Explain how to implement a FIFO queue with two stacks:

1. Starting with two empty stacks. Mark them Stack 1 and Stack 2.
2. Whenever the user performs the $pushback$ operation, always add this new element to the top of the Stack 1, regardless if the Stack 2 is empty.
3. For the $popfornt$ operation, we classify as follows:
    a. If the Stack 2 is empty, then move all the elements in Stack 1 to Stack 2 in a reversed order and remove the element at the top of the Stack 2 (this is also the element at the bottom of the Stack 1).
    b. If the Stack 2 is not empty, just remove the element at the top currently.
4. For the $front$ and $back$ operations, we apply similar methods as $popfront$ and $pushback$.

By following the rules listed above, we can have a FIFO queue implemented with two stacks.

Analyze why each FIFO operation takes amortized constant time:

1. $isEmpty$:

    For the whole FIFO queue with two stacks, just set a counter C to count the number of elements in the queue. When $isEmpty$ operation is performed, just check if the counter is greater than 0. If so, it returns FALSE and vice versa. So the total number of steps is a constant, the amortized complexity is in $\Theta(1)$.

2. $pushback$:

    Since we just need to add a new element at the top of the Stack 1 and increase the counter C by 1, the number of steps for adding new element, moving

the pointer and updating counter C is a constant, so the amortized complexity is in $\Theta(1)$.

3. $popfront$:

   For the cases where the Stack 2 is not empty, we just need to remove the top element by moving the pointer pointing the top of Stack 2 once and decrease the counter C by 1, so the amortized complexity is obvious in $\Theta(1)$.

   For the cases where the Stack 2 is not empty, we need to move the elements from Stack 1 to Stack 2. We apply the counting method here (Set another counter A):

   To prove the amortized complexity is in $\Theta(1)$, the contribution of the $popfront$ needs to be constant for a sequence of operations. Assume the Stack 1 has m elements when we do $popfront$. So $pushback$ operation must be performed at least m times to build the stack. If we add $c + 2$ to the contribution of each $pushback$ operation, which doesn't change the amortized complexity of the $pushfront$, then after building Stack 1 with m elements, the counter A is $(c + 2) * m$.

   For that $popfront$ operation, considering each movement for one element, we need to subtract $c + 2$ from A (2 because we need to move the pointer twice). So after finishing moving m elements, A remains 0. Since the removal of the top element needs constant steps, we have proved that the amortized complexity of $popfront$ is in $\Theta(1)$.

4. $front$:

   It is similar with the $popfront$:

For the case where the Stack 2 is not empty, we just need to return the top element, which indeed has the amortized complexity of $\Theta(1)$.

For the case where the Stack 2 is not empty, we use the same counting method as in the $popfront$ operation and can easily get that the amortized complexity is in $\Theta(1)$.

5. $back$:

Since we just need to return the element at the top of the Stack 1, the number of steps for copying element is a constant, so the amortized complexity is in $\Theta(1)$.