MEDIAN OF TWO SORTED ARRAYS

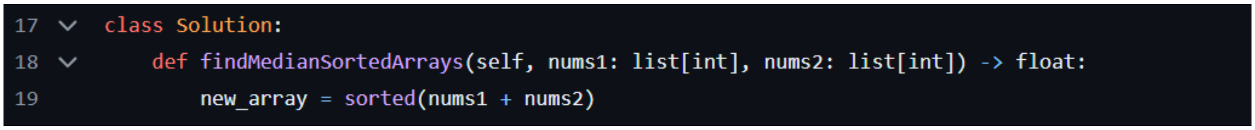
In this problem, we'll be given two different sorted arrays named **nums1** and **nums2**. We have to find the median of the final list which results in merging **nums1** and **nums2**.

Logic-

1. First, we will merge these two lists and sort them using *sorted(list)* method. Sorting step is necessary cause pre-sorted lists are sorted according to their independent nature to each other.
2. The idea of median is the middle point, element(s) which will divide the list into two halves.
3. Now, if the number of elements in a list are odd i.e. [1, 2, 3, 4, 5] then the element dividing the list in two halves is the median i.e. 3
4. But what if the number of elements in a list are even i.e. [1,2,3,4,5,6] then there will be two elements which will be dividing the list in two halves i.e. 3 and 4. So, the median in the case of two elements will be (3+4)/2.
5. *But we should never take more than 2 elements for median cause the whole idea of breaking into two parts with a point will be broken, it's like we are widening the width of the point.*

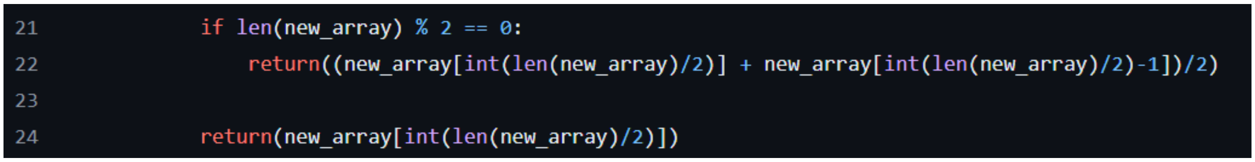
Let's look at the code for more understanding.

Code-



```
17  class Solution:
18      def findMedianSortedArrays(self, nums1: list[int], nums2: list[int]) -> float:
19          new_array = sorted(nums1 + nums2)
```

- We will begin by creating our class and function in line 17 and 18. The function will take two parameters – nums1 and nums2 as lists.
- Then we will merge the two lists, sort them and store them in a variable called new_array in line 19.



```
21          if len(new_array) % 2 == 0:
22              return((new_array[int(len(new_array)/2)] + new_array[int(len(new_array)/2)-1])/2)
23
24          return(new_array[int(len(new_array)/2)])
```

- Now in line 21, we will check if the length of list is even or odd i.e. the number of elements are even or odd.
- If the length is even, then we will pick our two elements from new_array.
- In line 22, we will take the i^{th} element which is at the index of length/2. And we will add this element with its one step earlier element. After adding them, we'll divide the result by 2 and return it.
- But if the length is not even, then we will simply return the element which is at the index of half of the length of new_array in line 24.

Time complexity analysis-

1. Line-18: Time complexity for concatenating the arrays is O(m+n) where m and n are the length of arrays respectively.
   The time complexity of sorting the concatenated array is O(nlogn).
2. Line 21 to 24: Checking the condition, accessing the array elements and performing arithmetic operations will take constant time i.e. O(1).

So, the overall time complexity of this algorithm is *O(nlogn)*.