Approach: Priority Queue Approach

Objective

To solve the "767. Reorganize String" problem, the priority queue approach leverages a max heap to maintain the frequency of each character. By doing so, we can alternate the most frequent characters with the remaining ones to ensure no adjacent characters are the same.

Key Data Structures

- **Max Heap**: Used for storing characters sorted by their frequency in descending order.

Enhanced Breakdown

1. **Initialization**:
    - Count the frequency of each character in the string.
    - Populate the max heap with these frequencies.
2. **Processing Each Character**:
    - Pop the top two characters from the max heap (i.e., the ones with the highest frequency).
    - Append these two characters to the result string.
    - Decrement their frequencies and re-insert them back into the max heap.
    - If only one character remains in the heap, make sure it doesn't exceed half of the string length, otherwise, return an empty string.
3. **Wrap-up**:
    - If there's a single remaining character with a frequency of 1, append it to the result.
    - Join all the characters to return the final reorganized string.

Example:

Given the input "aab":

- Max heap after initialization: [(-2, 'a'), (-1, 'b')]
- Result after first iteration: "ab"
- Wrap-up: Result is "aba"

Complexity:

Time Complexity:

1. **Priority Queue Approach:** $O(n \log k)$
    - O(n) for counting the frequency of each character in the string. Here, nn$n$ is the length of the string.
    - $O(k \log k)$ for building the max heap, where k is the number of unique characters in the string.
    - The heap operations (insertion and deletion) would require log$k$ time each. In the worst-case scenario, you would be doing these operations n times (once for each character in the string).