Linearithmic Algorithm to Find Common Name in Three Lists

Problem Definition

You are given three lists, each containing N names. The task is to determine if there is a name common to all three lists. If a common name exists, return the first such name in sorted order. The algorithm should run in $O(N \log N)$ time, where N is the size of each list.

Approach

The goal is to devise an algorithm that runs in linearithmic time, i.e., $O(N \log N)$. We can achieve this by using sorting and a merge-like process, which ensures efficient comparison across the three lists.

Algorithm Steps

1. **Sort the three lists:** Sorting each list individually takes $O(N \log N)$ time. Sorting all three lists takes $3 \cdot O(N \log N) = O(N \log N)$.
2. **Initialize three pointers:** Start at the beginning of each sorted list with three pointers i, j, and k, one for each list.
3. **Use a merge-like process to compare elements:**
   - If $A[i] = B[j] = C[k]$, return the name as the common name.
   - If the names are not equal, increment the pointer for the list with the smallest current name.
4. **Continue until one list is exhausted:** If one of the pointers reaches the end of its list, no common name exists.

Pseudocode

```
function find_common_name(A, B, C):
    # Step 1: Sort the three lists
```

```
sort(A)   # O(N log N)
sort(B)   # O(N log N)
sort(C)   # O(N log N)

# Step 2: Initialize pointers
i = 0
j = 0
k = 0

# Step 3: Use a merge-like process to find common elements
while i < N and j < N and k < N:
    if A[i] == B[j] and B[j] == C[k]:
        return A[i]  # Return the first common name
    elif A[i] < B[j]:
        i += 1
    elif B[j] < C[k]:
        j += 1
    else:
        k += 1

# Step 4: If no common name is found
return None
```

Explanation

1. **Sorting the lists:** Sorting each list takes $O(N \log N)$ per list. Sorting all three lists results in a time complexity of $O(N \log N)$.
2. **Iterating through the lists:** After sorting, the algorithm iterates through the three lists simultaneously in $O(N)$, advancing the pointer of the list with the smallest element at each step. This ensures that all names are checked without unnecessary comparisons.

Time Complexity

- Sorting each list takes $O(N \log N)$.
- The merge-like iteration through the lists takes $O(N)$.

Thus, the total time complexity is $O(N \log N)$, which is linearithmic.

Example

Consider the following three lists of names:

$$A = [\text{"Anna"}, \text{"Bob"}, \text{"Charlie"}, \text{"David"}]$$

$$B = [\text{"Anna"}, \text{"Charlie"}, \text{"Eve"}, \text{"Frank"}]$$
$$C = [\text{"Charlie"}, \text{"George"}, \text{"Anna"}, \text{"David"}]$$

1. Sort the three lists:

$$A = [\text{"Anna"}, \text{"Bob"}, \text{"Charlie"}, \text{"David"}]$$
$$B = [\text{"Anna"}, \text{"Charlie"}, \text{"Eve"}, \text{"Frank"}]$$
$$C = [\text{"Anna"}, \text{"Charlie"}, \text{"David"}, \text{"George"}]$$

2. Initialize pointers $i = 0$, $j = 0$, and $k = 0$, and start comparing:

$$A[0] = \text{"Anna"}, B[0] = \text{"Anna"}, C[0] = \text{"Anna"}$$

3. All match, so return "Anna".

Conclusion

This approach ensures a time complexity of $O(N \log N)$ by sorting the three lists and using a merge-like process to find the first common name. The algorithm is efficient and straightforward for solving the problem of finding a common name in three lists.