Median of Two Sorted Arrays

Victor Busa
victor.busa@gmail.com

May 22, 2018

1 Subject

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 $\mathcal{O}(log(m+n))$.

Example 1:

```
nums1 = [1, 3]
nums2 = [2]
```

The median is 2.0

Example 2:

```
nums1 = [1, 2]
nums2 = [3, 4]
```

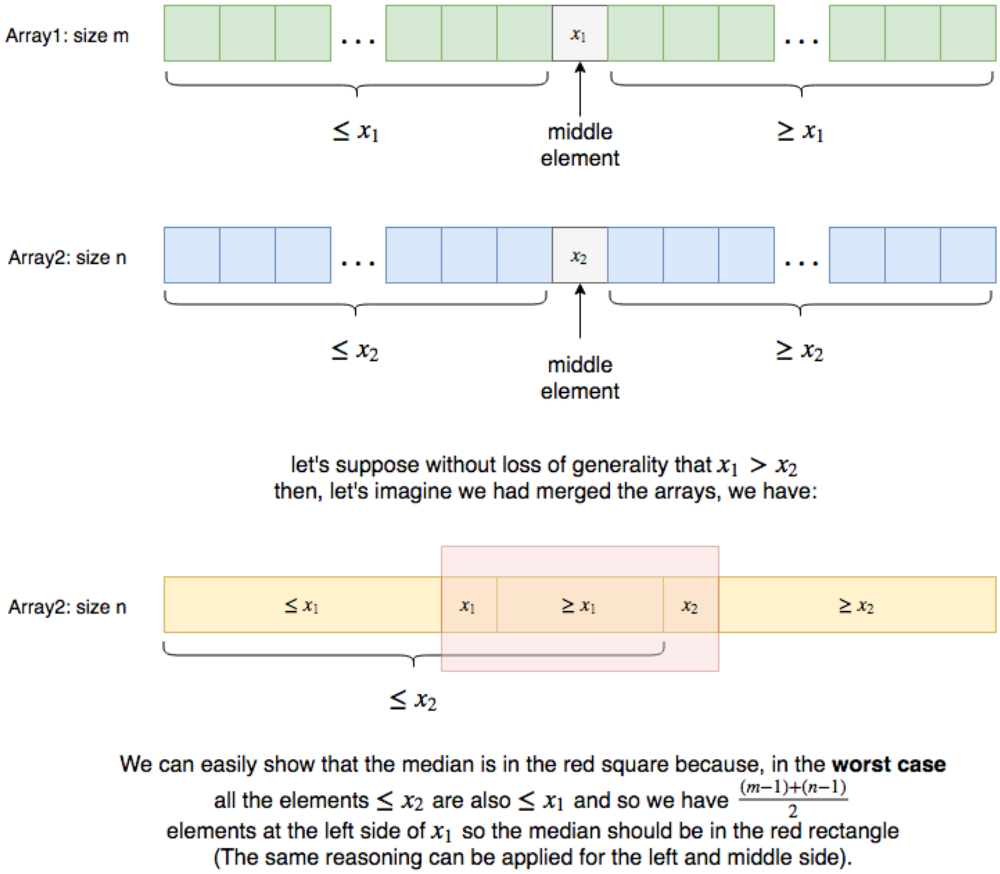
The median is $(2+3)/2 = 2.5$

2 Solution

As the solution should be in the order of $\mathcal{O}(log(m+n))$, we already know that the solution does not involve any sorting algorithm, because it would mean that we check at least all the number that is to say a complexity of $\mathcal{O}(min(n,m))$. How can be tackle this problem? Firstly we know that we have 2 **sorted** arrays and we know that we want a **logarithmic** complexity. So actually the algorithm that comes right away to our mind when we talk about sorted array and logarithmic complexity is the **dichotomic search**. The question is then how can be adapt the dichotomic search so that it gives us the median of 2 sorted arrays?

2.1 2.1 Wrong attempt

I would like to share with you an (wrong) idea just to let you know that nobody can always come with the right approach straightaway! My first idea was to split each array in half. Based on the split we can then determined in which part of the 2 arrays the median should remain. The idea was to apply this idea recursively in such a way that at the end I have the median. A picture is better than 1000 words, so the idea is resumed on Figure 2.1



let's suppose without loss of generality that $x_1 > x_2$
then, let's imagine we had merged the arrays, we have:

We can easily show that the median is in the red square because, in the **worst case**
all the elements $\leq x_2$ are also $\leq x_1$ and so we have $\frac{(m-1)+(n-1)}{2}$
elements at the left side of x_1 so the median should be in the red rectangle
(The same reasoning can be applied for the left and middle side).

