```
/*
给定两个大小为 m 和 n 的有序数组 nums1 和 nums2。

请你找出这两个有序数组的中位数，并且要求算法的时间复杂度为 O(log(m + n))。

你可以假设 nums1 和 nums2 不会同时为空。

示例 1:

nums1 = [1, 3]
nums2 = [2]

则中位数是 2.0

示例 2:

nums1 = [1, 2]
nums2 = [3, 4]

则中位数是 (2 + 3)/2 = 2.5

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/median-of-two-sorted-arrays
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:

- 方法一:根据中位数的定义,可以根据(m+n)的奇偶性来对两个有序数组(默认升序)进行归并,归并到一半的位置即可停止,但时间复杂度为$O(m+n)$,虽然提交通过,但实际不符合题目要求.(空间复杂度为$O(1)$
- 方法二:致谢二分递归法

  设两个数组的总长度为$N>0$,若N是奇数,则只需要找到第$\frac{N+1}{2}$小的数直接返回即可,若N为偶数,则需要找到第$\frac{N}{2}$小和第$\frac{N}{2}+1$小的数取平均后返回，所以问题可以简化为在两个有序(默认升序)数组中如何找到第k小的数
  - 假设两个有序数组的长度分别为4和10,要找第7小的数字;
  - 数组A:$1,3,4,9$
  - 数组B:$1,2,3,4,5,6,7,8,9,10$
  - 我们比较两个数组的$\frac{k}{2}$个数,若k是奇数,向下取整,也就是第3个数,对应数组1中的4和对应数组2中的3,若哪个数小,就表明对应数组的前$\frac{k}{2}$个数字都不是第k小的数,直接可以排除.
  - 在数组A:$1,3,4,9$和新的数组B:$4,5,6,7,8,9,10$作为新的数组中取找第$7-3=4$小的数.
  - 如此递归，直到某个数组的长度为0或者要找两个数组第1小的数即可结束递归.

方法一:C_归并

```
double findMedianSortedArrays(int* nums1, int nums1Size, int* nums2, int
nums2Size)
{
    int type        = (nums1Size % 2 + nums2Size % 2) % 2              ;
```

```
    int loc1        = (nums1Size + nums2Size)/2 -1                      ;
/*这里依然可能会有溢出*/
    int loc2        = loc1+1                                            ;
    int count       = 0                                                 ;
    int i           = 0                                                 ;
    int j           = 0                                                 ;
    double temp0    = 0.0                                               ;
    double temp1    = 0.0                                               ;
    double temp2    = 0.0                                               ;

    if(nums1Size==0 && nums2Size==0)
    {
        return 0.0;
    }
    else if(nums2Size ==0)
    {
        if(nums1Size%2==0)
        {
            return (double)(nums1[nums1Size/2-1]+nums1[nums1Size/2])/2.0;
        }
        else
        {
            return (double)(nums1[nums1Size/2]);
        }

    }
    else if (nums1Size ==0)
    {
        if(nums2Size%2==0)
        {
            return (double)(nums2[nums2Size/2-1]+nums2[nums2Size/2])/2.0;
        }
        else
        {
            return (double)(nums2[nums2Size/2]);
        }
    }
    else
    {
        while (1)
        {
            if(  (i<nums1Size)
               &&(j<nums2Size)
            )
            {
                if(nums1[i]<=nums2[j])
                {
                    temp0 = nums1[i];
                    i++;

                }
                else
                {
                    temp0 = nums2[j];
                    j++;

                }

```

```
            }
            else if(i<nums1Size)
            {
                temp0 = nums1[i];
                i++;

            }
            else if(j<nums2Size)
            {
                temp0 = nums2[j];
                j++;

            }
            else
            {
                break;
            }

            if(count==loc1)
            {
                temp1 = temp0;
            }
            if(count==loc2)
            {
                temp2 = temp0;
                break;
            }
            count++;
        }
        if(type == 1)
        {
            return temp2;
        }
        else
        {
            return (temp1+temp2)/2;
        }
    }
}

/*
执行用时 :22~44 ms
内存消耗 :7.6~7.7MB
*/
double findMedianSortedArrays(  int*    nums1       ,
                                int     nums1Size   ,
                                int*    nums2       ,
                                int     nums2Size
                             )
{

    int     temp        =   nums1Size   ;
    int*    nums        =   NULL        ;
    int     type        =   0           ;
    int     loc1        =   0           ;
    int     loc2        =   0           ;
    int     count       =   0           ;
    int     i           =   0           ;
```

