

MedianOfTwoSortedArray.java

```
1  package BinarySearch;
2
3  import java.util.Arrays;
4
5  public class MedianOfTwoSortedArray {
6      //Brute-Force
7      public static double medianBruteForce(int[] nums1, int[] nums2) {
8          // Get the sizes of both input arrays.
9          int n = nums1.length;
10         int m = nums2.length;
11
12         // Merge the arrays into a single sorted array.
13         int[] merged = new int[n + m];
14         int k = 0;
15         for (int i = 0; i < n; i++) {
16             merged[k++] = nums1[i];
17         }
18         for (int i = 0; i < m; i++) {
19             merged[k++] = nums2[i];
20         }
21
22         // Sort the merged array.
23         Arrays.sort(merged);
24
25         // Calculate the total number of elements in the merged array.
26         int total = merged.length;
27
28         if (total % 2 == 1) {
29             // If the total number of elements is odd, return the middle element as the median.
30             return (double) merged[total / 2];
31         } else {
32             // If the total number of elements is even, calculate the average of the two middle elements
33             int middle1 = merged[total / 2 - 1];
34             int middle2 = merged[total / 2];
35             return ((double) middle1 + (double) middle2) / 2.0;
36         }
37     }
38
39     //Better-Approach
40     public static double medianBetter(int[] nums1, int[] nums2) {
41         int n = nums1.length;
42         int m = nums2.length;
43         int i = 0, j = 0, m1 = 0, m2 = 0;
44
45         // Find median.
46         for (int count = 0; count <= (n + m) / 2; count++) {
47             m2 = m1;
48             if (i != n && j != m) {
49                 if (nums1[i] > nums2[j]) {
50                     m1 = nums2[j++];
51                 } else {
52                     m1 = nums1[i++];
53                 }
54             } else if (i < n) {
55                 m1 = nums1[i++];
56             } else {
57                 m1 = nums2[j++];
58             }
59         }
60
61         // Check if the sum of n and m is odd.
62         if ((n + m) % 2 == 1) {
63             return (double) m1;
64         } else {
65             double ans = (double) m1 + (double) m2;
66             return ans / 2.0;
67         }
68     }
69
70     //Optimal-Approach
71     public static double medianOptimal(int[] nums1, int[] nums2) {
72         int n1 = nums1.length, n2 = nums2.length;
73
74         // Ensure nums1 is the smaller array for simplicity
75         if (n1 > n2)
```

```

75 1      return medianOptimal(nums2, nums1);
76
77 1      int n = n1 + n2;
78 3      int left = (n1 + n2 + 1) / 2; // Calculate the left partition size
79      int low = 0, high = n1;
80
81 2      while (low <= high) {
82 2          int mid1 = (low + high) >> 1; // Calculate mid index for nums1
83 1          int mid2 = left - mid1; // Calculate mid index for nums2
84
85          int l1 = Integer.MIN_VALUE, l2 = Integer.MIN_VALUE, r1 = Integer.MAX_VALUE, r2 = Integer.MAX
86
87          // Determine values of l1, l2, r1, and r2
88 2          if (mid1 < n1)
89              r1 = nums1[mid1];
90 2          if (mid2 < n2)
91              r2 = nums2[mid2];
92 3          if (mid1 - 1 >= 0)
93 1              l1 = nums1[mid1 - 1];
94 3          if (mid2 - 1 >= 0)
95 1              l2 = nums2[mid2 - 1];
96
97 4          if (l1 <= r2 && l2 <= r1) {
98              // The partition is correct, we found the median
99 2              if (n % 2 == 1)
100 1                  return Math.max(l1, l2);
101              else
102 3                  return ((double)(Math.max(l1, l2) + Math.min(r1, r2))) / 2.0;
103          }
104 2          else if (l1 > r2) {
105              // Move towards the left side of nums1
106 1              high = mid1 - 1;
107          }
108          else {
109              // Move towards the right side of nums1
110 1              low = mid1 + 1;
111          }
112      }
113
114      return 0; // If the code reaches here, the input arrays were not sorted.
115  }
116 }

