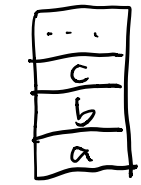
Ex 3

We initialize two stacks, the in-stack and out-stack

We first put elements in in-stack.

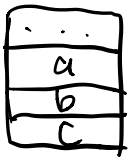


in-stack

Then we pop the elements in in-stack, and store them in out-stack.

← pointer

in-stack



← pointer.

out-stack

Now we analyze the Amortized Time

① isEmpty. We only need to decide whether the queue contains element. So the complexity is clearly in $\Theta(1)$.

② popfront.

Common case: (i) the out-stack isn't empty. then get the first element in out-stack and (ii) pop it.

if out-stack is empty. we can repeat the initialization step and (i) step.

Worst case: time complexity is $\Theta(n)$. because we need to do the initialization step. pop front element in out-stack needs $\Theta(c)$

So take potential to be $2C$. so the counter will accumulate to $2cn$. So it's enough to compensate the cost of worse case n.

So the time complexity is $\Theta(1)$

③ front, it's similar with popfront.

the amortised time complexity is $\Theta(1)$

④ push back. add a new element to the end of the queue. So we only need to put an new element on the the top of the in-stack.
so the time complexity is clearly $\Theta(1)$.

⑤ back. returns the back element of a non-empty queue without changing the queue.
If the in-stack is empty, pop all elements in out-stack and push all elements back to in-stack. Then return the top element in in-stack.
If the in-stack isn't empty. we only need to return the top element in in-stack.
The worst case time complexity is $\Theta(n)$.
Similar with the method in ② popfront.
We can get the time complexity is $\Theta(1)$.