Figure 2.1: First attempt: Wrong idea

This idea is correct at the first iteration but it cannot be applied recursively as show the following counter-example:

1. iteration of algorithm:

```
nums1 = [1, 7, 8, 9, 14, 15, 30, 36, 42, 78, 96, 102, 106]
nums2 = [1, 2, 5, 6, 10, 11, 17, 24]
len(nums1) = 13 # middle element = 30
len(nums2) = 8 # middle element = 10
```

As $30 > 10$ then we will keep the left part of nums1 and the right part of nums2

2. iteration of algorithm:

```
nums1 = [1, 7, 8, 9, 14, 15, 30]
nums2 = [10, 11, 17, 24]
len(nums1) = 7 # middle element = 9
len(nums2) = 5 # middle element = 17
```

As $17 > 9$ then we will keep the right part of nums1 and the left part of nums2

3. iteration of algorithm:

```
nums1 = [9, 14, 15, 30]
nums2 = [10, 11, 17]
```

```
len(nums1) = 4 # middle element = 15
len(nums2) = 3 # middle element = 17
```

As $17 > 15$ then we will keep the right part of nums1 and the left part of nums2

4. iteration of algorithm:

```
nums1 = [15, 30]
nums2 = [10, 11, 17]
len(nums1) = 2 # middle element = 30
len(nums2) = 3 # middle element = 11
```

At this iteration the algorithm just removed the number **14** which is the median in this example... So it doesn't work and it comes from the fact that the median of the 2 subarrays is not the median of the whole 2 arrays.

2.2 Solution

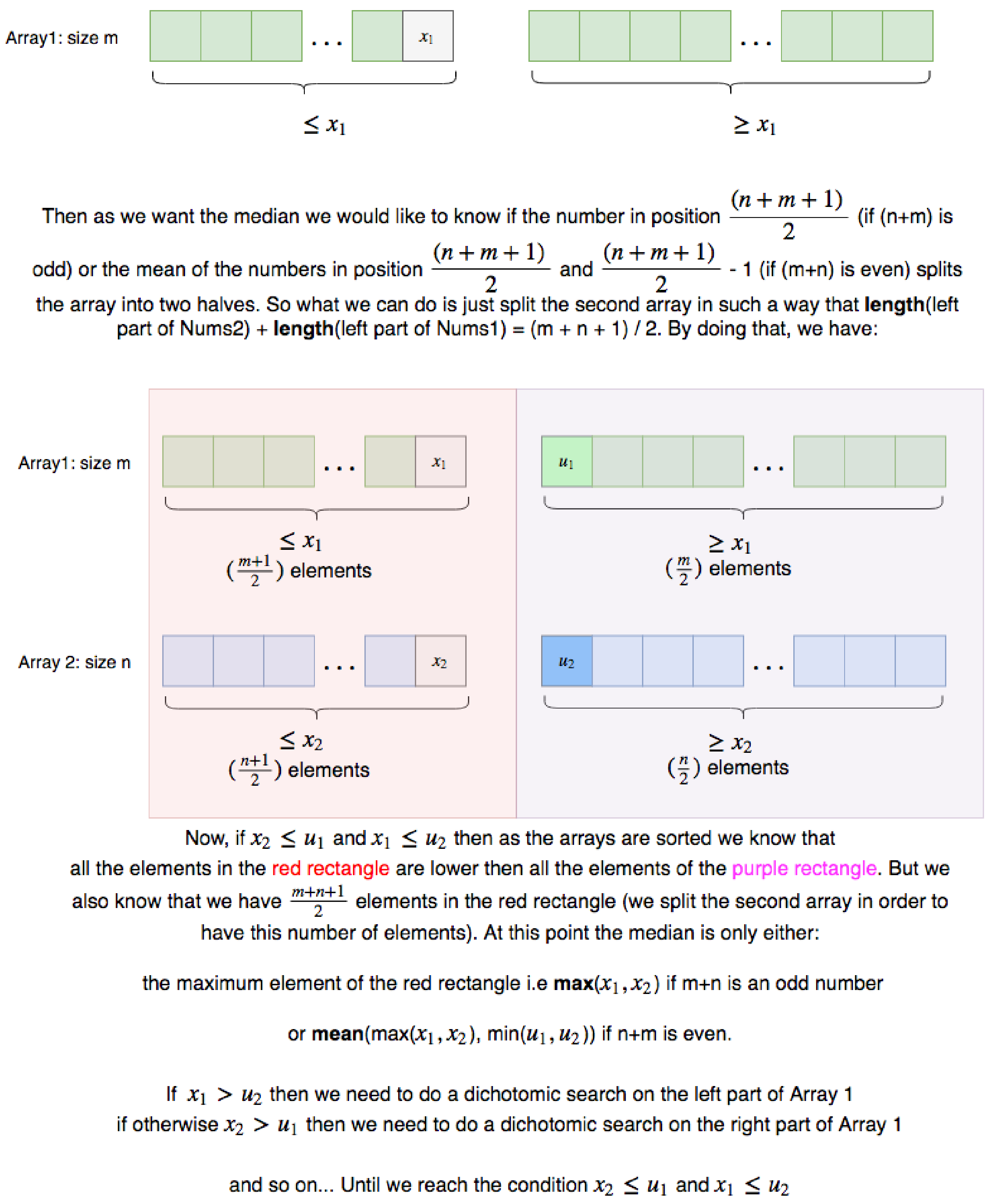
So now let's find out a good solution to solve this problem. Maybe we can just do a dichotomic search on one of the 2 arrays. Ok but then what? Let's suppose we do the first iteration of our dichotomic search and that our array on which we are performing the dichotomic search is of size m. Again, a picture is better than 1000 words, so the algorithm is sum up on Figure 2.2

