Censor Instructions IN3030-IN4330v24

Questions 3-6, 9-11, 13-15
Multiple choice – automatically graded.

Question 1: Parallelization of Odd Even Sort

As mentioned in the exam text:

> "IMPORTANT: your parallel algorithm MUST be LOYAL to the original algorithm. As an example of a NON-LOYAL solution is one where each of *k* threads does an Odd-Even sort of 1/*k* of the array followed by a merge sort of the *k* sorted parts. This is not acceptable as it substantially modifies the original algorithm: if we use the same technique for the sequential version, we will vastly improve the original sequential algorithm. Thus you are to parallelize the algorithm merely by reordering the original sorting steps and then running some of them in parallel."

Besides being correct, the chosen solution must be efficient – as to expect a good speedup.
This means that it should utilize all cores most of the time and it should not have excessive synchronization.

Question 2: Implementation of parallel Odd Even Sort

The main idea of the parallel algorithm was asked for in questions 1, so in this question it is "the devil is in the detail". That said, we ask for code, but not real Java code, just pseudo-code, so missing semicolons, missing trivial declarations of, say, `int i`, for the iteration in a loop is essentially irrelevant (the compiler will complain and the student will comply ;-)).

Again, as mentioned above, the developed parallel MUST be LOYAL to the original algorithm.

But it is also here that, for example, the boundary conditions are important: is it "I < N" or "I <= N" or "I < N-1"?
Are the boundary conditions met? In many cases, it is necessary to have locks on TWO consecutive parts of the array because some of the checks/exchanges cross segment boundaries.
Is the terminations done properly?
Does the synchronization work?
Are the semaphores correctly initialized?
Is there excessive synchronization that will impact performance?

Question 7: PairUp Synchronization
The program should be correct.
Use Java semaphores ONLY – and only `acquire()` and `release()`.

Is it deadlock free?
Are there any race conditions?
Are the semaphores properly initialized?

Question 8: Test Program for PairUpC
Is it actually Unit Testing?
The Unit should not be modified.
Does it show that a thread actually waits?
Does it cover suitable test cases?
Does it produce suitable output?

Question 12: Cache friendliness of Quicksort
Recognize that the partitioning part accesses the segment sequentially and stores the numbers less than or equal to the pivot sequentially, which both are cache friendly.
Recognize that the recursive call is depth first, which means that it progresses from the start of the array toward the end of the array and thus is cache friendly.