Concurrent Programming - Sheet 3

2.
Consider a network of this shape:

```
---o-----------
   |
---o---o-------
       |
---o---o---o---
   |       |
---o-------+---
           |
---o-------+---
   |       |
---o---o---o---
       |
---o---o-------
   |
---o-----------
```

Which can be recursively extended based on the number of elements. Every horizontal line signifies a process and every vertical line a 2 element barrier (which takes constant time to resolve if both processes are ready). There are $\Theta(\log n)$ syncing rounds so this runs in $\Theta(\log n)$ time.

5.
The number of smoothing rounds must be preset, call it k (or we could go to infinity). Then we create a worker per each pixel of the image that takes coordinates i, j, that repeats k times:

```
temp = majority over all neighbours
barrier.sync()
```

```
image(i,j) = temp
barrier.sync()
```

For `barrier = Barrier(n*n)`. This implementation makes sure that no pixels are being written to while they are being read from (first sync) and that no pixels are being read form while being written to (second sync). Alternatively if the overhead gets too big, we can assign more pixels (like several rows) to each worker. Then the procedure would be very similar but the process will store a temp for each pixel and then write from temp to the image per each pixel between the syncs.