算法：median

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_AA[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 ~ m**). And we know: **len(left_A) = i**, **len(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_BB[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_partA[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) len(left_part) == len(right_part)2) max(left_part) <= min(right_part)

then we divide all elements in {**A, B**} into two parts with equal length, and one part is always greater than the other. Then **median = (max(left_part) + min(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 >= m, we just need to set: i = 0 ~ m, j = (m + n + 1)/2 - i(2) B[j-1] <= A[i] **and** A[i-1] <= B[j]

ps.1 For simplicity, I presume **A[i-1],B[j-1],A[i],B[j]** are always valid even if **i=0/i=m/j=0/j=n** . I will talk about how to deal with these edge values at last.
ps.2 Why n >= m? Because I have to make sure j is non-nagative since 0 <= i <= m and j = (m + n + 1)/2 - i. If n < m , then j may be nagative, that will lead to wrong result.
So, all we need to do is:
Searching i **in** [0, m], to find an object `i` that: B[j-1] <= A[i] **and** A[i-1] <= B[j], (where j = (m + n + 1)/2 - i)

And we can do a binary search following steps described below:
<1> Set imin = 0, imax = m, **then** start searching **in** [imin, imax]<2> Set i = (imin + imax)/2, j = (m + n + 1)/2 - i<3> Now we have len(left_part)==len(right_part). And there are only 3 situations that we may encounter: B[j-1] <= A[i] **and** A[i-1] <= B[j] Means we have found the object `i`, so stop searching. **B[j-1] > A[i] Means A[i] is too small. We must** `ajust` **i to get** `B[j-1] <= A[i]`. **Can we** `increase` **i? Yes. Because when i is increased, j will be decreased. So B[j-1] is decreased and A[i] is increased, and** `B[j-1] <= A[i]` **may be satisfied. Can we** `decrease` **i?** `No!` **Because when i is decreased, j will be increased. So B[j-1] is increased and A[i] is decreased, and B[j-1] <= A[i] will be never satisfied. So we must** `increase` **i. That is, we must ajust the searching range to [i+1, imax]. So, set 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] <=B[j]`. **That is, we must ajust 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)**or** (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(left_part) <= min(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] <= A[i] and A[i-1] <= 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] <= B[j]. So, what we need to do is:

Searching i **in** [0, m], to find an object `i` that: (j == 0**or** i == m **or** B[j-1] <= A[i]) **and** (i == 0**or** j == n **or** A[i-1] <= B[j]) where j = (m + n + 1)/2 - i

And in a searching loop, we will encounter only three situations:
(j == 0**or** i == m **or** B[j-1] <= A[i]) **and** (i == 0**or** j = n **or** A[i-1] <= B[j]) Means i **is** perfect, we can stop searching. **j > 0and i < m and B[j - 1] > A[i] Means i is too small, we must increase it. i > 0and j < n and A[i - 1] > B[j] Means i is too big, we must decrease it.**

Thank @Quentin.chen , him pointed out that: i < m ==> j > 0 and i > 0 ==> j < n . Because:
m <= n, i < m ==> j = (m+n+1)/2 - i > (m+n+1)/2 - m >= (2*m+1)/2 - m >= 0 m <= n, i > 0 ==> j = (m+n+1)/2 - i < (m+n+1)/2 <= (2*n+1)/2 <= n

So in situation and , we don't need to check whether j > 0 and whether j < n.
Below is the accepted code:

```
defmedian(A, B):  m, n = len(A), len(B)  if m > n:  A, B, m, n = B, A, n, m  if n == 0:  raise ValueError  imin, imax, half_len = 0, m, (m + n + 1) / 2while imin <= imax:  i = (imin + imax) / 2  j = half_len - i  if i < m and B[j-1] > A[i]:  # i is too small, must increase it  imin = i + 1elif i > 0and A[i-1] > B[j]:  # i is too big, must decrease it  imax = i - 1else:  # i is perfectif i == 0: max_of_left = B[j-1]  elif j == 0: max_of_left = A[i-1]  else: max_of_left = max(A[i-1], B[j-1])  if (m + n) % 2 == 1:  return max_of_left  if i == m: min_of_right = B[j]  elif j == n: min_of_right = A[i]  else: min_of_right = min(A[i], B[j])  return (max_of_left + min_of_right) / 2.0
```