Suggested solution to the exam in INF236 spring 2022:

Problem 1
L1: 0,1,2,3 in some order
L2: One of 0,1,2,3, but most likely just 0
L3: One of 0,1,2,3
L4: c1: 4, c2: 2

Problem 2a
Insert before i-loop:
#pragma omp for private(j,t1,t2) schedule(runtime)

Problem 2b
The total work is approximately $n^2/2$. With static scheduling the first thread would do the first $n/2$ iterations at a cost of approximately $n^2/8$ while the second thread would do $3/8 \times n^2$ iterations. The second thread would then dominate the runtime giving a speedup of approximately 1.33. For dynamic one would expect a speedup of 2 (minus something for overhead).

Problem 3a
The parallel runtime is given by the recursion

$T(n) = T(n/2) + O(n) = \Theta(n + n/p \log(n/p))$ where $T(n/p) = n/p \log(n/p)$.

The speedup then becomes $S(p) = n \log(n)/(n + n/p \log(n/p))$ and is limited by $\log(n)$.

Problem 3b
Each thread participating in the partitioning of a particular segment agrees on a common pivot value and then divides the sequence into equally sized parts (using static partitioning). Each thread then does a local partitioning of its assigned part. To get the elements into their correct global positions we need to first perform a sequential prefix sum operation. Following this, each thread knows where to store its values. This could be done in parallel but is most likely not worth it.

For the next level, we assign half of the available threads to the left part and half to the right part. It would be hard to implement this in a recursive fashion (at least with OpenMP) and we would therefore have an outer sequential loop that keeps track of which level one is on. We must also store the position of where each sequence starts and where it ends. After $\log(p)$ levels we have p sequences each of length n/p (ignoring pivot elements) and each thread sorts one sequence sequentially.

Problem 3c
We assign $p/2^i$ threads to partitioning each sequence, where i is the current level of the outer loop (i.e. the recursive level). Then the number of threads assigned to each sequence will be one half of what it was on the previous level of recursion. But since the length of the sequences is also cut in half for each level, each thread is always responsible for partitioning n/p elements on each level (ignoring the pivots) for a cost of $O(n/p)$ per level.

Then the prefix sum will cost $O(p/2^i)$ on recursive level i. This gives a total cost of $\Theta(p)$ for all the prefix operations across all levels (geometric series). The number of levels before the length of each sequence becomes n/p is $\log(p)$.

This gives a total cost of $O(n/p \log(p) + p)$ for the first stage. When the length of the sequences reaches n/p we have p sequences and the algorithm switches to sequential sorting taking $O(n/p \log(n/p))$ time. The total time is then
$O(n/p(\log(p) + \log(n/p)) + p) = O(n \log(n)/p + p)$.

For large values of n the speedup is asymptotically $O(p)$.

Problem 4a
Immediately.

Problem 4b
It returns an integer that is the rank of the process in the specified communicator. The integer is returned as an argument.

Problem 4c
All of the above

Problem 5
The sequential work of a thread is linear in the size of the number of elements assigned to it. Thus this is $\Theta(n^2/p)$. The communication cost is proportional to the number of elements being sent. Each process sends and receives 4 messages of length $n/\sqrt{p}$. The computation to communication ratio then becomes $\Theta(n/\sqrt{p})$.