

```

1  /*
2  给定两个大小为 m 和 n 的有序数组 nums1 和 nums2。
3
4  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  $O(\log(m + n))$ 。
5
6  你可以假设 nums1 和 nums2 不会同时为空。
7
8  示例 1:
9
10 nums1 = [1, 3]
11 nums2 = [2]
12
13 则中位数是 2.0
14
15 示例 2:
16
17 nums1 = [1, 2]
18 nums2 = [3, 4]
19
20 则中位数是  $(2 + 3)/2 = 2.5$ 
21
22 来源：力扣 (LeetCode)
23 链接：https://leetcode-cn.com/problems/median-of-two-sorted-arrays
24 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。
25 */

```

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

```

1  double findMedianSortedArrays(int* nums1, int nums1Size, int* nums2, int
2  nums2Size)
3  {
4      int type          = (nums1Size % 2 + nums2Size % 2) % 2 ;

```

```

4     int loc1      = (nums1Size + nums2Size)/2 -1                                ;
    /*这里依然可能会有溢出*/
5     int loc2      = loc1+1                                                    ;
6     int count     = 0                                                         ;
7     int i         = 0                                                         ;
8     int j         = 0                                                         ;
9     double temp0   = 0.0                                                      ;
10    double temp1   = 0.0                                                      ;
11    double temp2   = 0.0                                                      ;
12
13    if(nums1Size==0 && nums2Size==0)
14    {
15        return 0.0;
16    }
17    else if(nums2Size ==0)
18    {
19        if(nums1Size%2==0)
20        {
21            return (double)(nums1[nums1Size/2-1]+nums1[nums1Size/2])/2.0;
22        }
23        else
24        {
25            return (double)(nums1[nums1Size/2]);
26        }
27    }
28
29    else if (nums1Size ==0)
30    {
31        if(nums2Size%2==0)
32        {
33            return (double)(nums2[nums2Size/2-1]+nums2[nums2Size/2])/2.0;
34        }
35        else
36        {
37            return (double)(nums2[nums2Size/2]);
38        }
39    }
40    else
41    {
42        while (1)
43        {
44            if( (i<nums1Size)
45                &&(j<nums2Size)
46            )
47            {
48                if(nums1[i]<=nums2[j])
49                {
50                    temp0 = nums1[i];
51                    i++;
52                }
53                else
54                {
55                    temp0 = nums2[j];
56                    j++;
57                }
58            }
59        }
60

```

```

61         }
62         else if(i<nums1Size)
63         {
64             temp0 = nums1[i];
65             i++;
66         }
67         else if(j<nums2Size)
68         {
69             temp0 = nums2[j];
70             j++;
71         }
72         else
73         {
74             break;
75         }
76
77         if(count==loc1)
78         {
79             temp1 = temp0;
80         }
81         if(count==loc2)
82         {
83             temp2 = temp0;
84             break;
85         }
86         count++;
87     }
88     if(type == 1)
89     {
90         return temp2;
91     }
92     else
93     {
94         return (temp1+temp2)/2;
95     }
96 }
97
98 }
99 }
100
101 /*
102 执行用时 :22~44 ms
103 内存消耗 :7.6~7.7MB
104 */
105 double findMedianSortedArrays( int*   nums1   ,
106                                int    nums1Size ,
107                                int*   nums2   ,
108                                int    nums2Size
109                                )
110 {
111
112     int    temp      =  nums1Size  ;
113     int*   nums      =  NULL       ;
114     int    type      =  0          ;
115     int    loc1      =  0          ;
116     int    loc2      =  0          ;
117     int    count     =  0          ;
118     int    i         =  0          ;

```

```

119     int    j          =    0          ;
120     double temp0      =    0.0        ;
121     double temp1      =    0.0        ;
122     double temp2      =    0.0        ;
123
124
125     if(nums1Size<nums2Size)
126     {
127         temp          =    nums1Size  ;
128         nums1Size     =    nums2Size  ;
129         nums2Size     =    temp       ;
130         nums          =    nums1      ;
131         nums1         =    nums2      ;
132         nums2         =    nums       ;
133     }
134
135     type = (nums1Size % 2 + nums2Size % 2) % 2;
136     loc1 = (nums1Size - nums2Size)/2 + nums2Size -1 ;
137     loc2 = loc1+1;
138
139     if( (nums1Size==0)
140         &&(nums2Size==0)
141     )
142     {
143         return 0.0;
144     }
145     else if(nums2Size ==0)
146     {
147         if(nums1Size%2==0)
148         {
149             return (double)(nums1[nums1Size/2-1]+nums1[nums1Size/2])/2.0;
150         }
151         else
152         {
153             return (double)(nums1[nums1Size/2]);
154         }
155     }
156
157     else if (nums1Size ==0)
158     {
159         if(nums2Size%2==0)
160         {
161             return (double)(nums2[nums2Size/2-1]+nums2[nums2Size/2])/2.0;
162         }
163         else
164         {
165             return (double)(nums2[nums2Size/2]);
166         }
167     }
168     else
169     {
170         while (1)
171         {
172             if( (i<nums1Size)
173                 &&(j<nums2Size)
174             )
175             {
176                 if(nums1[i]<=nums2[j])

