1. Consider the pair of relations R(A,B,C) and S(X,Y,Z) and the query

```
Select A, B, Z
From R, S
Where A = Y and Z < 10
```

a) Assume that R has 10,000 tuples and S has 50,000 tuples, that I/O blocks (and buffers) are 1000 bytes long, and that tuples may not cross block boundaries (i.e., each tuple is fully contained within one block). Assume also that the values for A, C, and Y require 10 bytes each, that values for B require 50 bytes, values for X require 20 bytes, and values for Z require 4 bytes. Furthermore, assume that Y is a non-nullable foreign key referencing R, and that values for Z are uniformly distributed in the range 1 to 25. How many blocks are needed to store R and S, and how many blocks are expected to be written for the output relation? Be sure to show how you calculated your answers.

Solution:
A value for attribute A requires 10 bytes to be stored, for B - 50 bytes and for C - 10 bytes. Therefore a tuple in R requires a total of 70 bytes to be stored. Since, 1000/70 ~ 14.29 and tuples may not cross blocks, we can insert at most 14 tuples in a 1000 byte long I/O block. Since R has 10000 tuples, we will need 714 "full" blocks of 14 tuples each and one block of 4 tuples to store R, or a total of 715 blocks (because 714*14+4=10000).

Similarly, a tuple of S requires 20+10+4=34 bytes to be stored. Again, since 1000/34 ~ 29.41, we can store at most 29 complete tuples from S in an I/O block. Therefore, we will need 1725 blocks to store S (1724 blocks of 29 tuples and one block of 4 tuples - 1724*29+4=50000).

Let's denote the output relation of the query as V. First, we need 10+50+4=64 bytes to store each tuple of V. A block can contain at most 1000/64, or 15 tuples. Since Y is a non-null foreign key referencing R, we will have 50,000 tuples in the natural join of R and S. From those tuples, 36% are expected to pass the predicate (Z<10), because the values of Z<10 make up 36% of the total possible values for Z (1 through 25) and each value of Z occurs with equal frequency (9/25 = 0.36). Therefore, we expect to have 18000 tuples in V, and since each block can contain at most 15 tuples, we will need a total of 18000/15=1200 blocks for V.

b) Using a straightforward translation and the nested loops join algorithm with R as the inner relation, we obtain the following execution plan P:

```
foreach tuple s in S do
    foreach tuple r in R do
```
if r.A = s.Y and s.Z < 10 then output <r.A, r.B, s.Z>

Using the statistics from part a) and assuming 3 ms to read or write a block, estimate the length of time required by execution plan P with no optimization.

**Solution:**
Recall, that each block can contain 14 tuples of R or 29 tuples of S. We assume that we have the available memory to store three blocks at the same time - one with tuples of R, one with tuples of S, and one for output tuples. The first line of the algorithm will be executed 50,000 times, and will require the copying into memory of all blocks from S, or a total of 1725 block read operations. The second line of the algorithm will require the copying into memory of all blocks of R, or 715 block reads. Since the second line will be executed for each tuple in S, the 715 blocks must be read 50,000 times. The third line requires the filling of the result table and will require a total of 1200 block writes. So, in total, we will have 1725 block reads (from line 1), 715*50000 reads from line 2 (each time we go to line 2, we need to read the table R sequentially, which is 715 blocks, and we reach line 2 – 50,000 times in the algorithm), and 1200 writes for line 3. The time for r/w operations will be (1725+715*50000+1200)*3ms \textasciitilde 29h 47m 39s.

c) Show the plan that would result from pushing down the selection ahead of the join and assuming the same join algorithm.

**Solution:**

foreach tuple s in S do
  if (s.Z < 10) then
    foreach tuple r in R do
      if r.A = s.Y then output <r.A, r.B, s.Z>


d) Without making any assumptions about the ordering of tuples in R, and without assuming the presence of indexes (even for primary keys), explain how knowing that A is the primary key for R could be used to further modify the plan to reduce the cost.

**Solution:**

Knowing that A is the key, we can produce the following access plan:

foreach tuple s in S do
  if (s.Z < 10) then
    foreach tuple r in R do
      if r.A = s.Y then
        begin
          output <r.A, r.B, s.Z>
          exit inner for loop
        end
The modification we have done is to search only for the first matching tuple in R in the inner loop. Knowing that A is the key in R, we know that there will be at-most one matching tuple in R for every tuple in S.

e) Describe a plan that could be used if there were a memory-resident index for A (i.e., no I/O is needed to read the index), and estimate the cost of your plan using the statistics from part a).

