280. Wiggle Sort

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]
       ↑  ↑  ↑  ↑
       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]) {
                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: ★★★★★

Previous Next

Comments: 24 Sort By

Type comment here... (Markdown is supported)

Preview Post

farhanmannan ★ 23 July 17, 2018 3:52 AM
I don't fully understand why the second one always works. You say "intuitively" we should be able to reorder it in one pass - could you go into that in more detail please? I get that the "wiggled" condition *seems* less strict than full sortedness, but I don't understand why just doing swaps in that "bubble sort" fashion always works... I guess what I'm looking for is an intuitive sketch of a proof.
18 Share Reply
SHOW 2 REPLIES

hieutrinh ★ 7 August 31, 2016 1:25 AM
Thanks for the analysis. I have a question, I tried to run all your solutions with this test case [1,2,2,1,2,1,1,1,1,3,2,2] but they produce the in correct result.
Ideally, it should show the result as [1, 3, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2] but it does not. Can you comment on this test case?
7 Share Reply
SHOW 2 REPLIES

azimbabu ★ 107 November 26, 2017 2:21 AM
For the second approach, it said in the worst case we swap at most n/2 times. But for the example input, number of swaps seems to be n-1.
6 Share Reply
SHOW 1 REPLY

liuxuan30 ★ 6 March 30, 2016 5:37 AM
I think we should address the second solution's core idea: greedy, rather than just pasting code and a small introduction what does the loop do
3 Share Reply
SHOW 1 REPLY

Neal_Yang ★ 219 December 22, 2019 9:58 AM
short code is not really good code
2 Share Reply

azimbabu ★ 107 March 11, 2019 9:08 AM
In the first solution, it uses Arrays.sort which uses QuickSort/DualPivotQuickSort. Space complexity is O(lgn) because of recursion call stack. Can't understand why the analysis said it's using heapsort and O(1).
2 Share Reply

rbacevedo ★ 5 September 1, 2017 1:17 AM
I literally did it that way and it says Time Limit exceded :/
0 Share Reply
SHOW 2 REPLIES

PeterCheng2333 ★ 1 July 21, 2019 3:19 AM
How do we prove that approach II is correct?
0 Share Reply
SHOW 1 REPLY

powerrc ★ 12 June 1, 2019 2:53 AM
Does sorting first really count as "reorder in-place" during interview?
0 Share Reply
SHOW 1 REPLY

Javava ★ 2 May 31, 2019 11:03 AM
For the 1st solution, I don't think java is using heapsort. According to this https://stackoverflow.com/questions/3707190/why-does-javas-arrays-sort-method-use-two-different-sorting-algorithms-for-diff
java 7 is using TimSort and Dual-pivot QuickSort. These are not O(1) space algorithms.
Correct me if I'm wrong...
Read More
0 Share Reply

‹ 1 2 3 ›