C++

Autocomplete

```
class Solution {
public:
    void wiggleSort(vector<int>&
nums) {

    }
};
```

Console Contribute

Quick Navigation

★★★★☆ Average Rating: 4.16 (45 votes)

Premium

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 : $O(1)$.

Report Article Issue

Comments: 35 Best Most Votes Newest to Oldest Oldest to Newest

Type comment here... (Markdown is supported)

Post

farhanmannan ★ 58 Last Edit: September 7, 2018 12:09 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.

49 Show 2 replies Reply

liuxuan30 ★ 26 Last Edit: October 22, 2018 5:05 PM

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

17 Hide 2 replies Reply

venu_bondugula ★ 11 September 30, 2021 12:27 PM

@1337c0d3r But you did not do.

0 Reply

1337c0d3r Admin ★ 3193 March 30, 2016 6:02 AM

Sure, I will revise the article to add more explanation. Please stay tuned.

1 Reply

Neal_Yang ★ 554 December 21, 2019 8:28 PM

short code is not really good code

30 Show 1 reply Reply

azimbabu ★ 238 Last Edit: October 21, 2018 10:28 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.

9 Show 1 reply Reply

hieutrinh ★ 11 Last Edit: September 23, 2018 1:38 PM

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?

8 Hide 2 replies Reply

alphaorc ★ 59 July 29, 2019 6:55 AM

What you're expecting is the case for Wiggle Sort II

0 Reply

ajuney ★ 2 December 27, 2018 3:12 PM

I think the output is still correct based on the question stating that nums[0] <= nums[1]

2 Reply

azimbabu ★ 238 March 10, 2019 8:38 PM

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).

4 Reply

PeterCheng2333 ★ 1 July 20, 2019 2:49 PM

How do we prove that approach II is correct?

0 Show 2 replies Reply

rbacevedo ★ 5 August 31, 2017 12:47 PM

I literally did it that way and it says Time Limit exceded :/

0 Show 2 replies Reply

powerrc ★ 14 May 31, 2019 2:23 PM

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

0 Hide 1 reply Reply

kevinhynes ★ 779 February 19, 2020 5:42 AM

Depends on whether or not you sort it in place. In python, its the difference between using the `sort()` method and the `sorted()` function.

`my_list.sort()` will just sort `my_list` in place.

`my_list = sorted(my_list)` will sort a copy of `my_list` and reassign it to `my_list`, ie *not* in place. This consumes extra space.

4 Reply

Javava ★ 4 May 30, 2019 10:33 PM

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...

0 Reply