1. [20 points] Locking.
   
   a. [10 points] Draw the compatibility matrix for the locks proposed in the
      “granularity of locking” paper.
      
      This one you can find in the paper...
      
   b. [10 points] Explain why an SIX lock is compatible with an IS lock but not an S
      lock.
      
      I gave most of the points for saying something like “SIX = S + IX” and then stating that
      IS is compatible with IX but S is not. But for full points I wanted to see at least something
      about the reason for this in terms of the semantics of the locks – for example, that S
      implies permission to read any record in the file without obtaining another lock, while IX
      implies you can obtain an X lock on a record in the file, so they conflict. But IS means
      you will ask for explicit X locks, and will discover any possible conflicts at the record
      level.

2. [10 points] Oracle Concurrency Control.

   In Oracle’s multiversion concurrency control scheme, when is it safe to delete an old
   version of a record?

   When there is no active transaction that could use it – that is, the timestamp of the oldest
   active transaction is greater than the timestamp of the oldest version of the record.

3. [20 points] Degrees of consistency.

   Non-strict two-phase locking (when you can release locks before the end of the
   transaction) is not equivalent to any of the four levels of consistency defined in Gray’s
   paper.
   
   a. [10 points] Explain this statement.

   Let’s take them one at a time. Levels zero and one are not equivalent to non-strict 2PL
   because they don’t even get read locks. Level two is not equivalent because it allows
   short read locks, which can violate 2PL. Level three is not equivalent because it
   mandates that locks be held until EOT, which is not required by (non-strict) 2PL.
b. [10 points] Give an example of a schedule that would be allowed by non-strict
two-phase locking but not by at least one of the four levels in Gray’s paper.

There are many (infinitely many?) possibilities, a logical candidate would be one that
involves a dirty read – allowed by 2PL, but not by level 3 in the Gray paper.

4. [20 points] Suppose we naively implemented B-tree locking so that the readers got
no locks, but writers got locks.
   a) [10 points] Describe an incorrect behavior that could occur in this scheme.

One example is that a transaction is looking for a key, finds a pointer to a leaf, then is
paused. Then the leaf is split, moving the desired key out of the node, so when the
original searcher “wakes up” it finds that the desired key is not present.

   b) [10 points] Give the basic idea of how the B-link approach avoids this behavior
(no need for proofs or algorithms here.)

The B-link approach works by (a) ensuring that if a key moves, it can only move to the
right, and (b) adding right link pointers and high keys to facilitate searching to the right.
Note that it is not enough just to say that B-link searches the right partner – first, the key
could be moved beyond that partner (further right), second, all this only works under the
guarantee that keys only move right.

5. [20 points.] Consider the Kung and Robinson optimistic concurrency control
algorithm.

   a) [15 points] Give the three validation conditions used by the algorithm to
determine if it is “safe” for Tj to commit given that Ti has already committed.

   These are just conditions 1, 2, and 3 from the paper.

   b) [5 points] Explain how it is possible that in the parallel validation technique, in
the equivalent serial evaluation order Tj can precede Ti, yet Tj can be given a
lower transaction number than Ti.

The key insight is that the validation phases are not serialized, so Ti could enter
validation before Tj, but exit validation after Tj. In this case Ti will obtain a larger
transaction number than Tj.
8. Join Algorithms [10 points]

When people actually implement the nested loops algorithms to evaluate equijoins, they use an algorithm something like this:

Suppose we have \( k \) memory pages available for the join.
For each \( k \)-page block of \( R \) {
    Read the \( k \)-page block of \( R \) into memory
    Build a hash table on the tuples in the block
    For each \( S \) page {
        Read the \( S \) page into memory
        Probe the hash table on the \( k \)-page \( R \) block for matches
        Write any join tuples so generated into an output buffer
    }
}

(You can ignore the pages required for the current \( S \) page and the output buffer.)

Here is the question: will this algorithm ever be as fast as Hybrid Hash Join? If so, when? If not, explain why not.

Most people got that if \( R \) fits entirely in memory, the two algorithms are the same. But the block algorithm could be faster if \( \frac{1}{2} \) of \( R \) fits in memory. Here is what happens:

Block nested loops: read first half of \( R \) (\( \frac{1}{2}|R| \)), read all of \( S \) (\( |S| \)), read second half of \( R \) (\( \frac{1}{2}|R| \)), read all of \( S \) (\( |S| \)), so the total is \( |R| + 2|S| \).

Hybrid hash: read \( R \), write half of \( R \) (\( \frac{3}{2}R \)), read \( S \), write \( \frac{1}{2}S \) (\( \frac{3}{2}S \)), read \( \frac{1}{2}R \), read \( \frac{1}{2}S \), so the total is \( 2|R| + 2|S| \).