Problem 0

The function calls occur in the following order:

```
fib(5)
|-- fib(4)
|   |-- fib(3)
|   |   |-- fib(2)
|   |   |   |-- fib(1) -> returns 1
|   |   |   |-- fib(0) -> returns 0
|   |   |   |-- returns 1
|   |   |-- fib(1) -> returns 1
|   |   |-- returns 2
|   |-- fib(2)
|   |   |-- fib(1) -> returns 1
|   |   |-- fib(0) -> returns 0
|   |   |-- returns 1
|   |-- returns 3
|-- fib(3)
|   |-- fib(2)
|   |   |-- fib(1) -> returns 1
|   |   |-- fib(0) -> returns 0
|   |   |-- returns 1
|   |-- fib(1) -> returns 1
|   |-- returns 2
|-- returns 5
```

Problem 1:

Time Complexity Analysis

- Heap insertion and deletion take O(log K) time.
- Since there are K*N elements, the total complexity is: O(NK log K)
  which is optimal for this problem.

Potential Improvements

- If K is small, a simple merge approach using sorted() might work faster in practice.
- Using divide & conquer (merge two lists at a time) would achieve a complexity of O(NK log K) but can be more cache-friendly.

Problem 2:

Time Complexity Analysis

- O(N) time complexity as we traverse the array once.
- O(1) extra space since we modify the array in place.

Potential Improvements

- Two-Pointer Optimization: Instead of checking every element, detect long duplicate sequences and perform jumps to speed up traversal.
- Binary Search for Next Unique Element: If duplicates appear frequently, use binary search to skip duplicate sequences efficiently.