Solution

Approach 1: Recursive Approach

To solve this problem, we need to understand "What is the use of median". In statistics, the median is used for:

> Dividing a set into two equal length subsets, that one subset is always greater than the other.

If we understand the use of median for dividing, we are very close to the answer.

First let's cut A into two parts at a random position i:

```
      left_A             |        right_A
A[0], A[1], ..., A[i-1]  |  A[i], A[i+1], ..., A[m-1]
```

Since A has m elements, so there are $m + 1$ kinds of cutting ($i = 0 \sim m$).

And we know:

> $\text{len}(\text{left\_A}) = i, \text{len}(\text{right\_A}) = m - i.$
>
> Note: when $i = 0$, left_A is empty, and when $i = m$, right_A is empty.

With the same way, cut B into two parts at a random position j:

```
      left_B             |        right_B
B[0], B[1], ..., B[j-1]  |  B[j], B[j+1], ..., B[n-1]
```

Put left_A and left_B into one set, and put right_A and right_B into another set. Let's name them left_part and right_part:

```
      left_part          |        right_part
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]
```

If we can ensure:

> 1. $\text{len}(\text{left\_part}) = \text{len}(\text{right\_part})$
> 2. $\max(\text{left\_part}) \leq \min(\text{right\_part})$

then we divide all elements in $\{\text{A}, \text{B}\}$ into two parts with equal length, and one part is always greater than the other. Then

$$\text{median} = \frac{\max(\text{left\_part}) + \min(\text{right\_part})}{2}$$

To ensure these two conditions, we just need to ensure:

> 1. $i + j = m - i + n - j$ (or: $m - i + n - j + 1$)
>
>   if $n \geq m$, we just need to set: $i = 0 \sim m$, $j = \frac{m+n+1}{2} - i$
> 2. $\text{B}[j-1] \leq \text{A}[i]$ and $\text{A}[i-1] \leq \text{B}[j]$

ps.1 For simplicity, I presume $\mathrm{A}[i-1], \mathrm{B}[j-1], \mathrm{A}[i], \mathrm{B}[j]$ are always valid even if $i = 0$, $i = m$, $j = 0$, or $j = n$. I will talk about how to deal with these edge values at last.

ps.2 Why $n \geq m$? Because I have to make sure j is non-negative since $0 \leq i \leq m$ and $j = \frac{m+n+1}{2} - i$. If $n < m$, then j may be negative, that will lead to wrong result.

So, all we need to do is:

> Searching i in $[0, m]$, to find an object i such that:
>
> $$\mathrm{B}[j-1] \leq \mathrm{A}[i] \text{ and } \mathrm{A}[i-1] \leq \mathrm{B}[j], \text{ where } j = \frac{m+n+1}{2} - i$$

And we can do a binary search following steps described below:

1. Set $\mathrm{imin} = 0, \mathrm{imax} = m$, then start searching in $[\mathrm{imin}, \mathrm{imax}]$
2. Set $i = \frac{\mathrm{imin}+\mathrm{imax}}{2}$, $j = \frac{m+n+1}{2} - i$
3. Now we have $\mathrm{len}(\mathrm{left\_part}) = \mathrm{len}(\mathrm{right\_part})$. And there are only 3 situations that we may encounter:
   - $\mathrm{B}[j-1] \leq \mathrm{A}[i]$ and $\mathrm{A}[i-1] \leq \mathrm{B}[j]$
     Means we have found the object i, so stop searching.
   - $\mathrm{B}[j-1] > \mathrm{A}[i]$
     Means $\mathrm{A}[i]$ is too small. We must adjust i to get $\mathrm{B}[j-1] \leq \mathrm{A}[i]$.
     Can we increase i?
     Yes. Because when i is increased, j will be decreased.
     So $\mathrm{B}[j-1]$ is decreased and $\mathrm{A}[i]$ is increased, and $\mathrm{B}[j-1] \leq \mathrm{A}[i]$ may be satisfied.
     Can we decrease i?
     No! Because when i is decreased, j will be increased.
     So $\mathrm{B}[j-1]$ is increased and $\mathrm{A}[i]$ is decreased, and $\mathrm{B}[j-1] \leq \mathrm{A}[i]$ will be never satisfied.
     So we must increase i. That is, we must adjust the searching range to $[i+1, \mathrm{imax}]$.
     So, set $\mathrm{imin} = i + 1$, and goto 2.

