```
import io.threadcso._
import scala.math._
import scala.collection.mutable._

object Q6 extends App{
  /* This produces a stream of n random integers
   * and passes it through right */
  def producer_component(n: Int, right: ![Int]) = proc {
    for(i <- 1 to n)
      right!scala.util.Random.nextInt(1000)
  }

  /* This is the component that is required.
   * Out represents a channel through which the
   * output is written, and left/right have the
   * meaning defined in the question. */
  def sorting_component(out: ![Int], left: ?[Int], right: ![Int])
      = proc {

    // buffer represents the maximal value that we have seen so far.
    // it is set to None, if and only if we have seen no
    // integers so far.
    var buffer: Option[Int] = None
    repeat{
      val x = left?()
      buffer match {
        case None => buffer = Some (x)
        case Some(y) => {
          buffer = Some (max(x, y))
          right!min(x, y)
        }
      }
    }
    right.closeOut()
    out!buffer.get
  }

  def test_sort(n: Int) = {
    // These are the channels between components
    val inner_channels = Array.fill(n + 1)(OneOne[Int])

    // These are the channels through which the components
    // communicate with the testing rig.
    val output_channels = Array.fill(n)(OneOne[Int])
```

```
    // This will contain the result of sorting.
    val results = ArrayBuffer.fill(n)(0)

    // This is a process that runs all the components
    // in parallel.
    val components = || (for(i <- 0 until n) yield
      sorting_component(
        output(i),
        inner_channels(i),
        inner_channels(i+1)))

    // This process will write the values in the
    // output channels in results.
    val output_procedures = || (for(i <- 0 until n) yield
      proc { results(i) = output_channels(i)?() })

    // the procedure as a whole
    val system = producer_component(n, inner_channels(0)) ||
      components ||
      output_procedures

    run(system)

    assert(results sameElements results.sorted.reverse, "Test failed")
    println("Test completed")
  }

  for(i <- 1 to 100)
    test_sort(i)
  exit()
}
```

In terms of sequentially ordered messages, suppose we number messages as follows: let $M(i,j)$ denote the i'th message passed on the j'th channel (starting from the producer, indexing from 0). Then note that $M(i,j)$ must necessarily come before $M(i+1,j)$ and $M(i,j+1)$; and as the messages are defined for $i+j \leq n$; this implies that the longest chain of sequentially "bound" messages is length $O(n)$ (since the length of a path of messages is the Manhattan distance between the initial and final message, and the messages at the "edge" are at distance at most n from message $M(0,0)$).