**Non-Recursive algorithm of Wiggle-Sort**

**Algorithm Logic:**

void wiggleSort1(int a[], int size) {

for(int i = 0; i < size - 1; i++) {

if ((i % 2 == 1 && a[i] <= a[i+1]) ||

(i % 2 == 0 && a[i] >= a[i+1])) {

swap(&a[i], &a[i+1]);

}

}

}

**Pseudocode:**

1. **Wigglesort(array,size){**
2. For i ← 0 to size – 1 step 1
3. Do {
4. If (i is even AND array[i] > array[i+1]) OR

(i is odd AND array[i] < array[i+1])

5. then **swap**(array[i], array[i+1]) } }

Main:

6.Read size of array

7.Read elements into array

8.write array before sorting

9. wiggleSort(array, size)

10.write array after wiggle sort

1.**swap**(x, y):  
2. temp ← x  
3. x ← y  
4. y ← temp

**Analysis**

1. **Algorithm Explanation:**

It is a **Greedy One**-**Pass Algorithm** thatachieves the wiggle sort pattern**:**

* **Even indices (i % 2 = = 0): enforce a[i] < a[i+1]**
* **Odd indices (i % 2 = = 1): enforce a[i] > a[i+1]**

If the condition isn’t achieved, the elements are swapped.

**Control Structures used:**

* **Sequencing:**

The steps of loops, conditions and swap are executed one after another in order.

That’s how sequencing is applied in our algorithm implementation:

1. Check the index parity (i % 2)
2. Compare values a[i] and a[i+1]
3. Evaluate the condition
4. If true, call swap
5. Repeat for the next i

* **if-then-else:**

Used to check whether to swap elements based on the index and value conditions:   
**if ((i%2 = =1 && a[i]<=a[i+1]) ||**

**(i%2 = =0 && a[i]>=a[i+1]))**

* **For loop:**

The main iteration is a loop from i = 0 to i = size-2 (i < size-1)

**for (int i = 0; i < size - 1; i++)**

1. **Time Complexity:**

Time complexity for best, average and worst cases is linear complexity **O(n).**

Reasons:

* The for loop runs size-1 times O(n)
* Each iteration does O (1) work which is at most one swap

So, the overall time complexity is **O(n)**

1. **Space Complexity:**

Our wiggle sort non-recursive algorithm doesn’t have arrays created inside wiggleSort1(), dynamic memory allocation (malloc, etc.), recursion or stack-heavy operations.

It has input size of n but wiggleSort1() doesn’t use any space proportional to n It only uses a constant amount of space,so

**space complexity is O(1).**

1. **Constraints:**

**Constraints applied in our non-recursive algorithm are:**

1. **1 <= nums.length <= 5 \* 104**

Which means that the length or size of the input array should be at least 1 and shouldn’t exceed 50,000 elements, this is implemented in our algorithm code in these lines:

**int a[50000];**

**printf("Enter the size of array\n");**

**int size;**

**scanf("%d", &size);**

**if (size < 1 || size > 50000) {**

**printf("Invalid size, size must be between 1 and 50000.\n");**

**return 1;**

**}**

1. **0 <= nums[i] <= 5000**

This constraint means that the value of each element in the array can be any number from 0 to 5000, this is implemented in our algorithm code using this condition:

**if (a[i] < 0 || a[i] > 5000) {**

**printf("Invalid element, value of elements must be between 0 and 5000.\n");**

The function executes efficiently on all valid inputs, so it is guaranteed that there will be valid answer for any input numbers under these limits.