1. Consider the following variant of the LOOFA crossbar scheduler. Each input maintains a separate list of active VOQs. Before each transfer phase, the scheduler performs an odd-even sorting step as in the approximate LOOFA scheduler described in the notes. When a new VOQ $V_{ij}$ becomes active at input $i$, that VOQ is inserted into the list of active VOQs immediately after the “last” VOQ $V_{ih}$ for which $q_h \leq q_j$. If there is no such VOQ, $V_{ij}$ is inserted at the front of the list at input $i$.

Explain that if we insert $V_{ij}$, after $V_{ih}$ as described, we can conclude that $\text{slack}_{ij} \geq \text{slack}_{ih} - 1$.

Explain how the insertion of $V_{ij}$ after $V_{ih}$ causes $\text{minSlack}$ to decrease by at most 1.

After the insertion $p_{ij} = p_{ih} + 1$. Since $q_i \leq q_{ij}$, it follows from the definition of slack that $\text{slack}_{ij} \geq \text{slack}_{ih} - 1$. All VOQs that precede $V_{ij}$ after the insertion are unaffected by the insertion. Those behind it, see their $p$ values increase by 1, causing their slack values to decrease by 1. Consequently, none of the VOQs has a slack value that is more than 1 smaller than the previous value of $\text{minSlack}$. Consequently, the new value of $\text{minSlack}$ is no more than one smaller than the previous value.

2. Discuss the differences between the scheduler discussed in problem 1 and approximate LOOFA. Give an example showing how approximate LOOFA may fail to be work-conserving, even with speedup of 2.

The key difference is that the scheduler in problem 1 maintains a separate list for every input, rather than a single list that all inputs share. This means that the implementation described in the notes won’t work for the scheduler in problem 1. It also means that approximate LOOFA is not work-conserving, while the scheduler in the previous problem is, even though it does not maintain fully sorted lists.

Consider the following example to show that approximate LOOFA is not work-conserving. Suppose at time 0, the crossbar has no cells in any of its queues and the output list is sorted with output 0 first, then 1, then 2, etc. Assume that $n > 10$. Now suppose that at time 0, inputs 1, 2 and 3 all receive cells for output 0 and that the crossbar transfers cells from inputs 2 and 3 during the two transfer phases at time 0. Now suppose that at time 1, inputs 1, 4 and 5 receive cells for output 0 and that the crossbar transfers cells from inputs 4 and 5 to output 0. (These transfers are all consistent with approximate LOOFA.) At the start of step 2, this means that input 1 will have two cells for output 0 and output 0 will have two cells in its output queue. Also, notice that in the approximately sorted output list, output 0 can be no more than eight positions from the front of the list, since there have been four transfer phases in the first two steps and the odd-even sorting step done in each transfer phase moves an element by at most two positions in the list. Consequently, output 0 still comes before output $n-1$ in the list. Now suppose that at time 2, input 1 receives a cell for output $n-1$. Since output 0 comes before $n-1$ in the list, input 1 will send a cell to output 0 in the first transfer phase. Consequently, at the start of the output phase, input 0 will still have a cell for output $n-1$ and output $n-1$ will have no cell to send. This demonstrates that approximate LOOFA is not work-conserving.
Note that while ordinary LOOFA has a shared list, because the list is fully sorted, this is equivalent to giving each input its own list, since the active VOQs would appear in the same order in all input lists. Although the shared list includes inactive VOQs, these don’t affect the selection of cells to be transferred. Consequently, the implementation described in the notes could be used for full LOOFA if the controller performed enough odd-even sorting steps after each transfer to keep the list of outputs fully sorted.

3. Explain why the WDRR scheduling discipline is a PIFO discipline.

A queueing discipline is PIFO if the order in which two packets or cells will be forwarded can be determined when the second of the two packets (cells) arrives. That is, no later arriving packet can change the relative order in which two packets that arrived earlier are sent. Suppose WDRR was not PIFO. Then there must be some situation that can violate the PIFO condition. Suppose x and y are two packets in a WDRR queueing system and assume that x arrives after y. If x and y are in the same queue then clearly y must be sent before x. Assume then that x and y are in different queues. Consider a later arriving packet z. If z goes into the same queue as either x or y, it will not come to the front of its queue until after x or y does. Hence, it cannot change their relative transmission order, since the WDRR packet scheduler only considers packets at the fronts of queues when making scheduling decisions. Assume then that z goes into a third queue. While this may add a new queue to the WDRR scheduling list, it does not change the relative position of x’s and y’s queues in the scheduling list. Nor, does it affect the credits they have available to them or their scheduling weights. Consequently, the WDRR scheduler when it comes to either of the queues for x or y will send the same set of packets on each pass through the scheduling list, so the relative transmission order of x and y cannot be upset by the presence (or absence) of z.

4. The figure below shows the state of a crossbar with an NVR-P scheduler. The letters in the central array represent cells in VOQs. In each column, the cells for that column’s outputs are labeled so that the alphabetical order of the labels corresponds to the PIFO ordering at that output. Find a stable matching for this configuration and identify the cells that get transferred in each column.

To get a stable matching, we start by letting each input request its first choice and then letting the outputs select from the “offers” they receive. This results in the following pairings (0,2), (2,1) and (3,0) and leaves input 1 and output 3 unmatched.
In the second round, input 1 makes an offer to its second choice, which is output 0. Output 0 can choose to stay with its current match or accept the new offer. In this case, output 0 prefers the new offer to the previous match, so it accepts. This leaves us with the matches (0,2), (1,0), (2,1) while leaving input 3 and output 3 unmatched.

In the third round, input 3 makes an offer to its second choice, which is output 1. Output 1 can choose to stay with its current match or accept the new offer. In this case, output 1 prefers the new offer to the previous match, so it accepts. This leaves us with the matches (0,2), (1,0), (3,1) while leaving input 2 and output 3 unmatched.

In the fourth round, input 2 makes an offer to its second choice, which is output 2. Output 2 can choose to stay with its current match or accept the new offer. In this case, output 2 prefers the new offer to the previous match, so it accepts. This leaves us with the matches (1,0), (2,2), (3,1) while leaving input 0 and output 3 unmatched.

In the fifth round, input 0 makes an offer to its second choice, which is output 3. Since output 3 has no prior match, this gives us the stable matching (0,3), (1,0), (2,2), (3,1).

With this matching, output 0 gets the cell labeled a in column 0, output 1 gets the cell labeled a in column 1, output 2 gets the cell labeled a in column 2 and output 3 gets the cell labeled d in column 3.