Maximum Distance in Arrays

Given `m` arrays, and each array is sorted in ascending order. Now you can pick up two integers from two different arrays (each array picks one) and calculate the distance. We define the distance between two integers `a` and `b` to be their absolute difference `|a-b|`. Your task is to find the maximum distance.

Example 1:

```
Input:
[[1,2,3],
 [4,5],
 [1,2,3]]
Output: 4
Explanation:
One way to reach the maximum distance 4 is to pick 1 in the first or third array and
pick 5 in the second array.
```

Note:

1. Each given array will have at least 1 number. There will be at least two non-empty arrays.
2. The total number of the integers in **all** the `m` arrays will be in the range of [2, 10000].
3. The integers in the `m` arrays will be in the range of [-10000, 10000].

Solution 1

```
public class Solution {
    public int maxDistance(int[][] arrays) {
        int result = Integer.MIN_VALUE;
        int max = arrays[0][arrays[0].length - 1];
        int min = arrays[0][0];

        for (int i = 1; i < arrays.length; i++) {
            result = Math.max(result, Math.abs(arrays[i][0] - max));
            result = Math.max(result, Math.abs(arrays[i][arrays[i].length - 1] - mi
n));
            max = Math.max(max, arrays[i][arrays[i].length - 1]);
            min = Math.min(min, arrays[i][0]);
        }

        return result;
    }
}
```

LeetCode updated the input to List.

```
public class Solution {
    public int maxDistance(List<List<Integer>> arrays) {
        int result = Integer.MIN_VALUE;
        int max = arrays.get(0).get(arrays.get(0).size() - 1);
        int min = arrays.get(0).get(0);

        for (int i = 1; i < arrays.size(); i++) {
            result = Math.max(result, Math.abs(arrays.get(i).get(0) - max));
            result = Math.max(result, Math.abs(arrays.get(i).get(arrays.get(i).size(
) - 1) - min));
            max = Math.max(max, arrays.get(i).get(arrays.get(i).size() - 1));
            min = Math.min(min, arrays.get(i).get(0));
        }

        return result;
    }
}
```

written by shawngao original link here

Solution 2

```
public int maxDistance(int[][] arrays) {
  if (arrays == null || arrays.length == 0) return 0;

  int diff = Integer.MIN_VALUE;
  int m = arrays.length;

  int min =  arrays[0][0], max = arrays[0][arrays[0].length-1];
  for (int i = 1; i < m; i++) {
   int head = arrays[i][0];
   int tail = arrays[i][arrays[i].length-1];
   diff = Math.max(Math.abs(max-head), diff);
                       diff = Math.max(Math.abs(tail-min), diff);
   max = Math.max(tail, max);
   min = Math.min(head, min);
  }

  return diff;
    }
```

written by xmztzt original link here

Solution 3

The max distance must exist in one of these three differences, 1. difference between maximum and minimum, or 2. difference between 2nd maximum(the num that is only smaller than the maximum) or minimum, or 3. difference between maximum or 2nd minimum(the num that is only bigger than the minimum). Therefore, we just need to find these numbers.
Beside the link below is a more efficient solution by shawngao
https://discuss.leetcode.com/topic/92859/java-solution-min-and-max

```
public class Solution {
    public int maxDistance(int[][] a) {
        int min = Integer.MAX_VALUE; // ma](link url)x
        int max = Integer.MIN_VALUE; // min
        int k = 0, m =0;
        for(int i =0;i<a.length;i++){
            if(a[i][0]<min){
                min = a[i][0];
                k = i;
            }
            if(a[i][a[i].length-1]>max){
                max = a[i][a[i].length-1];
                m = i;
            }
        }
        if(k!=m) return max - min; // if max and min not in same array then return d
iff
        int ndMin = Integer.MAX_VALUE; // 2nd min
        for(int i = 0;i<a.length;i++){
            if(i==k) continue; // exclude the array with min
            ndMin = Math.min(ndMin,a[i][0]);
        }
        int ndMax = Integer.MIN_VALUE; // 2nd max
        for(int i = 0;i<a.length;i++){
            if(i==m) continue; // exclude the array with max
            ndMax = Math.max(ndMax,a[i][a[i].length-1]);
        }
        return Math.max(max - ndMin,ndMax - min);
    }
}
```

written by wxl163530 original link here