```

Mutations

```

13 1. Replaced integer addition with subtraction → KILLED
15 1. negated conditional → KILLED
   2. changed conditional boundary → KILLED
16 1. Changed increment from 1 to -1 → KILLED
18 1. changed conditional boundary → KILLED
   2. negated conditional → KILLED
19 1. Changed increment from 1 to -1 → KILLED
23 1. removed call to java/util/Arrays::sort → KILLED
28 1. negated conditional → KILLED
   2. Replaced integer modulus with multiplication → KILLED
30 1. replaced double return with 0.0d for BinarySearch/MedianOfTwoSortedArray::medianBruteForce → KILLED
   2. Replaced integer division with multiplication → KILLED
33 1. Replaced integer division with multiplication → KILLED
   2. Replaced integer subtraction with addition → KILLED
34 1. Replaced integer division with multiplication → KILLED
35 1. replaced double return with 0.0d for BinarySearch/MedianOfTwoSortedArray::medianBruteForce → KILLED
   2. Replaced double division with multiplication → KILLED
   3. Replaced double addition with subtraction → KILLED
46 1. Replaced integer division with multiplication → KILLED
   2. Replaced integer addition with subtraction → KILLED
   3. negated conditional → KILLED
   4. changed conditional boundary → KILLED
48 1. negated conditional → KILLED
   2. negated conditional → KILLED
49 1. negated conditional → KILLED
   2. changed conditional boundary → SURVIVED
50 1. Changed increment from 1 to -1 → KILLED
52 1. Changed increment from 1 to -1 → KILLED
54 1. changed conditional boundary → NO_COVERAGE
   2. negated conditional → NO_COVERAGE
55 1. Changed increment from 1 to -1 → NO_COVERAGE
57 1. Changed increment from 1 to -1 → NO_COVERAGE
62 1. negated conditional → KILLED
   2. Replaced integer addition with subtraction → KILLED
   3. Replaced integer modulus with multiplication → KILLED
63 1. replaced double return with 0.0d for BinarySearch/MedianOfTwoSortedArray::medianBetter → KILLED

```

65	1. Replaced double addition with subtraction → KILLED
66	1. Replaced double division with multiplication → KILLED 2. replaced double return with 0.0d for BinarySearch/MedianOfTwoSortedArray::medianBetter → KILLED
74	1. negated conditional → KILLED 2. changed conditional boundary → KILLED
75	1. replaced double return with 0.0d for BinarySearch/MedianOfTwoSortedArray::medianOptimal → NO_COVERAGE
77	1. Replaced integer addition with subtraction → KILLED
78	1. Replaced integer addition with subtraction → KILLED 2. Replaced integer division with multiplication → KILLED 3. Replaced integer addition with subtraction → KILLED
81	1. negated conditional → KILLED 2. changed conditional boundary → SURVIVED
82	1. Replaced integer addition with subtraction → KILLED 2. Replaced Shift Right with Shift Left → KILLED
83	1. Replaced integer subtraction with addition → KILLED
88	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED
90	1. changed conditional boundary → SURVIVED 2. negated conditional → SURVIVED
92	1. changed conditional boundary → SURVIVED 2. negated conditional → SURVIVED 3. Replaced integer subtraction with addition → SURVIVED
93	1. Replaced integer subtraction with addition → KILLED
94	1. negated conditional → KILLED 2. Replaced integer subtraction with addition → SURVIVED 3. changed conditional boundary → SURVIVED
95	1. Replaced integer subtraction with addition → KILLED
97	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED 3. changed conditional boundary → SURVIVED 4. negated conditional → KILLED
99	1. negated conditional → KILLED 2. Replaced integer modulus with multiplication → KILLED
100	1. replaced double return with 0.0d for BinarySearch/MedianOfTwoSortedArray::medianOptimal → KILLED
102	1. replaced double return with 0.0d for BinarySearch/MedianOfTwoSortedArray::medianOptimal → KILLED 2. Replaced double division with multiplication → KILLED 3. Replaced integer addition with subtraction → KILLED
104	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED
106	1. Replaced integer subtraction with addition → NO_COVERAGE
110	1. Replaced integer addition with subtraction → TIMED_OUT

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- BinarySearch.MedianOfTwoSortedArrayTest.testMedianBetter_OddLengthArrays(BinarySearch.MedianOfTwoSortedArrayTest) (0 ms)
- BinarySearch.MedianOfTwoSortedArrayTest.testMedianBetter_EvenLengthArrays(BinarySearch.MedianOfTwoSortedArrayTest) (0 ms)
- BinarySearch.MedianOfTwoSortedArrayTest.testMedianBruteForce_OddLengthArrays(BinarySearch.MedianOfTwoSortedArrayTest) (0 ms)
- BinarySearch.MedianOfTwoSortedArrayTest.testMedianBruteForce_EvenLengthArrays(BinarySearch.MedianOfTwoSortedArrayTest) (0 ms)
- BinarySearch.MedianOfTwoSortedArrayTest.testMedianOptimal_EvenLengthArrays(BinarySearch.MedianOfTwoSortedArrayTest) (0 ms)
- BinarySearch.MedianOfTwoSortedArrayTest.testMedianOptimal_OddLengthArrays(BinarySearch.MedianOfTwoSortedArrayTest) (0 ms)

Report generated by [PIT](#) 1.15.0