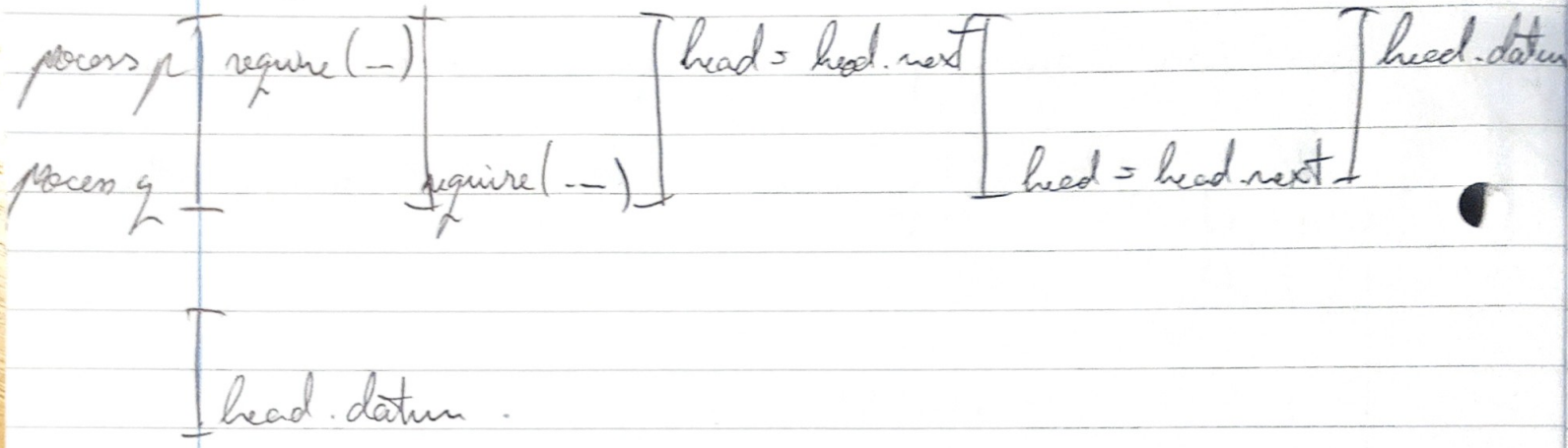
Concurrent Programming
HT22 V Cesar Trujeda

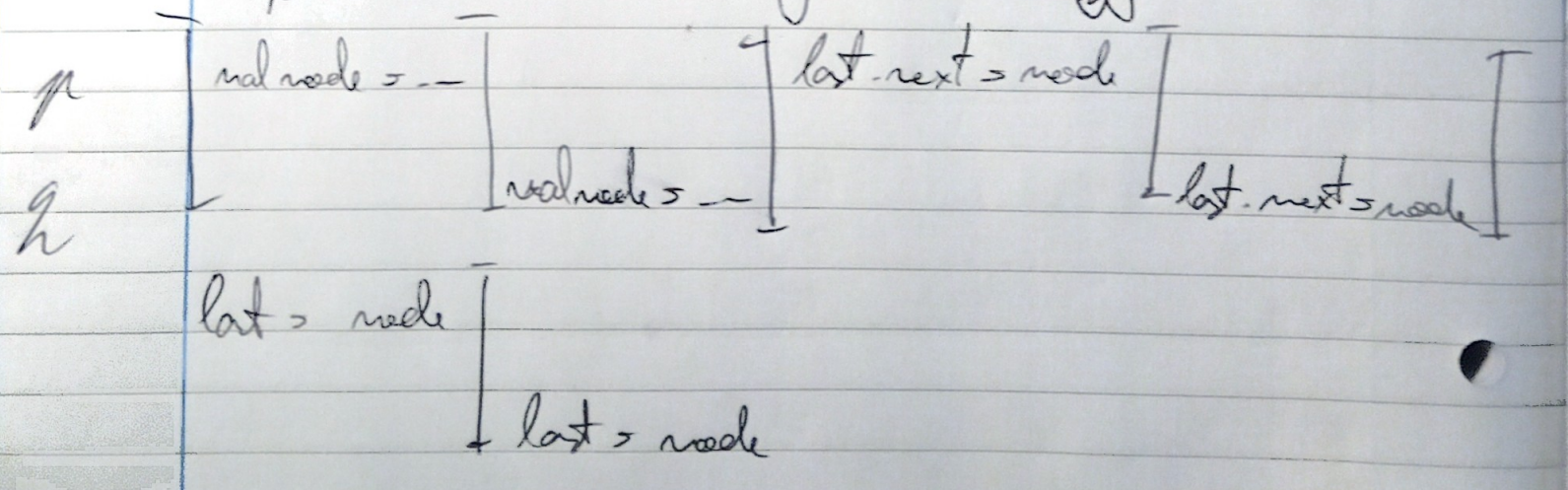
Question 3

Dequeue could enter a memory race with itself; an example being this case:



While it is true that two variables are popped, both threads will return the same value (head.datum). This would also probably completely break the queue if it were to only have one item inside.

Enqueue could enter a memory race with itself aswell:



In this case, only one of the enqueues is actually successfully executed while the other is basically ignored.

Question 4 Q4. scala

The network would not be able to sort more than n values:

If we had n+m values such that the first n are greater than the last m, what would happen is that: these n values will become the largest values "remembered" by the n nodes and the last m values would just pass through in the same order they come in (therefore, unsorted).

Each of the n values has to go through all n nodes so that would necessitate $O(n^2)$ messages.

Let m_k be the message that transmits the k-th value from the input to the first node and m'_k be the message that transmits the last remaining value (the largest) between the k-th and (k+1)-th node. We then have:

$$m_1 < m_2 < m_3 \dots < m_n < m'_1 < m'_2 < m'_3 \dots < m'_{n-1}$$

The largest totally ordered chain of messages should be $\sim 2n$.

I did some tests with the following results.

Length:	Times:	
10	0.249×10^7	} Here, the time seems to be $O(n)$
50	1.05×10^7	
100	1.81×10^7	
1000	133.5×10^7	} but it jumps to $O(n^2)$ once the

This is probably because the ratio between the length is too big. Big are the length and number of threads is too high.

Question 5

I decided that the smallest task would be computing $c[i][j]$

which means doing n multiplications and $n-1$ additions:

$$c[i][j] = \sum_{k=1}^{n} a[i][k] \cdot b[k][j]$$

I used the version that encapsulates the concurrency within objects.

Doing some tests, we get the following times:

Matrix size n	Number of workers p	numTasks	Concurrent Time x10	Sequential Time x10
500	16	1	24858 ms	2086 ms
500	16	10	1069 ms	2072 ms
500	16	20	987 ms	2373 ms
500	16	50	1000	2326
50	16	1	380	16
50	16	10	439	16
1000	16	1	70029	62454
1000	16	10	26579	62012
1000	16	25	29503	60447
1000	16	50	28563	60192
2000	16	25	243390	881678
2000	32	50	331781	1062304

From my experiments, the best number of workers is roughly 16 and the best number of tasks is around 30. If the sizes of the matrix is less than 100x100, there's no point in using concurrency