Well there are edge cases that we should take into account but the idea of the algorithm remains the same. We should still understand why this algorithm will terminate at some point... That is to say... Are we sure we will hit the condition $x_1 < u_2$ and $x_2 < u_1$ at some point?

Assume x_1 is always greater than u_2. By construction of the algorithm that means that x_1 will decrease and u_2 will increase at each iteration of the algorithm such that the smallest element of array 1 will be greater than a certain number at a certain position of array 2 (in the case of m $>$ n) or a certain element of array 1 will be greater than the biggest element of array 2 (if n $>$ m). In both case the algorithm will return the median by construction. We just need to use ∞ and $-\infty$ for the edge cases (the edge cases are treated in the code).

Finally one last thing we want to know is the complexity of the algorithm. Actually as we run a dichotomic search on only one array and that all the other operations are constants (mean, max, min, increment,...) the complexity of the algorithm is logarithmic in the length of the array we choose for the dichotomic search. As we want the lowest complexity possible we can just do the dichotomic search on the array that has the less number of elements... So finally the complexity is:

Time complexity: $\mathcal{O}(log(min(n,m)))$
Space complexity: $\mathcal{O}(1)$



Then as we want the median we would like to know if the number in position $\frac{(n+m+1)}{2}$ (if (n+m) is odd) or the mean of the numbers in position $\frac{(n+m+1)}{2}$ and $\frac{(n+m+1)}{2}$ - 1 (if (m+n) is even) splits the array into two halves. So what we can do is just split the second array in such a way that **length**(left part of Nums2) + **length**(left part of Nums1) = (m + n + 1) / 2. By doing that, we have:

Now, if $x_2 \leq u_1$ and $x_1 \leq u_2$ then as the arrays are sorted we know that all the elements in the red rectangle are lower then all the elements of the purple rectangle. But we also know that we have $\frac{m+n+1}{2}$ elements in the red rectangle (we split the second array in order to have this number of elements). At this point the median is only either:

the maximum element of the red rectangle i.e **max**(x_1, x_2) if m+n is an odd number

or **mean**(max(x_1, x_2), min(u_1, u_2)) if n+m is even.

If $x_1 > u_2$ then we need to do a dichotomic search on the left part of Array 1
if otherwise $x_2 > u_1$ then we need to do a dichotomic search on the right part of Array 1

and so on... Until we reach the condition $x_2 \leq u_1$ and $x_1 \leq u_2$

Figure 2.2: Algorithm: The idea is to do a dichotomic search on one of the 2 arrays and then to split the second array in order to have the same number on the right and on the left part.