**Recursive algorithm of Wiggle-Sort**

**Algorithm logic:**

void wiggleSort(int\* nums, int numsSize) {

mergeSort(nums, 0, numsSize- 1);

int half = (numsSize + 1) / 2;

int\* small = ...; // first half reversed

int\* large = ...; // second half reversed

int i = 0, s = 0, l = 0;

while (i < numsSize) {

if (i % 2 == 0) nums[i++] = small[s++];

else nums[i++] = large[l++];

}

}

**Pseudocode:**

sort(nums)

split nums into small (first half) and large (second half)

reverse small

reverse large

for i from 0 to n-1:

if i is even: nums[i] = small[next]

else:

nums[i] = large[next]

**1)Algorithm Explanation:**

It is a **sort and interleave algorithm** thatachieves the wiggle sort pattern**:**

-It sorts the array

- it split the array into two halves and reverse each

- Interleave largest of smaller half and largest of larger half

**Control Structures used:**

**While loop:**

-To reconstruct the array in a wiggle sort pattern.

* i goes from 0 to numsSize - 1.
* At each step, it decides **which element** to place at position i:
  + Even indices (0, 2, 4, ...) get elements from the **smaller half** (small[]).
  + Odd indices (1, 3, 5, ...) get elements from the **larger half** (large[]).

**If-Then-Else:**

-To alternate between inserting smaller and larger numbers.

* i % 2 == 0 means the current position is even → insert from small[].
* i % 2 == 1 means the current position is odd → insert from large[].

**2) Time Complexity**

Time complexity for best, average and worst case is **O(nlogn)**

**Reasons:**

* **Dividing** the array in half recursively → this takes **log n** levels.
* At each level, it **merges** all elements (which takes O(n) total at each level).

**3) Space Complexity:**

Space Complexity is **O(n)** since the wiggle sort is recursive so at each merge step, temporary arrays are allocated.

**4) Constraints:**

**1 ≤ numsSize ≤ 50000**

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");**