```

```

177         {
178             temp0 = nums1[i];
179             i++;
180
181         }
182         else
183         {
184             temp0 = nums2[j];
185             j++;
186
187         }
188
189     }
190     else if(i<nums1Size)
191     {
192         temp0 = nums1[i];
193         i++;
194
195     }
196     else if(j<nums2Size)
197     {
198         temp0 = nums2[j];
199         j++;
200
201     }
202     else
203     {
204         break;
205     }
206
207     if(count==loc1)
208     {
209         temp1 = temp0;
210     }
211     if(count==loc2)
212     {
213         temp2 = temp0;
214         break;
215     }
216     count++;
217 }
218 if(type == 1)
219 {
220     return temp2;
221 }
222 else
223 {
224     return (temp1+temp2)/2;
225 }
226 }
227 }
228 /*
229 执行结果:
230 通过
231 显示详情
232 执行用时 :24 ms, 在所有 C 提交中击败了63.63% 的用户
233 内存消耗 :7.6 MB, 在所有 C 提交中击败了87.35%的用户
234 */

```

方法二:C_递归二分.

```
1
2  int getKth (   int*   nums1   ,
3                int    start1  ,
4                int    end1    ,
5                int*   nums2   ,
6                int    start2  ,
7                int    end2    ,
8                int    k        /*要找第k小的数(k=1,2,3,...)*/
9                )
10 {
11     int len1    =   end1 - start1 + 1   ;
12     int len2    =   end2 - start2 + 1   ;
13     int i       =   start1              ;
14     int j       =   start2              ;
15
16     if(len1 > len2) /*保证len1<=len2*/
17     {
18         return getKth(nums2,start2,end2,nums1,start1,end1,k);
19     }
20     /* 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*/
21     if((len1==0)&&(len2==0))
22     {
23         return 0; /*若一开始输入的就是两个长度为0的数组,直接返回*/
24     }
25     else if(len1 == 0)
26     {
27         return nums2[start2+k-1];
28     }
29     /* 递归结束条件2,找到最小点数据(第1小)*/
30     if(k==1)
31     {
32         return nums1[start1] <= nums2[start2] ? nums1[start1] :
nums2[start2];
33     }
34
35     /*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*/
36     i += (len1 <= k/2 ? len1 : k/2 ) - 1;
37     j += (len2 <= k/2 ? len2 : k/2 ) - 1;
38
39     if(nums1[i]>nums2[j])
40     {
41         return getKth(nums1,start1,end1,nums2,j+1,end2,k-(j-start2+1));
42     }
43     else
44     {
45         return getKth(nums1,i+1,end1,nums2,start2,end2,k-(i-start1+1));
46     }
47 }
48
49 double findMedianSortedArrays( int*   nums1   ,
50                                int    nums1Size ,
51                                int*   nums2   ,
52                                int    nums2Size
```

```

53         )
54     {
55         unsigned int left    =    (nums1Size + nums2Size+1 )/2        ;    /*用
unsigned int 做防溢出*/
56         unsigned int right  =    (nums1Size + nums2Size    )/2 + 1    ;
57
58         if(left!=right)
59         {
60             return    0.5*(getKth(nums1,0,nums1Size-1,nums2,0,nums2Size-1,left) +
getKth(nums1,0,nums1Size-1,nums2,0,nums2Size-1,right) );
61         }
62         else
63         {
64             return getKth(nums1,0,nums1Size-1,nums2,0,nums2Size-1,left);
65         }
66     }
67
68     /*
69     执行结果:
70     通过
71     显示详情
72     执行用时 :16 ms, 在所有 C 提交中击败了95.77% 的用户
73     内存消耗 :7.4 MB, 在所有 C 提交中击败了95.45%的用户
74     */

```