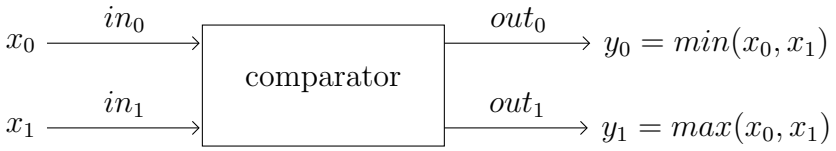
Concurrent Programming
Practical 1: Sorting Networks

This practical builds some networks to sort a list of numbers. The networks are not very sensible as software sorting networks; however, they could be implemented efficiently in hardware. The main aims of the practical are to help you get used to thinking about concurrent systems, and to gain familiarity with the SCL library.

Deadline: practical sessions in Week 3. Your answer should be in the form of well-commented code, answering the questions below. You can gain a mark of S by giving decent answers to all the non-optional questions; you can gain a mark of S+ by giving good answers to all the questions up to Question 5. The section marked "Just for fun" is just that.

Each sorting network will be built from *comparators*: simple two-element sorting components. A comparator can be pictured as below:



The comparator has two input channels, in_0 and in_1, and two output channels, out_0 and out_1. If it inputs x_0 and x_1, it outputs their minimum on out_0 and their maximum on out_1.

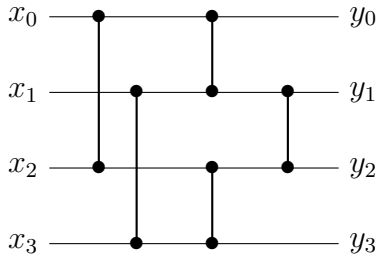
Question 1

Implement a comparator with the following signature:

```
/** A single comparator, inputting on in0 and in1, and outputting on out0
  * (smaller value) and out1 (larger value). */
def comparator(in0: ?[Int], in1: ?[Int], out0: ![Int], out1: ![Int]): ThreadGroup
```

the process should be willing to perform the inputs in either order, and perform the outputs in either order. The process should repeat this behaviour until one of its channels is closed.

Below is a sorting circuit for four inputs using five comparators.



The first four comparators direct the smallest and largest values to the top and bottom outputs; the final comparator sorts out the middle two values. Note that the first two comparators can run concurrently, as can the second pair: the longest path involves three comparators.

Question 2

Implement this sorting circuit, using the following signature.

```
/** A sorting network for four values. */
def sort4(ins: List[?[Int]], outs: List[![Int]]): ThreadGroup = {
  require(ins.length == 4 && outs.length == 4)
  ...
}
```

Test your implementation using the following idea: pick four random Ints, and send them in on the input channels; receive the outputs and check that they are a sorted version of the inputs; repeat many times.

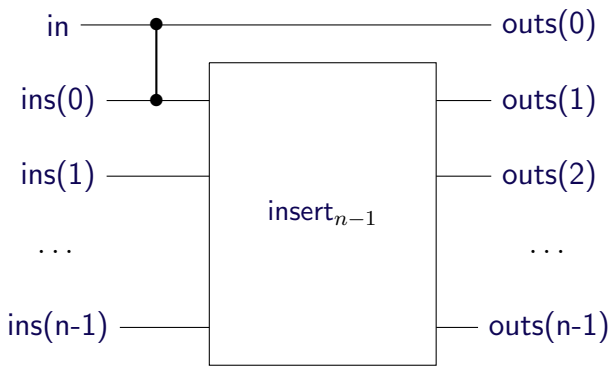
We will now implement a sorting network based on the idea of insertion sort.

Question 3

We want to implement a circuit to insert a value into a sorted list of $n \geq 1$ values, with the following signature.

```
/** Insert a value input on in into a sorted sequence input on ins.
  * Pre: ins.length = n && outs.length = n+1, for some n >= 1.
  * If the values xs input on ins are sorted, and x is input on in, then a
  * sorted permutation of x::xs is output on ys. */
def insert(ins: List[?[Int]], in: ?[Int], outs: List[![Int]]): ThreadGroup = {
  val n = ins.length; require(n >= 1 && outs.length == n+1)
  ...
}
```

Consider the circuit below to implement insert (for $n \geq 2$). The box labelled "insert_{n-1}" is (recursively) a circuit to insert the output of the comparator into the sorted list $\langle \mathsf{ins(1)}, \ldots, \mathsf{ins(n\text{-}1)} \rangle$ of length $n - 1$.



Study the circuit and persuade yourself that it indeed implements the requirements. Then implement insert based upon the circuit. You will also need a base case. Test the circuit.

