2. MEDIAN OF TWO SORTED ARRAYS

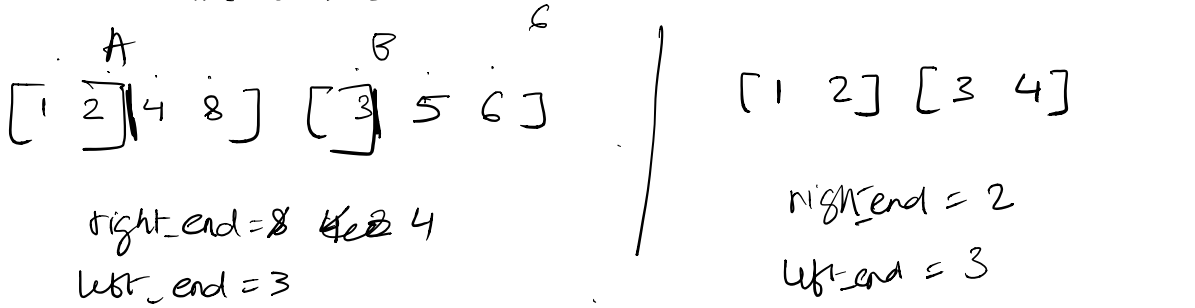
There are 2 sorted arrays nums1 and nums2 of size m and n respectively.

Find the median of the two sorted arrays. The overall runtime complexity should be O(log(m+n)).

You can assume nums1 and nums2 cannot be both empty.

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

median = 2.0



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

1 2 3 4 5 6 8

left_part	right_part
A [1 2]	A [4, 8]
B [3]	B [5, 6]

max(left_part) = 3
min(right_part) = 4
median = $\frac{3+4}{2} = 7/2$ 3.5

$x \rightarrow x_1\ x_2 \mid x_3\ x_4\ x_5\ x_6$

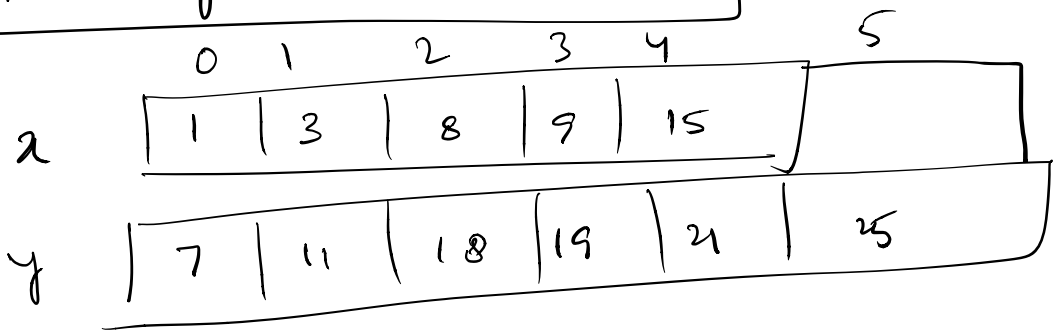
$y \rightarrow y_1\ y_2\ y_3\ y_4\ y_5 \mid y_6\ y_7\ y_8$

we have to find a partition such that;

$$x_2 \leq y_6$$
$$y_5 \leq x_3$$

so, median $= \text{avg}(\max(x_2, y_5), \min(x_3, y_6))$

Complexity $= O(\log(\min(x, y)))$



ALGORITHM:

partition X + Partition y $= (x + y + 1)/2$

If FOUND:

max_left_x $\leq$ min_right_y

max_left_x $\leq$ min_right_y

else IF :

max_left_x > min_right_y

move towards left in x

else :
move towards right in y.

A : | 1 | 4 | 5 | 6 | 8 |
B : | 2 | 3 | 7 |

brute Force:

Approach I

- Combine A and B into C.
- Then sort C → 1 2 3 [4 | 5] 6 7 8 → 4.5

Complexity — O(n log n)

Approach - II

- Pick the smallest element in A and B.

1st pass

A : 1 4 5 6 8
B : 2 3 7

C : 1 move pointer on A

2nd pass

A : 1 4 5 6 8
B : 2 3 7

C : 1 2 move pointer on B.

3rd pass

A : 1 4 5 6 8
B : 2 3 7

C : 1 2 3 move pointer on B.