- $A[i-1] > B[j]$:
  Means $A[i-1]$ is too big. And we must decrease i to get $A[i-1] \leq B[j]$.
  That is, we must adjust the searching range to $[imin, i-1]$.
  So, set $imax = i-1$, and goto 2.

When the object i is found, the median is:

> $\max(A[i-1], B[j-1])$, when $m+n$ is odd
>
> $\frac{\max(A[i-1],B[j-1])+\min(A[i],B[j])}{2}$, when $m+n$ is even

Now let's consider the edges values $i=0, i=m, j=0, j=n$ where $A[i-1], B[j-1], A[i], B[j]$ may not exist. Actually this situation is easier than you think.

What we need to do is ensuring that $\max(\text{left\_part}) \leq \min(\text{right\_part})$. So, if i and j are not edges values (means $A[i-1], B[j-1], A[i], B[j]$ all exist), then we must check both $B[j-1] \leq A[i]$ and $A[i-1] \leq B[j]$. But if some of $A[i-1], B[j-1], A[i], B[j]$ don't exist, then we don't need to check one (or both) of these two conditions. For example, if $i=0$, then $A[i-1]$ doesn't exist, then we don't need to check $A[i-1] \leq B[j]$. So, what we need to do is:

> Searching i in $[0, m]$, to find an object i such that:
>
> $(j = 0$ or $i = m$ or $B[j-1] \leq A[i])$ and $\frac{m+n+1}{2}$
>
> $(i = 0$ or $j = n$ or $A[i-1] \leq B[j])$, where $j =$ $- i$

And in a searching loop, we will encounter only three situations:

1. $(j = 0$ or $i = m$ or $\mathrm{B}[j-1] \leq \mathrm{A}[i])$ and
   $(i = 0$ or $j = n$ or $\mathrm{A}[i-1] \leq \mathrm{B}[j])$
   Means i is perfect, we can stop searching.
2. $j > 0$ and $i < m$ and $\mathrm{B}[j-1] > \mathrm{A}[i]$
   Means i is too small, we must increase it.
3. $i > 0$ and $j < n$ and $\mathrm{A}[i-1] > \mathrm{B}[j]$
   Means i is too big, we must decrease it.

Notes

Thanks to @Quentin.chen (https://leetcode.com/Quentin.chen) for pointing out that:
$i < m \implies j > 0$ and $i > 0 \implies j < n$. Because:

$$m \leq n,\ i < m \implies j = \underbrace{\frac{m+n+1}{2}}_{} - i > \underbrace{\frac{m+n+1}{2}}_{} - m \geq \underbrace{\frac{2m+1}{2}}_{} - m \geq 0$$

$$m \leq n,\ i > 0 \implies j = \qquad - i < \qquad \leq \qquad \leq n$$

So in situation 2. and 3. , we don't need to check whether $j > 0$ and whether $j < n$.