```
    int     j           =   0           ;
    double  temp0       =   0.0         ;
    double  temp1       =   0.0         ;
    double  temp2       =   0.0         ;


    if(nums1Size<nums2Size)
    {
        temp        =   nums1Size   ;
        nums1Size   =   nums2Size   ;
        nums2Size   =   temp        ;
        nums        =   nums1       ;
        nums1       =   nums2       ;
        nums2       =   nums        ;
    }

    type = (nums1Size % 2 + nums2Size % 2) % 2;
    loc1 = (nums1Size - nums2Size)/2 + nums2Size -1 ;
    loc2 = loc1+1;

    if(  (nums1Size==0)
       &&(nums2Size==0)
    )
    {
        return 0.0;
    }
    else if(nums2Size ==0)
    {
        if(nums1Size%2==0)
        {
            return (double)(nums1[nums1Size/2-1]+nums1[nums1Size/2])/2.0;
        }
        else
        {
            return (double)(nums1[nums1Size/2]);
        }

    }
    else if (nums1Size ==0)
    {
        if(nums2Size%2==0)
        {
            return (double)(nums2[nums2Size/2-1]+nums2[nums2Size/2])/2.0;
        }
        else
        {
            return (double)(nums2[nums2Size/2]);
        }
    }
    else
    {
        while (1)
        {
            if(  (i<nums1Size)
               &&(j<nums2Size)
            )
            {
                if(nums1[i]<=nums2[j])
```

```
                {
                    temp0 = nums1[i];
                    i++;

                }
                else
                {
                    temp0 = nums2[j];
                    j++;

                }

            }
            else if(i<nums1Size)
            {
                temp0 = nums1[i];
                i++;

            }
            else if(j<nums2Size)
            {
                temp0 = nums2[j];
                j++;

            }
            else
            {
                break;
            }

            if(count==loc1)
            {
                temp1 = temp0;
            }
            if(count==loc2)
            {
                temp2 = temp0;
                break;
            }
            count++;
        }
        if(type == 1)
        {
            return temp2;
        }
        else
        {
            return (temp1+temp2)/2;
        }
    }
}
/*
执行结果:
通过
显示详情
执行用时 :24 ms, 在所有 C 提交中击败了63.63% 的用户
内存消耗 :7.6 MB, 在所有 C 提交中击败了87.35%的用户
*/
```

方法二:C_递归二分.

```
int getKth  (   int*    nums1   ,
                int     start1  ,
                int     end1    ,
                int*    nums2   ,
                int     start2  ,
                int     end2    ,
                int     k           /*要找第k小的数(k=1,2,3,...)*/
            )
{
    int len1    =   end1 - start1 + 1   ;
    int len2    =   end2 - start2 + 1   ;
    int i       =   start1              ;
    int j       =   start2              ;

    if(len1 > len2) /*保证len1<=len2*/
    {
        return getKth(nums2,start2,end2,nums1,start1,end1,k);
    }
    /* 递归结束条件1,某个数组已经全是比第k小的数都小了或者这个数组的长度本来就是0*/
    if((len1==0)&&(len2==0))
    {
        return 0;/*若一开始输入的就是两个长度为0的数组,直接返回*/
    }
    else if(len1 == 0)
    {
        return nums2[start2+k-1];
    }
    /* 递归结束条件2,找到最小点数据(第1小)*/
    if(k==1)
    {
        return nums1[start1] <= nums2[start2] ? nums1[start1] :
nums2[start2];
    }

    /*取要求长度和现有长度的较小值作为比较索引,防止访问越界*/
    i += (len1 <= k/2 ? len1 : k/2 ) - 1;
    j += (len2 <= k/2 ? len2 : k/2 ) - 1;

    if(nums1[i]>nums2[j])
    {
        return getKth(nums1,start1,end1,nums2,j+1,end2,k-(j-start2+1));
    }
    else
    {
        return getKth(nums1,i+1,end1,nums2,start2,end2,k-(i-start1+1));
    }
}

double findMedianSortedArrays(  int*        nums1       ,
                                int         nums1Size   ,
                                int*        nums2       ,
                                int         nums2Size
```

```
                            )
{
    unsigned int left    =   (nums1Size + nums2Size+1 )/2       ;  /*用 unsigned int 做防溢出*/
    unsigned int right   =   (nums1Size + nums2Size   )/2 + 1    ;

    if(left!=right)
    {
        return  0.5*(getKth(nums1,0,nums1Size-1,nums2,0,nums2Size-1,left) + getKth(nums1,0,nums1Size-1,nums2,0,nums2Size-1,right) );
    }
    else
    {
        return getKth(nums1,0,nums1Size-1,nums2,0,nums2Size-1,left);
    }
}

/*
执行结果:
通过
显示详情
执行用时 :16 ms, 在所有 C 提交中击败了95.77% 的用户
内存消耗 :7.4 MB, 在所有 C 提交中击败了95.45%的用户
*/
```

AlimyBreak
2019.08.25