Articles → 280. Wiggle Sort ▼

6 0 0

280. Wiggle Sort 2

**** March 5, 2016 | 29.6K views Average Rating: 4.44 (27 votes)

Given an unsorted array nums , reorder it in-place such that nums[0] <= nums[1] >= nums[2] <= nums[3]....

Example:

```
Input: nums = [3,5,2,1,6,4]
Output: One possible answer is [3,5,1,6,2,4]
```

Solution

Approach #1 (Sorting) [Accepted]

The obvious solution is to just sort the array first, then swap elements pair-wise starting from the second element. For example:

```
[1, 2, 3, 4, 5, 6]
     1 1 1 1
     swap swap
=> [1, 3, 2, 5, 4, 6]
```

```
public void wiggleSort(int[] nums) {
   Arrays.sort(nums);
   for (int i = 1; i < nums.length - 1; i += 2) {
        swap(nums, i, i + 1);
private void swap(int[] nums, int i, int j) {
   int temp = nums[i];
   nums[i] = nums[j];
   nums[j] = temp;
```

Complexity analysis

- Time complexity : $O(n \log n)$. The entire algorithm is dominated by the sorting step, which costs $O(n \log n)$ time to sort n elements.
- Space complexity : O(1). Space depends on the sorting implementation which, usually, costs O(1)auxiliary space if heapsort is used.

Approach #2 (One-pass Swap) [Accepted]

Intuitively, we should be able to reorder it in one-pass. As we iterate through the array, we compare the current element to its next element and if the order is incorrect, we swap them.

```
public void wiggleSort(int[] nums) {
    boolean less = true;
    for (int i = 0; i < nums.length - 1; i++) {
        if (less) {
            if (nums[i] > nums[i + 1]) {
                swap(nums, i, i + 1);
        } else {
            if (nums[i] < nums[i + 1]) {</pre>
                swap(nums, i, i + 1);
        less = !less;
    }
}
```

We could shorten the code further by compacting the condition to a single line. Also observe the boolean value of less actually depends on whether the index is even or odd.

```
public void wiggleSort(int[] nums) {
    for (int i = 0; i < nums.length - 1; i++) {
        if (((i \% 2 == 0) \&\& nums[i] > nums[i + 1])
                || ((i \% 2 == 1) \&\& nums[i] < nums[i + 1])) {
            swap(nums, i, i + 1);
   }
}
```

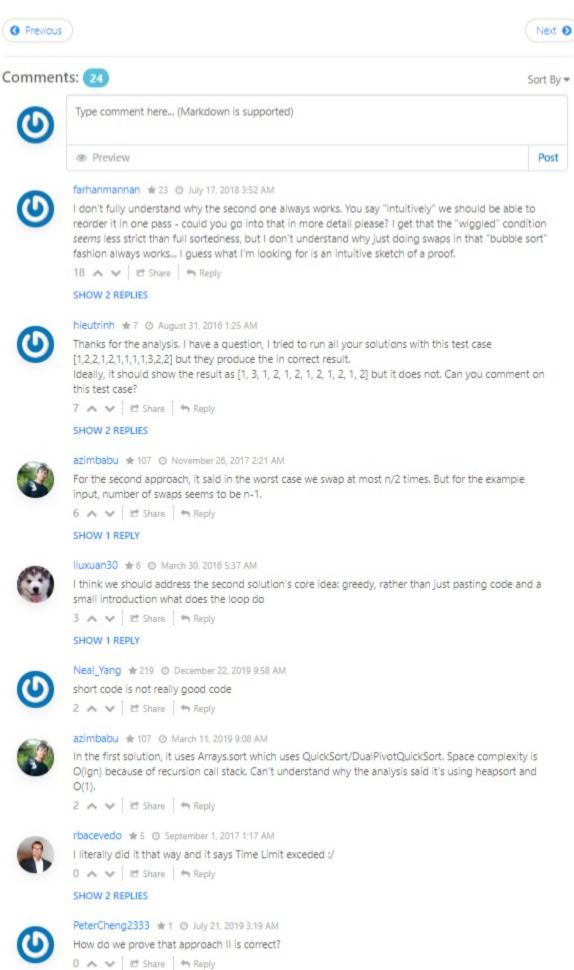
Here is another amazing solution by @StefanPochmann who came up with originally here.

```
public void wiggleSort(int[] nums) {
   for (int i = 0; i < nums.length - 1; i++) {
       if ((i \% 2 == 0) == (nums[i] > nums[i + 1])) {
           swap(nums, i, i + 1);
```

Complexity analysis

- Time complexity : O(n). In the worst case we swap at most $\frac{n}{2}$ times. An example input is [2,1,3,1,4,1].
- Space complexity: O(1).

Rate this article: ** ** ** **



0 ∧ ∨ ≥ Share ↑ Reply

sorting-algorithms-for-diff

Correct me if I'm wrong...

SHOW 1 REPLY

SHOW 1 REPLY

powerrc # 12 @ June 1, 2019 2:53 AM

Javava 🛊 2 💿 May 31, 2019 11:03 AM

0 ∧ ∨ Et Share ★ Reply

Does sorting first really count as "reorder in-place" during interview?

For the 1st solution, I don't think java is using heapsort. According to this

java 7 is using TimSort and Dual-pivot QuickSort. These are not O(1) space algorithms.

https://stackoverflow.com/questions/3707190/why-does-javas-arrays-sort-method-use-two-different-

Read More