- keep repeating the process of finding the smallest no. between A and B. And move in the direction of list with smallest element at that index. → This takes $O(n)$

- Now find median using $\left(\frac{n+2}{2}\right)^{th}$ element for odd or avg$\left(n/2^{th}, (n+2)^{th}/2\right)$ for even.

  This takes $O(1)$

- Total Complexity = O

* **OPTIMIZATION — Variation of the Problem Technique**

This technique states that,

"Can you vary or change your problem to create a new problem whose solution will help you solve original problem".

Rephrasing the problem statement using the above heuristic, our problem can be stated as,

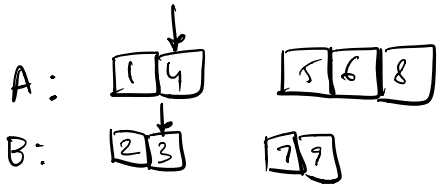
"Given 2 sorted arrays A and B of sizes m, n Find the numbers which are NOT medians of the 2 sorted arrays"

We will delete all the non-medians from array A and B.

To do this we follow the following steps:

- Find a pivot that creates two chunks in both arrays A and B.
- Now we use these pivots to find out the k^{th} element, such that the k^{th} element is the median of A and B ($k = (m+n+1)/2$)

↓ maybe median



Assuming that we don't know anything about A and B.

This might give 2 cases:

pivot(A) + pivot(B) $\geq k$

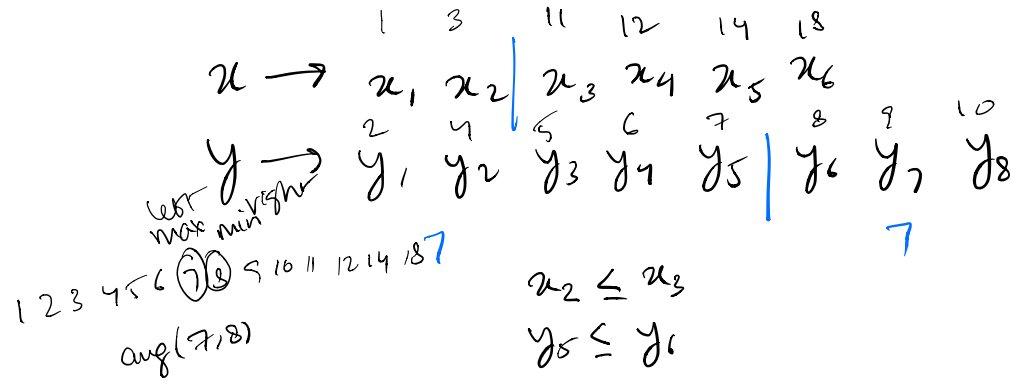
- A[pivot(A)] < B[pivot(B)]
- A[pivot(A)] > B[pivot(B)]

pivot(A) + pivot(B) $< k$.

- " same. "

we want to partition the array into 2 halves, such that no. of elements in each half is exactly same. And every in the left half is $\leq$ every element on the right half.

HOW TO REACH THE ABOVE CONDITION?



$x_2 \leq x_3$

$y_5 \leq y_6$

So if, $x_2 \leq y_6$

$y_5 \leq x_3$

then we have reached a condition where we every element on left half is less than or equal every element on right.

Median will be:

total # elements

- EVEN $\Rightarrow$ $\text{avg}(\max(x_2, y_5), \min(x_3, y_6))$
- ODD $\Rightarrow$ $\max(x_2, y_5)$

Complexity $\rightarrow$ $O(\log(\min(x, y)))$

ALGORITHM / PSEUDO CODE

Use Binary Search

- Given 2 sorted arrays, find the two partitions / pivots, such that, both partitions should have equal no. of elements. ie.;

$$\text{partitionX} + \text{partitionY} = (m+n+1)/2$$

- Now such a partition is FOUND, IF;

COND-1

$$\text{maxLeft\_X} \leq \text{minRight\_Y} \text{ , and}$$
$$\text{maxLeft\_Y} \leq \text{minRight\_X}$$

COND-2

- ELSE IF:

