

# Problem 324: Wiggle Sort II

## Problem Information

**Difficulty:** Medium

**Acceptance Rate:** 0.00%

**Paid Only:** No

## Problem Description

Given an integer array

nums

, reorder it such that

nums[0] < nums[1] > nums[2] < nums[3]...

You may assume the input array always has a valid answer.

Example 1:

Input:

nums = [1,5,1,1,6,4]

Output:

[1,6,1,5,1,4]

Explanation:

[1,4,1,5,1,6] is also accepted.

Example 2:

Input:

nums = [1,3,2,2,3,1]

Output:

[2,3,1,3,1,2]

Constraints:

$1 \leq \text{nums.length} \leq 5 * 10^4$

4

$0 \leq \text{nums}[i] \leq 5000$

It is guaranteed that there will be an answer for the given input

nums

.

Follow Up:

Can you do it in

$O(n)$

time and/or

in-place

with

$O(1)$

extra space?

## Code Snippets

### C++:

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

### Java:

```
class Solution {
    public void wiggleSort(int[] nums) {
        }
    }
}
```

### Python3:

```
class Solution:
    def wiggleSort(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """

```

### Python:

```
class Solution(object):
    def wiggleSort(self, nums):
        """
        :type nums: List[int]
        :rtype: None Do not return anything, modify nums in-place instead.
        """

```

### JavaScript:

```
/**
 * @param {number[]} nums
 * @return {void} Do not return anything, modify nums in-place instead.
```

```
*/  
var wiggleSort = function(nums) {  
  
};
```

### TypeScript:

```
/**  
Do not return anything, modify nums in-place instead.  
*/  
function wiggleSort(nums: number[]): void {  
  
};
```

### C#:

```
public class Solution {  
    public void WiggleSort(int[] nums) {  
  
    }  
}
```

### C:

```
void wiggleSort(int* nums, int numsSize) {  
  
}
```

### Go:

```
func wiggleSort(nums []int) {  
  
}
```

### Kotlin:

```
class Solution {  
    fun wiggleSort(nums: IntArray): Unit {  
  
    }  
}
```

**Swift:**

```
class Solution {  
    func wiggleSort(_ nums: inout [Int]) {  
  
    }  
}
```

**Rust:**

```
impl Solution {  
    pub fn wiggle_sort(nums: &mut Vec<i32>) {  
  
    }  
}
```

**Ruby:**

```
# @param {Integer[]} nums  
# @return {Void} Do not return anything, modify nums in-place instead.  
def wiggle_sort(nums)  
  
end
```

**PHP:**

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @return NULL  
     */  
    function wiggleSort(&$nums) {  
  
    }  
}
```

**Dart:**

```
class Solution {  
    void wiggleSort(List<int> nums) {  
  
    }
```

```
}
```

### Scala:

```
object Solution {  
    def wiggleSort(nums: Array[Int]): Unit = {  
        }  
    }  
}
```

## Solutions

### C++ Solution:

```
/*  
 * Problem: Wiggle Sort II  
 * Difficulty: Medium  
 * Tags: array, greedy, sort  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(1) to O(n) depending on approach  
 */  
  
class Solution {  
public:  
    void wiggleSort(vector<int>& nums) {  
  
    }  
};
```

### Java Solution:

```
/**  
 * Problem: Wiggle Sort II  
 * Difficulty: Medium  
 * Tags: array, greedy, sort  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)
```

```

* Space Complexity: O(1) to O(n) depending on approach
*/



class Solution {
    public void wiggleSort(int[] nums) {

    }
}

```

### Python3 Solution:

```

"""
Problem: Wiggle Sort II
Difficulty: Medium
Tags: array, greedy, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:

    def wiggleSort(self, nums: List[int]) -> None:
        # TODO: Implement optimized solution
        pass

```

### Python Solution:

```

class Solution(object):
    def wiggleSort(self, nums):
        """
:type nums: List[int]
:rtype: None Do not return anything, modify nums in-place instead.
"""

```

### JavaScript Solution:

```

/**
 * Problem: Wiggle Sort II
 * Difficulty: Medium
 * Tags: array, greedy, sort

```

```

/*
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

/**
 * @param {number[]} nums
 * @return {void} Do not return anything, modify nums in-place instead.
 */
var wiggleSort = function(nums) {

};


```

### TypeScript Solution:

```

/** 
 * Problem: Wiggle Sort II
 * Difficulty: Medium
 * Tags: array, greedy, sort
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

/**
Do not return anything, modify nums in-place instead.
*/
function wiggleSort(nums: number[]): void {

};


```

### C# Solution:

```

/*
 * Problem: Wiggle Sort II
 * Difficulty: Medium
 * Tags: array, greedy, sort
 *
 * Approach: Use two pointers or sliding window technique

```

```

 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

public class Solution {
    public void WiggleSort(int[] nums) {

    }
}

```

### C Solution:

```

/*
 * Problem: Wiggle Sort II
 * Difficulty: Medium
 * Tags: array, greedy, sort
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

void wiggleSort(int* nums, int numsSize) {

}

```

### Go Solution:

```

// Problem: Wiggle Sort II
// Difficulty: Medium
// Tags: array, greedy, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func wiggleSort(nums []int) {
}

```

### Kotlin Solution:

```
class Solution {  
    fun wiggleSort(nums: IntArray): Unit {  
        }  
    }  
}
```

### Swift Solution:

```
class Solution {  
    func wiggleSort(_ nums: inout [Int]) {  
        }  
    }  
}
```

### Rust Solution:

```
// Problem: Wiggle Sort II  
// Difficulty: Medium  
// Tags: array, greedy, sort  
//  
// Approach: Use two pointers or sliding window technique  
// Time Complexity: O(n) or O(n log n)  
// Space Complexity: O(1) to O(n) depending on approach  
  
impl Solution {  
    pub fn wiggle_sort(nums: &mut Vec<i32>) {  
        }  
    }  
}
```

### Ruby Solution:

```
# @param {Integer[]} nums  
# @return {Void} Do not return anything, modify nums in-place instead.  
def wiggle_sort(nums)  
  
end
```

### PHP Solution:

```
class Solution {
```

```
/**  
 * @param Integer[] $nums  
 * @return NULL  
 */  
function wiggleSort(&$nums) {  
  
}  
}
```

### Dart Solution:

```
class Solution {  
void wiggleSort(List<int> nums) {  
  
}  
}
```

### Scala Solution:

```
object Solution {  
def wiggleSort(nums: Array[Int]): Unit = {  
  
}  
}
```