Algorithm Task

Non-Recursive Wiggle-Sort Algorithm

Algorithm Logic (C code):

Non-Recursive Wiggle-Sort Code

```
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

Non-Recursive Wiggle-Sort Pseudocode

- For i from 0 to $n-2$:
  - If (i is odd and $a[i] \leq a[i+1]$) OR (i is even and $a[i] \geq a[i+1]$):
    - Swap $a[i]$ and $a[i+1]$

Analysis

- **Algorithm Explanation:**
  - Greedy, single-pass algorithm to enforce **wiggle sort**:
    - Even i: $a[i] < a[i+1]$
    - Odd i: $a[i] > a[i+1]$
  - Uses sequencing, if-then-else, and a for loop.
- **Time Complexity:** $O(n)$ (one pass, each step $O(1)$)
- **Space Complexity:** $O(1)$ (no extra space)

- **Constraints:**
  - $1 \leq n \leq 50,000$
  - $0 \leq a[i] \leq 5,000$

Recursive Wiggle-Sort Algorithm

Algorithm Logic (C code):

Recursive Wiggle-Sort Code

```
// Assumes mergeSort function is defined
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

Recursive Wiggle-Sort 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] = next from small
  - Else: nums[i] = next from large

Analysis

- **Algorithm Explanation:**

  - Sorts array, splits into halves, reverses each, and interleaves.
  - Uses while loop and if-then-else for alternating insertion.

- **Time Complexity:** $O(n \log n)$ (sorting dominates)
- **Space Complexity:** $O(n)$ (temporary arrays for halves)
- **Constraints:**
  - $1 \leq n \leq 50,000$
  - $0 \leq a[i] \leq 5,000$

Comparison of Approaches

Summary Comparison

- The **non-recursive** method is faster ($O(n)$), simpler, and uses constant space, making it well-suited for most practical cases.
- The **recursive (sort-based)** method is more general and works well with duplicate values, but is slower ($O(n \log n)$) and requires extra space.