  $\text{maxLeft\_X} > \text{minRight\_Y}$

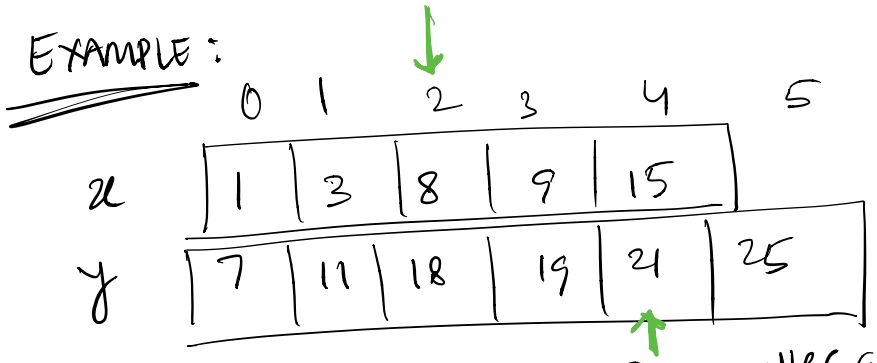
  move towards left in X as we are too much in the right

COND-3

- ELSE:

  move towards right in X as we are too much in left.

EXAMPLE:



Pass 1:

	Left			Right		
X	1	3		8	9	15
Y	7	11	18	19	21	25

start = 0

end = 4

partitionX = 0 $4+0/2 = 2$

partitionY = $\frac{(5+6+1)}{2} - 2$

$= 4$

Part_Y = $\frac{(m+n+1)}{2}$ − Part_X

Checking:

COND-1: max_left_X ≤ min_right_Y ✓

✗ 3 ≤ 21

and, max_left_Y ≤ min_right_X ✗

19 ≤ 8

COND-2: max_left_X > min_right_Y

✗ 8 > 21

COND-3 max_left_X < min_right_Y

✓ 3 < 21

Hence we are too much in the left, we should move right in X.

Pass-2

	0	1	2	3	4	5
x =	1	3	8	9	15	
y =	7	11	18	19	21	25

	left_half			right_half		
x	1	3	8	9	15	
y	7	11	18	19	21	25

start = P_x + 1 = 3

end = 4

$P\_x = (3+4)/2 = 3$

$P\_y = \frac{(m+n+1)}{2} - P\_x$

$= \frac{5+6+1}{2} - 3$

$= 3$

COND-1: $max\_left\_x \leq min\_right\_y$

$8 \leq 9$ ✓

$max\_left\_y \leq min\_right\_x$

$18 \leq 9$ ✗

COND-2: $max\_left\_x > min\_right\_y$

$8 > 19$ ✗

We are still too much in left, move more right.

Pass-3

	0	1	2	3	4	5
x =	1	3	8	9	15	
	7	11	18	19	21	25

start = P_x + 1 = (4)

end = (4)

$P\_x = 4+4/2 = (4)$

$P\_y = \frac{5+6+1}{2} - 4$

$= (2)$

	left	right
x	1 3 8 9	15
y	7 11	18 19 21 25

COND-1 : $9 < 18$ ✓ $11 < 15$ ✓ FOUND!

✓

Hence median (odd # elem) = max (max_left_x , max_left_y)

$= \max(9, 11)$

$= 11$

CODE :

$O(\log(\min(x, y)))$

```
medianSortedArrays( x , y ):
    if len(x) > len(y):
        return medianSortedArrays(y, x)

    m = len(x)
    n = len(y)

    start, end = 0, 0
    // Binary Search
    while start <= end:
        Partition_X = (start + end) / 2
        partition_Y = (m+n+1) / 2 - Partition_X
```

// Partition_x = 0 if there is nothing on the left side

maxLeft_X = partition_X == 0 ? $-\infty$: x[partition_X-1]

minRight_X = partition_X == m ? ∞ : x[partion_X]

maxLeft_Y = y[partition_Y-1]

minRight_Y = y[partition_Y].

COND-1

IF maxLeft_X $\leq$ minRight_Y

AND maxLeft_Y $\leq$ minRight_X :

IF Even :

return avg. (max(maxleftX, maxleftY),
min (minleftX, minleftY))

ELSE ODD :

return max(maxLeft_X, maxLeft_Y)

COND-2

ELSE IF maxLeft_X > minRight_Y :

end = partition_X - 1 // Go left

ELSE :

start = partition_X + 1 // Go right