Part C **(50 points)**

1. **(15 points)** You are using a B+ tree to index a file. The file contains (unspanned) records that are 256 bytes long; within each record there is a 4 byte key field. The B+ tree nodes are 4KB (4096 bytes) in size. Pointers within the B+ tree are 4 bytes each.

   (a) What is the maximum number of records we can index with a 3 level B+ tree?

   (b) What is the minimum number of records we can index with a 3 level B+ tree?

   (c) Suppose that you have a file with a number of records that is between the two numbers computed above. Assume we have 100 main memory buffers exclusively devoted to hold B+ tree nodes. Estimate the expected number of IOs needed to fetch a random file record into memory. Explain how you get your answer.

2. **(17 points)** You are using a transaction processing system that stores both before and after images in its log. In this system, log records typically are 250 bytes long, with the before image of the modified record taking up 100 bytes, and the after image taking another 100 bytes.

   Your assistant has suggested the following idea for reducing the size of log records, and supposedly the overall system performance. Say a transaction updates a record, originally with value r_o to new value r_n. Instead of logging both r_o and r_n, the system can log the exclusive "or" of these two images, r_{on}. This reduces the log record size from 250 bytes to 150 bytes.

   If after a crash we wish to undo an action, we take the new value found in the appropriate record in the database, r_n, and do an exclusive "or" with r_{on} (found in log) to obtain r_o, which is then stored in the database. Similarly, if we wish to perform a redo, we read the old value from the database, r_o, do an exclusive or with r_{on} to get r_n.

   (a) Unfortunately, the scheme proposed by your assistant does not quite work correctly. Please explain what the problem is.

   (b) Suggest an efficient scheme for fixing the problem, that still uses the idea of logging the exclusive "or." (Hint: You only have to save in the log an additional one bit.) Explain how the correct scheme works.

   (c) Even the correct version of this scheme may not be advisable. Please discuss the potential disadvantages of the scheme, and how other schemes (not based on exclusive "or"s) may reduce log size.

3. **(18 points)** Two large relations, R_1 and R_2, are partitioned across four computers. Each computer holds a fourth of the tuples of each relation. The relations are partitioned at random. The relations are of equal size. Each computer has enough main memory to hold half of one partition, plus a 10 extra buffers for miscellaneous use (e.g., reading in a block of another partition, or receiving a block of tuples from another machine).

   You want to perform a natural join of the two relations, on some common attribute J. You want to use a variation of the standard merge-join algorithm that takes advantage of the available parallelism.

   Suggest such a distributed merge-join algorithm that works for this environment. Try to take advantage of available resources as much as possible. Describe your algorithm clearly (clarity of explanation is important!), and discuss the tradeoffs involved.