Question 4

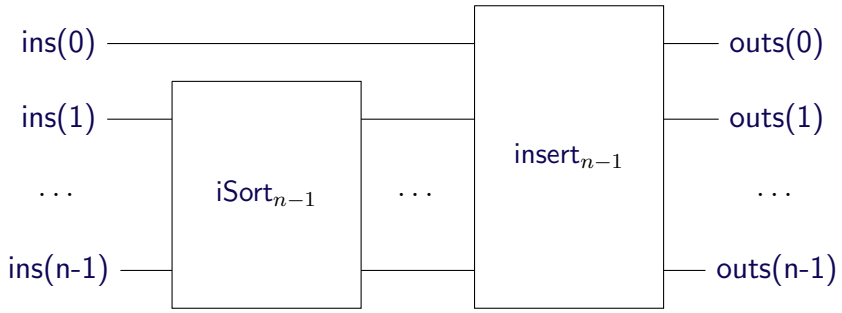
Optional: The circuit from the previous question had a path containing $O(n)$ comparators. Design, implement and test a circuit for insert such that the longest path has length $O(\log n)$.

Question 5

Use your function from either Question 3 or 4 to implement insertion sort. Use the following signature.

```
/** Insertion sort. */
def insertionSort(ins: List[?[Int]], outs: List[![Int]]): ThreadGroup = {
  val n = ins.length; require(n >= 2 && outs.length == n)
  ...
}
```

You should base your implementation on the following cicuit, where the subcircuit iSort$_{n-1}$ recursively sorts $n-1$ inputs.

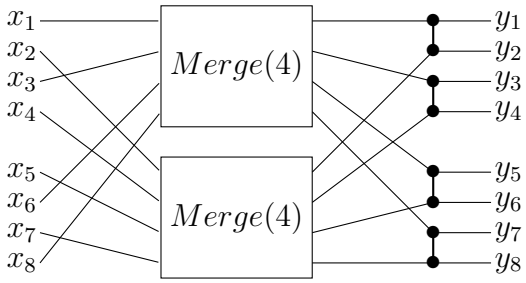


Test your implementation using the ideas from Question 2.

Just for fun

Finally, we will implement a technique known as *bitonic sorting*. The idea is closely based on merge sort: small inputs are sorted directly; larger inputs are split into two; each half is sorted recursively; and the results are merged together. For simplicity, we will assume that the number of inputs is a power of 2.

The merging circuit can be defined recursively. The base case is straightforward. Below is the merging circuit for eight inputs, making use of two four-input merge circuits and four comparators. It assumes that the two halfs of the input, $\langle x_0, x_1, x_2, x_3 \rangle$ and $\langle x_4, x_5, x_6, x_7 \rangle$ are each sorted, and merges them into a single sorted sequence.



Each half of the input is split between the two sub-merges. Corresponding outputs from the two sub-merges are fed into comparators.

Question 6

Implement this sorting algorithm. Use the following signature, for a circuit for 2^k values.

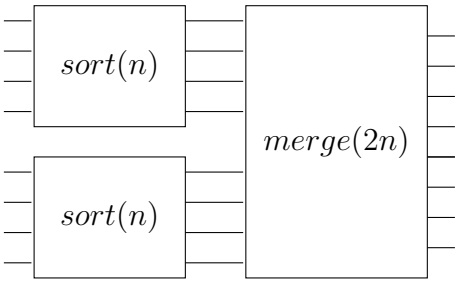
```
/** A merging network for 2^k values. If the values received on
  * ins1 and ins2 are sorted, then their merger is output on outs. */
def merge(k: Int)(ins1: List[?[Int]], ins2: List[?[Int]], outs: List[![Int]])
    : ThreadGroup = {
  val n = 1 << k; val halfN = n/2
  require(k >= 1 && ins1.length == halfN && ins2.length == halfN &&
          outs.length == n)
  ...
}
```

You might want to use the following function for doing the splitting.

```
/** Split xs into two lists, alternately.
  * @return the lists <xs(0), xs(2), xs(4),...> and <xs(1), xs(3), xs(5), ...>
  */
private def split[T](xs: List[T]): (List[T], List[T]) =
```

```
if(xs.isEmpty) (List[T](), List[T]())
else{ val (ys, zs) = split(xs.tail); (xs.head::zs, ys) }
```

The sorting circuit itself can also be defined recursively. The base case is straightforward. The recursive case is illustrated below: the input is split in half; each half is sorted; the results are merged.



Question 7

Implement this sorting circuit. Use the following signature.

```
/** A bitonic sorting network for 2^k values. */
def bitonicSort(k: Int)(ins: List[?[Int]], outs: List[![Int]]): ThreadGroup = {
  val n = 1 << k
  require(ins.length == n && outs.length == n)
  ...
}
```

Test your implementation using the ideas from Question 2.