Problem 2 (10 points)

A group of four active (non-failed) sites is trying to terminate a transaction T after a coordinator failure.

(a) Assume that a *three phase commit protocol* is in use. The table below shows various scenarios, one per line. For each scenario, the state of the four sites is given, using the terminology used in class. For each scenario, indicate if

- The sites should try to commit (write "COMMIT" in the space provided);
- The sites should try abort (write "ABORT") the transaction;
- The sites must block (write "BLOCK") until other active sites join the group;
- The indicated scenario can never arise (write "IMPOSSIBLE").

Site 1	Site 2	Site 3	Site 4	OUTCOME
W	W	W	W	
W	C	W	W	
W	W	A	W	
P	W	W	W	
P	P	P	P	
P	P	P	C	
P	P	A	P	

(b) Now assume that a *two phase commit protocol* is used. Indicate the outcome for each scenario below, using "COMMIT, "ABORT," "BLOCK," or "IMPOSSIBLE."

Site 1	Site 2	Site 3	Site 4	OUTCOME
W	W	W	W	
W	C	W	W	
W	W	A	W	
C	W	A	W	

Problem 3 (10 points)

Consider a distributed hash join executed on 6 computers as follows. Computer C_1 (source) contains two relations, R and S. Relation R contains 4000 tuples, while S contains 400 tuples. Four computers, J_1, J_2, J_3, J_4 are used for the join itself, and the final computer, C_2 collects the answer. We have no a priori knowledge of the join selectivity.

The join is performed as follows:

1. C_1 reads each R tuple from its local disk, applies a hash function and sends the tuple to either J_1, J_2, J_3 or J_4. We can assume that the hash function will partition R and S into exactly 4 parts, but we cannot assume anything about the order in which tuples will be sent to the J computers. Each receiving computer builds a hash table as the tuples arrive. The hash table fits in memory.

2. Next, C_1 repeats the process for S, hash partitioning it to J_1, J_2, J_3, J_4. As the tuples arrive at the join computers, they are joined with the R tuples. If there is a match, the answer tuple is immediately sent to C_2.

Assume the following timing information:

- C_1 takes 1 msec to read each R or S tuple, hash its key, and send it to one of the J computers. (That is, n tuples can be pumped out in n msec, if the receiving computers can take them at that rate.)
- J_i takes 2 msec to receive a tuple, hash it, and add it to a hash table.
- J_i takes 2 msec to receive a tuple, hash it, probe an existing hash table for a join match. If there is a match, it takes an additional 1 msec to send each answer tuple to C_2.
- C_2 takes 1 msec to receive a tuple and store it in its answer relation.

When two computers interact, work proceeds at the slower rate of the two. For example, if one computer can pump out messages ever 1 msec, but the other takes 2 msec to process them, then one message will be transferred only every 2 msec. Within each computer, there is no paralellism. For example, if J_i is sending an answer to C_2 in step (2) above, it cannot receive another tuple from C_1 until it is finished sending the answer tuple.

(Problem 3 continued)

(a) In the *very worst case*, how long will the join take? You can assume anything that would make the join slower, as long as it does not change the parameters given here. (For instance, it is *not* OK to assume that R has twice as many tuples.)

You can ignore times for any coordination messages (e.g., a message that says there is no more data), and any process startup or shutdown times. Measure the join time from the time the first R tuple is read at C_1 to the time the last answer tuple is processed at C_2. Also, explain clearly what constitutes the very worst case you are considering.

(b) In the *very best case*, what is the join time? You can assume anything that would make the join faster, as long as it does not change the parameters given here. Explain clearly what constitutes the very best case you are considering.