Previous Next

4. Median of Two Sorted Arrays

Aug. 30, 2017 | 751.3K views

Average Rating: 3.47 (718 votes)

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)).

You may assume **nums1** and **nums2** cannot be both empty.

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
```

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 $\text{A}[i-1], \text{B}[j-1], \text{A}[i], \text{B}[j]$ 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:
>
> $\text{B}[j-1] \leq \text{A}[i]$ and $\text{A}[i-1] \leq \text{B}[j]$, where $j = \frac{m+n+1}{2} - i$

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

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

When the object i is found, the median is:

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

Now let's consider the edges values $i = 0, i = m, j = 0, j = n$ where $\text{A}[i-1], \text{B}[j-1], \text{A}[i], \text{B}[j]$ 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 $\text{A}[i-1], \text{B}[j-1], \text{A}[i], \text{B}[j]$ all exist), then we must check both $\text{B}[j-1] \leq \text{A}[i]$ and $\text{A}[i-1] \leq \text{B}[j]$. But if some of $\text{A}[i-1], \text{B}[j-1], \text{A}[i], \text{B}[j]$ don't exist, then we don't need to check one (or both) of these two conditions. For example, if $i = 0$, then $\text{A}[i-1]$ doesn't exist, then we don't need to check $\text{A}[i-1] \leq \text{B}[j]$. 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 $\text{B}[j-1] \leq \text{A}[i]$) and
> ($i = 0$ or $j = n$ or $\text{A}[i-1] \leq \text{B}[j]$), where $j = \frac{m+n+1}{2} - i$

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

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

Thanks to @Quentin.chen for pointing out that: $i < m \implies j > 0$ and $i > 0 \implies j < n$. Because:

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

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

Java | Python | Copy

```
6          raise ValueError
7
8      imin, imax, half_len = 0, m, (m + n + 1) / 2
9      while imin <= imax:
10         i = (imin + imax) / 2
11         j = half_len - i
12         if i < m and B[j-1] > A[i]:
13             # i is too small, must increase it
14             imin = i + 1
15         elif i > 0 and A[i-1] > B[j]:
16             # i is too big, must decrease it
17             imax = i - 1
18         else:
19             # i is perfect
20
21             if i == 0: max_of_left = B[j-1]
22             elif j == 0: max_of_left = A[i-1]
23             else: max_of_left = max(A[i-1], B[j-1])
24
25             if (m + n) % 2 == 1:
26                 return max_of_left
27
28             if i == m: min_of_right = B[j]
29             elif j == n: min_of_right = A[i]
30             else: min_of_right = min(A[i], B[j])
31
32             return (max_of_left + min_of_right) / 2.0
```

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 $O(1)$.

Rate this article:

Previous Next

Comments: 595

Sort By

Type comment here... (Markdown is supported)

Preview Post

contacttoakhil 2312 March 21, 2019 7:19 PM

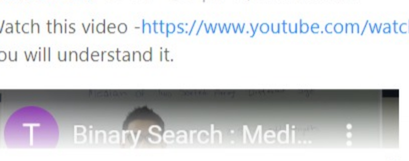
If idea behind an explanation is to confuse the audience by fancy mathematical stuff then congratulations, mission accomplished.

2262 Share Reply

SHOW 23 REPLIES

praveennvs0 639 April 19, 2019 9:33 AM Report

Watch this video -https://www.youtube.com/watch?v=LPFhl65R7ww&t=1013s
You will understand it.



Read More

639 Share Reply

SHOW 17 REPLIES

ahmedahamid 196 March 10, 2019 11:46 PM

a gazillion times better explanation is this: https://medium.com/@hazemu/finding-the-median-of-2-sorted-arrays-in-logarithmic-time-1d3f2ecbeb46

194 Share Reply

SHOW 16 REPLIES

BrianChesbrough 87 February 6, 2019 7:11 AM Report

wut

87 Share Reply

SHOW 1 REPLY

zhutianqi 235 June 16, 2018 11:32 AM

This code is wrong, time complexity is wrong.

99 Share Reply

SHOW 8 REPLIES

yayawest 66 June 8, 2018 9:39 AM

Why is this approach considered a recursive approach if a while loop is used? Is this not an iterative approach?

66 Share Reply

SHOW 5 REPLIES

ishanguliani 89 February 25, 2019 1:18 AM Report

do interviewers expect this ?

85 Share Reply

SHOW 14 REPLIES

garyoakenshield 53 February 9, 2019 1:37 PM

Condensed: the median is the `(m+n+1)/2` th item if you were to combine `A` and `B` and sort them. This algorithm looks at how many items from `A` it makes sense to add to a list (call it `C`) of length `(m+n+1)/2`, filling the rest in with items from `B`. The last item of `C` is the median.

Take the first half of `A` and add them to `C`, then fill the rest in with `B` (i.e., `B[0:j]` where `j = |C|`

Read More

52 Share Reply

SHOW 2 REPLIES

shravan9 79 February 1, 2019 3:55 PM Report

where is the recursion?

75 Share Reply

SHOW 4 REPLIES

CodeP 31 July 11, 2018 3:21 AM

Java implementation is not O(log(min(m,n)))
Line 14 should be iMin = i + 1; // i is too small

Line 17: iMax = i - 1

31 Share Reply

SHOW 1 REPLY