Lecture: Replication and Consistency
Exercise Sheet 2
https://pl.cs.uni-kl.de/homepage/de/teaching/ws19/rac/

1 Quiescent consistency

Quiescent consistency is a weaker variant of linearizability. It also requires that that method calls should appear to happen in a one-at-a-time, sequential order. But under quiescent consistency, only method calls *separated by a period of quiescence* appear to take effect in their real-time order.
Give an example of an execution that is quiescently consistent but not sequentially consistent, and another that is sequentially consistent but not quiescently consistent.

2 Atomic Integers

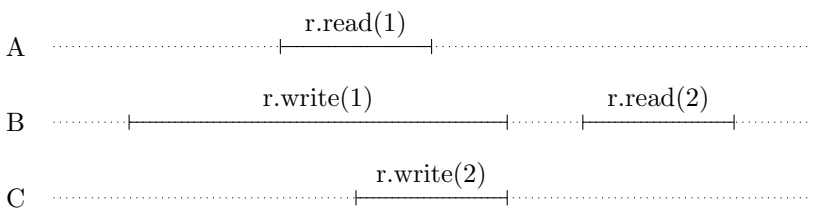
The AtomicInteger class (in the `java.util.concurrent.atomic` package) is a container for an integer value. One of its methods is `boolean compareAndSet(int expect, int update)`. This method compares the object's current value to `expect`. If the values are equal, then it atomically replaces the object's value with `update` and returns true. Otherwise, it leaves the object's value unchanged, and returns false. This class also provides `int get()` which returns the object's actual value.
Consider the FIFO queue implementation:

```
class IQueue<T> {
  AtomicInteger head = new AtomicInteger(0);
  AtomicInteger tail = new AtomicInteger(0);
  T[] items = (T[]) new Object[Integer.MAX VALUE];
  public void enq(T x) {
    int slot ;
    do {
      slot = tail.get();
    } while (! tail.compareAndSet(slot, slot+1));
    items [slot] = x;
  }
  public T deq() throws EmptyException {
    T value;
    int slot;
    do {
      slot = head.get();
      value = items[slot];
      if (value == null)
         throw new EmptyException();
    } while (! head.compareAndSet(slot, slot+1));
    return value;
  }
}
```

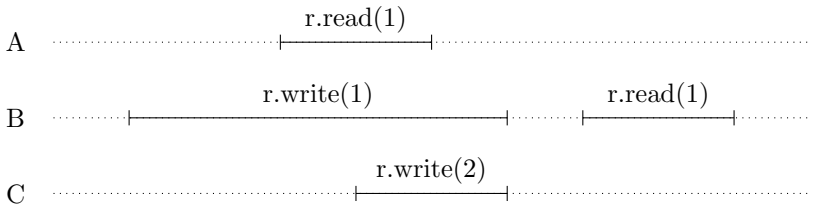
It stores its items in an array `items`, which, for simplicity, we will assume has unbounded size. It has two `AtomicInteger` fields: `tail` is the index of the next slot from which to remove an item, and head is the index of the next slot in which to place an item. Give an example showing that this implementation is not linearizable.

3 Classifying histories

For each of the histories shown, are they quiescently consistent? Are they sequentially consistent? Or linearizable? Justify your answer.
History 1:



History 2:



4 Strange methods....

Consider the following rather unusual implementation of a method `m`. In every history, the i^{th} time a thread calls `m`, the call returns after 2^i steps. Is this method wait-free, bounded wait-free, or neither?

5 Back to Peterson

Does Peterson's two-thread mutual exclusion algorithm work if we replace shared atomic registers with regular registers?

6 Atomic Registers

You learn that your competitor, the Acme Atomic Register Company, has developed a way to use Boolean (single-bit) atomic registers to construct an efficient write-once single-reader single-writer atomic register for an N-valued integer. Through your spies, you acquire the code fragment shown below, which is unfortunately missing the code for `read()`. Your job is to devise a read() method that works for this class, and to justify (informally) why it works. (Remember that the register is write-once, meaning that your read will overlap at most one write.)

```
class AcmeRegister implements Register {
  // N is the total number of threads

  // Atomic multi-reader single-writer registers
  private BoolRegister [] b = new BoolMRSWRegister[3 * N];
  public void write(int x) {
    boolean[] v = intToBooleanArray(x);
    // copy v[i] to b[i] in ascending order of i
    for (int i = 0; i < N; i++)
      b[i ]. write(v[i ]);
    // copy v[ i ] to b[N+i] in ascending order of i
    for (int i = 0; i < N; i++)
      b[N+i].write(v[ i ]);
    // copy v[ i ] to b[2N+i] in ascending order of i
    for (int i = 0; i < N; i++)
      b[(2*N)+i].write(v[i]);
  }
  public int read() {
    // missing code
  }
}
```

7 Variations on the Bakery Algorithm

Define a wraparound register that has the property that there is a value v such that adding 1 to v yields 0, not $v+1$. If we replace the Bakery algorithm's shared variables with a (a) regular, (a) safe or (b) wraparound registers, then does it still satisfy (1) mutual exclusion, (2) FIFO ordering?