Exercise 4

Problem 1. A *splitter* is a shared object that has only one operation, called *splitter*, that can return *stop*, *left* or *right*. Every splitter object ensures the following properties:

1. If a single process executes *splitter*, then the process is returned *stop*;
2. If two or more processes execute *splitter*, then not all of them get the same output value; and
3. At most one process is returned *stop*.

Your task is to implement a wait-free, atomic splitter object using *only* atomic (multi-valued, MRMW) registers. (Assume that each process invokes *splitter* only once. If two or more processes execute *splitter*, then they may execute concurrently or one after another. In both cases, the properties above should be ensured.)

Problem 2. The snapshot algorithm presented in the lecture has step complexity that is a function of the number of processes n. That is, in the worst case, a process needs $f(n)$ steps to complete a single *update* or *scan* operation, where f is some function.

Imagine a situation where n is very large but usually only a few processes use a snapshot object. In such a scenario, it would be best to have a snapshot implementation which step complexity is not a function of n but of the number of processes that use the shared object.

Your task is to write such an algorithm. More precisely, you should devise an algorithm for a (wait-free, atomic) snapshot object such that the step complexity of its *update* and *scan* operations is $f(k)$, where k is the number of processes that ever invoked either of the operations (in the current execution) and f is some function independent of n.