To derive the recurrence relation $T(n)$ for the runtime complexity of WeirdSort, let's break down how the algorithm operates:

Steps of WeirdSort

1. If the size of the sublist is exactly 2, the algorithm makes a constant-time comparison and possibly a swap, which is $O(1)$.
2. If the size of the sublist is larger than 2, the list is divided into three overlapping sublists:
    - **First sublist:** from index start to end$-k$ (with length approximately $\frac{2n}{3}$).
    - **Second sublist:** from index start $+ k$ to end (with length approximately $\frac{2n}{3}$).
    - **Third sublist:** from index start to end $- k$ (with length approximately $\frac{2n}{3}$).

   Where $k = \left(\frac{\text{end}-\text{start}}{3}\right)$ is one-third of the length of the current sublist.

Recursive Structure

In each recursive step, WeirdSort makes three recursive calls, each on a sublist of size approximately $\frac{2n}{3}$. This results in the following recurrence relation for the runtime:

$$T(n) = 3T\left(\frac{2n}{3}\right) + O(1)$$

- $3T\left(\frac{2n}{3}\right)$ accounts for the three recursive calls on sublists of size $\frac{2n}{3}$.
- $O(1)$ is the constant time required for comparisons and swaps in the base case (when $n = 2$).

Solving the Recurrence

To solve this recurrence relation, we will use the **master theorem** for divide-and-conquer recurrences of the form:

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$$

where:

- $a = 3$ (the number of recursive calls),
- $b = \frac{3}{2}$ (the factor by which the sublist size is reduced),

- $d = 0$ (since the non-recursive work is constant).

Now we check the conditions of the master theorem:

1. **Calculate log base b of a:**

$$\log_b a = \log_{\frac{3}{2}} 3 = \frac{\log 3}{\log \frac{3}{2}} \approx 2.7095$$

2. **Compare d with $\log_b a$:**
   - $d = 0$
   - $\log_b a \approx 2.7095$

Since $d < \log_b a$, by the master theorem, the time complexity is determined by the recurrence growth, and we conclude:

$$T(n) = O(n^{\log_b a}) = O(n^{2.7095})$$

Conclusion

The time complexity of WeirdSort is approximately:

$$T(n) = O(n^{2.71})$$

This is super-polynomial, meaning WeirdSort is less efficient than typical efficient sorting algorithms such as **merge sort** or **quicksort**, which both have a time complexity of $O(n \log n)$.