A quack is a data structure combining properties of both stacks and queues. It can be viewed as a list of elements written left to right such that three operations are possible:

- QUACKPush(x): add a new item x to the left end of the list;
- QUACKPop(): remove and return the item on the left end of the list;
- QUACKPull(): remove the item on the right end of the list.

Implement a quack using three stacks and $O(1)$ additional memory, so that the amortized time for any QUACKPush, QUACKPop, or QUACKPull operation is $O(1)$. In particular, each element in the quack must be stored in exactly one of the three stacks. Importantly, you cannot access the component stacks except through the interface functions Push and Pop.

Justify your answer using just one of the standard amortization methods (aggregate, accounting, or potential).

(Hint:) Use three stacks, $S_{pop}$, $S_{pull}$, and $S_{work}$. Any new element will be pushed onto $S_{pop}$. The newest elements will be on top of $S_{pop}$ and we attempt to keep the oldest elements on top of $S_{pull}$. Ideally then, a pop will be a pop from $S_{pop}$, and a pull will be a pop from $S_{pull}$. If $S_{pop}$ or $S_{pull}$ is empty when a pop/pull is requested, move half of the elements from one to the other using $S_{work}$ as intermediate storage.