Java | Python

Copy

```java
class Solution {
    public double findMedianSortedArrays(int[] A, int[] B) {
        int m = A.length;
        int n = B.length;
        if (m > n) { // to ensure m<=n
            int[] temp = A; A = B; B = temp;
            int tmp = m; m = n; n = tmp;
        }
        int iMin = 0, iMax = m, halfLen = (m + n + 1) / 2;
        while (iMin <= iMax) {
            int i = (iMin + iMax) / 2;
            int j = halfLen - i;
            if (i < iMax && B[j-1] > A[i]){
                iMin = i + 1; // i is too small
            }
            else if (i > iMin && A[i-1] > B[j]) {
                iMax = i - 1; // i is too big
            }
            else { // i is perfect
                int maxLeft = 0;
                if (i == 0) { maxLeft = B[j-1]; }
                else if (j == 0) { maxLeft = A[i-1]; }
                else { maxLeft = Math.max(A[i-1], B[j-1]); }
                if ( (m + n) % 2 == 1 ) { return maxLeft; }

                int minRight = 0;
                if (i == m) { minRight = B[j]; }
```

Notes

Complexity Analysis

- Time complexity: $O\big(\log\big(\min(m,n)\big)\big)$.
  At first, the searching range is $[0, m]$. And the length of this searching range will be reduced by half after each loop. So, we only need $\log(m)$ loops. Since we do constant operations in each loop, so the time complexity is $O\big(\log(m)\big)$. Since $m \leq n$, so the time complexity is $O\big(\log\big(\min(m,n)\big)\big)$.
- Space complexity: $O(1)$.
  We only need constant memory to store 9 local variables, so the space complexity is $O(1)$.

Comments: 189

Sort By

Type comment here... (Markdown is supported)

Preview | Post

EVikVik (/evikvik) ★ 2 3 days ago

Why we just add one here? i+j=m−i+n−j (or: m - i + n - j + 1)

2 Share Reply

SHOW 1 REPLY

user6698K (/user6698k) ★ 0 September 4, 2018 10:27 AM

For Python 3 code :

```
    nums = nums1 + nums2
    nums.sort()
```

Read More

0 Share Reply

SHOW 2 REPLIES

halalyon (/halalyon) ★ 0 August 31, 2018 12:02 PM

Pretty detailed and wonderful solution! Look through it several times!

0 Share Reply

DJPoland19 (/djpoland19) ★ 8 August 31, 2018 3:05 AM

Quite easy in Python 3. Here's my code:

```
class Solution:
    def findMedianSortedArrays(self, nums1, nums2):
        newNums = nums1 + nums2
```

Read More

8 Share Reply

SHOW 7 REPLIES

miaoyanting1997 (/miaoyanting1997) ★ 10 August 29, 2018 8:15 PM

my python solution:

Read More

0 Share Reply

SHOW 1 REPLY

kaiteli14 (/kaiteli14) ★ 1 August 28, 2018 8:11 AM

how is this O(log(min(m,n))? if you look at iMin and iMax, it only increments by 1 or decrements by 1 and it basically increment or decrement (iMin+iMax)/2 by 1 which will go through m/2 items, which is clearly O(m/2) = O(min(m,n)), the log part is not making any sense. We are not picking right or left to search, we are just increasing iMin or decreasing iMax by 1 which definitely looked linear to me, am I missing something here? Can anyone help me? Thank you.

Read More

0 Share Reply

SHOW 5 REPLIES

Notes

gys (/gys) ★ 7 August 24, 2018 3:09 PM

gys:my c++ solution:

Read More

3 Share Reply

vincenttien (/vincenttien) ★ 0 August 20, 2018 7:24 PM

My C# solution:

The assumption was stated that both arrays are sorted so you just need to merge to nums1.Length + nums2.Length / 2 and just take the last element (or calc the mean on the last two elements)

Read More

0 Share Reply

SHOW 1 REPLY

sailfish (/sailfish) ★ 14 August 17, 2018 6:11 AM

My Python3 solution without using list merge and sort features, I return the value immediately after I get it and I hope can save some time by this way, but I failed and still cannot fight the version using Python3 library functions. There are too many checks in the code.

My code is below, just for fun.

Read More

1 Share Reply

SHOW 2 REPLIES

ricace (/ricace) ★ 0 August 15, 2018 8:16 AM

Thanks for solution

0 Share Reply

< 1 2 3 4 5 6 ... 18 19 >