LeetCode 4

https://leetcode.com/problems/median-of-two-sorted-arrays/description/

Yifeng Zeng

Description

4. Median of Two Sorted Arrays

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 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

Idea Report

Since both input arrays are sorted and the time complexity is required to be O(log(m+n)), we can utilize the method of binary search, basically we want to throw out about half of the impossible candidates at a time. To get the median, we can just find the kth element in the two input arrays,

the k may depends on the length of the two input arrays (the len / 2 + 1 small). So this problem is reduced to a problem that we want to find the kth element in two sorted array. To find the kth element, we can compare the k/2-th element in each array (n1 and n2), it is safe to throw away the smaller element and the elements before it because it's garanteed that the result will not be there. Then we can recurcively search from that index, and since we throw away k/2 numbers, we only need to look for another k - k/2 numbers. Until k == 1, we found the result.

Code:

```
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int len = nums1.length + nums2.length;
        if (len % 2 == 1) {
            return findKth(nums1, nums2, 0, 0, len / 2 + 1);
        }
        return (findKth(nums1, nums2, 0, 0, len / 2)
                + findKth(nums1, nums2, 0, 0, len / 2 + 1)) / 2.0;
    }

    private int findKth(int[] nums1, int[] nums2, int i, int j, int k) {
        if (i >= nums1.length) {
            return nums2[j + k - 1];
        }
        if (j >= nums2.length) {
            return nums1[i + k - 1];
        }

        if (k == 1) {
            return Math.min(nums1[i], nums2[j]);
        }

        int n1 = i + k / 2 - 1 >= nums1.length ?
                 Integer.MAX_VALUE : nums1[i + k / 2 - 1];
        int n2 = j + k / 2 - 1 >= nums2.length ?
                 Integer.MAX_VALUE : nums2[j + k / 2 - 1];
        if (n1 < n2) {
            return findKth(nums1, nums2, i + k / 2, j, k - k / 2);
        }
        return findKth(nums1, nums2, i, j + k / 2, k - k / 2);
    }
}
```

Summary

- Throw away half of candidates at a time, binary seach methodology.