Homework #2

(3 points) Motivation: In functional languages, we can think of data structures as directed acyclic graphs (DAGs), where each node (representing a struct) can contain only $O(1)$ data and $O(1)$ pointers to other nodes. We cannot change any nodes, we can only create new nodes. How can we implement a double ended queue a.k.a. deque in such languages?

One way is to represent the deque using two stacks (which are easy to implement). For example, a double ended queue with elements

$$1, 2, 3, 4, 5, 6, 7, 8, 9$$

can be represented using 2 stacks:

$$\begin{aligned} \texttt{front}: &\ \vdash 3, 2, 1 \\ \texttt{rear}: &\ \vdash 4, 5, 6, 7, 8, 9 \end{aligned}$$

($\vdash$ is the bottom of the stack, the top is on the right).

If we want to add/remove an element from the beginning of the queue, we add/remove it from `front`, if we want to add/remove an element from the end, we add/remove it from `rear`. The only problem is if one of the stacks gets empty. For example, if we remove elements 1, 2, 3 from the beginning, we get an empty `front`:

$$\begin{aligned} \texttt{front}: &\ \vdash \\ \texttt{rear}: &\ \vdash 4, 5, 6, 7, 8, 9 \end{aligned}$$

What if we now want to take the element from the beginning again?

a) One idea is to take all elements from `rear` and insert them in reverse order into `front`. We solve the removal and insertion at the other end symmetrically. Show that this is not a good implementation and there is a sequence of $O(n)$ operations (starting from empty deque) which takes $\Omega(n^2)$ time.

b) Better idea[1] is to maintain the invariant that

- $size(\texttt{front}) \leq 4\,size(\texttt{rear}) + 1$ and
- $size(\texttt{rear}) \leq 4\,size(\texttt{front}) + 1$.

If the invariant is violated during any operation, we rearrange the stacks so that both stacks have the same size (± 1 if the number of elements is odd).

Prove that this algorithm runs in $O(1)$ *amortized time* for each operation.

[1] see e.g. `Data.Deque` library in Haskell, which works exactly like this: `https://hackage.haskell.org/package/dequeue-0.1.12/docs/src/Data-Dequeue.html#check`