Solution:
We can do an index join between the two tables. Here is the plan:

foreach tuple s in S do
  if (s.Z < 10) then
    begin
      use the index to find the block in which the tuple with key value s.Y is stored
      retrieve the tuple from the block in r and output <r.A, r.B, s.Z>
    end

What the plan does is to scan all the tuples from S, and for each scanned tuple to use the existing index on A in R to find the matching tuple from R.

To calculate the cost, recall, that each block can contain 14 tuples of R or 29 tuples of S. We again assume that we have the available memory to store three blocks at the same time - one with tuples of R, one with tuples of S, and one for output tuples. The first line of the algorithm will be executed 50,000 times, and will require the copying into memory of all blocks from S, or a total of 1725 block read operations. The other line of the algorithm that requires I/O is the retrieve line, but it will need to retrieve only one block. So, in total we will have 1725 block reads (from line 1) and 18000 single block reads for the retrieve line (recall that 18000 tuples of S are expected to pass the condition s.Z <10), and we will write 1200 blocks in the same line to the result table. Therefore, in total we will have 1725+18000+1200 I/O operations, which will take (1725+18000+1200)*3ms ≈ 63s.

2. Assume that transactions executing an SQL program involving relations flight and fc may follow either of two possible execution sequences, modelled as:

T:
R(flight)
R(flight)
W(flight)
W(fc)
commit

T':
R(flight)
W(flight)
R(fc)
abort
a) Assume that transaction T1 follows (or at least begins to follow) execution sequence T and transaction T2 follows execution sequence T’. Show a concurrent execution schedule for T1 and T2 that is possible under 2PL but requires cascading aborts. Explain whether your schedule would be possible under Strict 2PL.

Solution:
Here is a schedule following 2PL and requiring cascading aborts:

(T2 obtains read lock on flight)
R2(flight)
(T2 obtains write lock on flight)
W2(flight)
(T2 obtains read lock on fc)
R(fc)
(T2 releases all locks)
(T1 obtains read lock on flight)
R1(flight)
R1(flight)
(T1 obtains a write lock on flight)
W1(flight)
(T1 obtains a write lock on fc)
W1(fc)
(T1 releases all locks)
T2 aborts

The schedule is 2PL because neither transaction obtains a lock after having released one. The schedule requires a cascading abort because when T2 aborts, T1 has read "dirty" data written from T2 that has aborted (the dirty data is from the table flight) and therefore has to aboard as well. Under strict 2PL T2 will not be allowed to release its locks on the table flight before committing/aborting and therefore T1 would have not read dirty data - i.e. the cascading abort would have been avoided.

b) Assuming that transactions T3 and T4 both follow execution sequence T, show a concurrent execution of T3 and T4 in which both transactions have degree 2 consistency but at least one of them does not have degree 3 consistency. Draw the serializability graph for your schedule.

Solution:
R3(flight)
R4(flight)
R4(flight)
W4(flight)
W4(fc)
C4
R3(flight)
W3(flight)
Neither T3 nor T4 reads data that has been written by an uncommitted transaction. However, T4 writes on data that was read by T3 before T3 commits. The serializability graph is:

3. Assume that the Steal/No-force policy is used and when the system restarts after a crash, it finds the following log (from Slide 10-7):

```
T1, begin
T1, x, 99, 100
T2, begin
T2, y, 199, 200
T3, begin
T3, z, 51, 50
T2, w, 1000, 10
T2, commit
T4, begin
T3, abort
T4, y, 200, 50
T5, begin
T5, w, 10, 100
T4, commit
```

a) Show the sequence of values that would be written to x, y, z and w during recovery.

Solution:

Note that this log does not include compensation log records that record the operations followed to unwind the aborted transaction T3. Therefore, we must first undo all operations in reverse order for any transaction that did not commit (whether or not it aborted); then redo operations in forward order for transactions that did commit:

- set w = 10 (T5 did not commit)
- set z = 51 (T3 aborted)
- set x = 99 (T1 did not commit)
- set y = 200 (T2 committed)
- set w = 10 (T2 committed)
- set y = 50 (T4 committed)
After these operations are completed, the programs corresponding to T1 and T5 could be restarted, since they are the known transactions in progress at the time of the crash.

b) Explain why it is not sufficient just to undo T1 and T5 even though the log shows that the other transactions were completed before the crash (either committed or aborted).

Solution:
Because a steal/no-force protocol is implemented, it may be the case that some of the results from the committed transactions are not propagated to the stable storage or that the result of aborted transactions are. Therefore, it is the case that additionally to the undo of transaction T1 and T5 which did not finish, a partial redo has to be done on the committed transactions (i.e. transactions T2 and T4) and a partial undo of the aborted transactions (T3) as part of the recovery schedule. Note as well that as explained in the solution to 3a the order in which the recovery operations are performed is important.