缩小.

B、如果存在halflen==k的情况，那么也是具有较大中位数的数组的mid位开始(即bMid-bEnd)，较小的中位数数组从(amid+1)开始,因为bMid>aMid,而两者的[0,Mid]之间的数又刚好等于halflen，所以也只有可能是bMid恰好是第k个数.

C、如果存在A[aMid]==B[bMid]的情况，则寻找第k个数和**A[aMid]==B[bMid]无直接联系，只和数组的前mid部分加起来等不等于halflen有关.**

**②问题延伸:找出两个排序数组的中位数.**

My solution:

class Solution {

public:

double find\_median\_num(vector<int>& nums1,int aBeg, int aEnd,vector<int>nums2,int bBeg,int bEnd){

int alen = aEnd - aBeg + 1;

int blen = bEnd - bBeg + 1;

//nums1数组只有1个数(nums1数组元素始终比较少)

if (alen==1){

if(blen==1)return(double)(nums1[aEnd]+nums2[bEnd])/2; //两个数组中同时都只有一个元素

else if(blen%2==0){ //nums2此时有偶数个数

if (nums2[bBeg+blen/2]>=nums1[aEnd]&&nums1[aEnd]>=nums2[bBeg+blen/2-1]){

return nums1[aEnd];

}

else if(bmid>nums1[aEnd])return nums2[bBeg+blen/2-1];

else return nums2[bBeg+blen/2];

}

else{ //nums2数组中有基数个数

if (nums2[bBeg+(blen-1)/2+1]>=nums1[aEnd]&&nums1[aEnd]>=nums2[bBeg+(blen-1)/2-1]){

return (double)(nums2[bBeg+(blen-1)/2]+nums1[aEnd])/2;

}

else{

if (nums1[aEnd]>nums2[bBeg+(blen-1)/2+1]) {

return (double)(nums2[bBeg+(blen-1)/2]+nums2[bBeg+(blen-1)/2+1])/2;

}

else return (double)(nums2[bBeg+(blen-1)/2]+nums2[bBeg+(blen-1)/2-1])/2;

}

}

}

//求中位数

double amid, bmid;

if (alen % 2 == 0)amid =(double)(nums1[aBeg+alen/2-1]+nums1[aBeg+alen/2])/2;

else amid = nums1[aBeg + (alen - 1) / 2];

if (blen % 2 == 0)bmid =(double)(nums2[bBeg+blen/2]+nums2[bBeg+blen/2-1])/2;

else bmid = nums2[bBeg+(blen - 1)/2];

//两个数组中的元素个数都为2

if (alen%2==0&&blen%2==0){

if (nums2[bBeg+blen/2]<=nums1[aBeg+alen/2]&&nums1[aBeg+alen/2-1]<=nums2[bBeg+blen/2-1]) return bmid;

if (nums1[aBeg+alen/2]<=nums2[bBeg+blen/2]&&nums2[bBeg+blen/2-1]<=nums1[aBeg+alen/2-1]) return amid;

}

//判断

if (amid == bmid)return amid;

else if (amid > bmid){ //a的中位数比b的中位数大

//alen为偶数,则要删去alen/2个元素

if (alen%2==0)return find\_median\_num(nums1, aBeg, aEnd-alen/2,nums2, bBeg+alen/2,bEnd);

//alen为基数,则要删去(alen+1)/2或是(alen-1)/2+1个元素

else return find\_median\_num (nums1, aBeg, aEnd-(alen+1)/2+1,nums2, bBeg+(alen+1)/2-1, bEnd);

}

else { //amid < bmid

//alen为基数

if (alen % 2 == 0)return find\_median\_num(nums1, aBeg+alen/2, aEnd, nums2, bBeg, bEnd-alen/2);

//alen为基数

else return find\_median\_num(nums1, aBeg+(alen+1)/2-1, aEnd, nums2, bBeg, bEnd-(alen+1)/2+1);

}

}

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

int size1 = nums1.size();

int size2 = nums2.size();

if (size1 == 0 && size2 == 0)return 0;

else if (size1 == 0 && size2 != 0) {

if (size2 % 2 == 0)return (double)(nums2[size2/2] + nums2[size2/2-1]) / 2;

else return nums2[(size2 - 1) / 2];

}

else if (size1 != 0 && size2 == 0) {

if (size1 % 2 == 0)return (double)(nums1[size1/2] + nums1[size1/2-1]) / 2;

else return nums1[(size1-1)/2];

}

else {

//始终将元素小的放在前面

If (size2>size1) return find\_median\_num(nums1, 0, size1 - 1, nums2, 0, size2 - 1);

else return find\_median\_num(nums2, 0, size2 - 1, nums1, 0, size1 - 1);

}

}

};

分析:

1. 在单个数组的情况下，数组的中位数和数组的元素个数有关，若元素个数为基数，则返回数组中间的那个元素,若元素个数为偶数，则返回最中间两个元素的平均值。
2. 求两个排序数组的中位数不仅满足以上条件外,还存在以下情况：若两个数组的元素个数加起来为基数则说明中位数要么在array a中要么在array b中；若总元素个数为偶数，则中位数为两个元素A和B的平均值，可能情况有:

1、A和B分别在array a和array b中；

2、A和B可能都在array a中；

3、A和B可能都在array b中,比如说{1, 2, 6, 7},{3, **4**, **5**, 8},中位数在4和5之间.

**示例代码:**

class Solution {

public:

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

if (nums1.empty() && nums2.empty()) return 0;

if (nums1.size()>nums2.size()){

swap(nums1, nums2);

}

return findMedian(nums1, 0, nums1.size(), nums2, 0, nums2.size());

}

double findMedian(vector<int>& nums1, int left1, int right1, vector<int>& nums2, int left2, int right2){

while (true){

if (right1 - left1 <= 4){

int size1 = right1 - left1;

int size2 = right2 - left2;

int diff = max(0, (size2 - size1) / 2 - 1);

vector<int> v(nums2.begin() + left2 + diff, nums2.begin() + right2 - diff);

for (int i = left1; i<right1; ++i){

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

}

sort(v.begin(), v.end());

int size = v.size();

return (v[size / 2] + v[(size - 1) / 2]) / 2.0;

}

else{

int mid1 = (left1 + right1) / 2;

int mid2 = (left2 + right2) / 2;

if (nums1[mid1]<nums2[mid2]){//remove left of nums1 and right of nums2

**int diff = min(mid1 - left1, right2 - mid2 - 1);**

**left1 += diff;**

**right2 -= diff;**

}

else{

int diff = min(right1 - mid1 - 1, mid2 - left2);

right1 -= diff;

left2 += diff;

}

}

}

return 0;

}

};

**分析:** 在每次进行cut的时候，并没有将偶数情况下的中间两个数切开, {1, 2, 6, 7},{3, **4**, **5**, 8},即切割指针永远在4和5的左边或者是右边(始终将4和5都包含进去).