Given a sorted array A with length m, we can split it into two part:

```
{ A[0], A[1], ... , A[i - 1] } | { A[i], A[i + 1], ... , A[m - 1] }
```

All elements in right part are greater than elements in left part.

The left part has "i" elements, and right part has "m - i" elements.

There are "m + 1" kinds of splits. (i = 0 ~ m)

When i = 0, the left part has "0" elements, right part has "m" elements.

When i = m, the left part has "m" elements, right part has "0" elements.

For array B, we can split it with the same way:

```
{ B[0], B[1], ... , B[j - 1] } | { B[j], B[j + 1], ... , B[n - 1] }
```

The left part has "j" elements, and right part has "n - j" elements.

Put A's left part and B's left part into one set. (Let's name this set "LeftPart")

Put A's right part and B's right part into one set. (Let's name this set"RightPart")

```
           LeftPart           |            RightPart
{ 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) LeftPart's length == RightPart's length (or RightPart's length + 1)

2) All elements in RightPart is greater than elements in LeftPart.
```

then we split all elements in {A, B} into two parts with eqaul length, and one part is

always greater than the other part. Then the median can be easily found.

To ensure these two condition, 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]

    considering edge values, we need to ensure:

          (j == 0 or i == m or B[j - 1] <= A[i]) and

              (i == 0 or j == n or A[i - 1] <= B[j])
```

So, all we need to do is:

```
Search i from 0 to m, to find an object "i" to meet condition (1) and (2) above.
```

And we can do this search by binary search. How?

If B[j0 - 1] > A[i0], then the object "ix" can't be in [0, i0]. Why?

```
Because if ix < i0, then jx = (m + n + 1) / 2 - ix > j0,

then B[jx - 1] >= B[j0 - 1] > A[i0] >= A[ix]. This violates

the condition (2). So ix can't be less than i0.
```

And if A[i0 - 1] > B[j0], then the object "ix" can't be in [i0, m].

So we can do the binary search following steps described below:

```
1. set imin, imax = 0, m, then start searching in [imin, imax]

2. i = (imin + imax) / 2; j = (m + n + 1) / 2 - i

3. if B[j - 1] > A[i]: continue searching in [i + 1, imax]
   elif A[i - 1] > B[j]: continue searching in [imin, i - 1]
   else: bingo! this is our object "i"
```

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)
```

Below is the accepted code:

```
def Median2(A, B):
    m, n = len(A), len(B)

    if m > n:
        return Median2(B, A)

    imin, imax = 0, m
    while imin <= imax:
        i = (imin + imax) / 2
        j = (m + n + 1) / 2 - i
        if j > 0 and i < m and B[j - 1] > A[i]:
            imin = i + 1
        elif i > 0 and j < n and A[i - 1] > B[j]:
            imax = i - 1
        else:
            if i == 0:
                num1 = B[j - 1]
            elif j == 0:
                num1 = A[i - 1]
            else:
                num1 = max(A[i - 1], B[j - 1])

            if (m + n) & 1:
                return num1

            if i == m:
                num2 = B[j]
            elif j == n:
                num2 = A[i]
            else:
                num2 = min(A[i], B[j])

            return (num1 + num2) / 2.0
```