1. Program X(.) has two parameters, B0 and B1, each a non-empty bag of integers (a bag is a multiset). The program creates a fifo channel system and two app systems attached to the channel. The sids of the app systems are stored in variables s0 and s1. Each system has a bag variable; initially s0's is B0 and s1's is B0. X(.) goes through a succession of cycles. In each cycle, a maximal integer in s0's bag is exchanged with a minimal integer in s1's bag. It ends as expected.

For brevity, the app programs are expressed by "initialization" steps and rules. Also for brevity, the channel program is abstracted by variable α01, the sequence of messages in transit from s0 to s1, and variable α10, the sequence of messages in transit from s1 to s0. An s0 send appends to α01's tail. An s1 receive removes from α01's head, blocking if empty. Initially, these sequences are empty.

```
program X(B0, B1) {
   ia {B0 and B1 are non-empty bags of integers}
   Seq α01 ← [];                              // start channel
   Seq α10 ← [];                              // start channel
   Sid s0 ← startSystem(App0(B0));
   Sid s1 ← startSystem(App1(B1));
}

program App0(B0) {
  initialization step:
  bool done ← false;
  Bag bg ← B0;
  remove a max entry from bg and send it;

  rule receive n;          // doable whenever msg is receivable
    if (n < max(bg))
       add n to bg;
       remove a max entry from bg and send it;
      else
         add n to bg; done ← true;
}

program App1(B1) {
  initialization step:
   Bag bg ← B1;

  rule receive n;          // doable whenever msg is receivable
     add n to bg;
     remove a min entry from bg and send it;
}
```

Atomicity assumptions: The following chunks of code are atomic

- X.init: consisting of X.main, s0.initialization and s1.initialization.
- s0.receive
- s1.receive

Progress assumption: Every atomic step is executed with weak fairness.
(So a receive rule is eventually executed if there is an incoming message.)

Part a

Prove or disprove that `X(B0,B1)` satisfies *Inv* A_0, where

A_0 : `s0.done` $\Rightarrow$
[`s0.bg` has the smallest `B0.size` elements of `union(B0,B1)`] and
[`s1.bg` has the highest `B1.size` elements of `union(B0,B1)`] and
[$\alpha 01$ and $\alpha 10$ are empty]

If you prove, come up with a list of predicates such that their conjunction, say B, satisfies the invariance rule and implies A; i.e., the following hold:

- `X`.init establishes B.
- `s0`.receive and `s1`.receive each unconditionally preserve B.
- $B \Rightarrow A_0$

All you need to supply is the list of predicates. (No need to explain why their conjunciton satisfies the above conditions.)

If you disprove, come up with a finite evolution that ends in a state that does not satisfy A_0.

Part b

Prove or disprove that `X(B0,B1)` satisfies L_0, where

L_0 : `not s0.done` *leads-to* `s0.done`

If you prove, come up with a function F that satisfies the following:

- F is never increased by a rule execution
- in any state, there is a rule that is enabled (i.e., there is an incoming msg) and whose execution decreases F
- for some (lower bound) x, $F = x$ implies `done` is true.

All you need to supply is F. (No need to explain why it satisfies the above conditions.)

If you disprove, come up with an evolution (finite or infinite) that satisfies the progress assumptions and does not satisfy L_0.