Problem 1 (Evaluation of Relational Operators) [56 Points]

Given the following relational operators and some properties about their input relations:

(a) Duplicate elimination operator over unsorted relation R
(b) Grouping operator (group by column X) over a sorted relation R on column X
(c) Grouping operator (group by column X) over unsorted relation R
(d) Sorting operator (sort by column X) over unsorted relation R
(e) Sorting operator (sort by column X), and assume the operator can use a B-tree index that exists on R.X to read the tuples.
(f) Join of two relations R and S
(g) Bag Union of relations R and S

1) For each of the items above, report whether the operator is "Blocking" or "Non-Blocking" and describe why. **[2 Points each Item, 1 for conclusion, 1 for your explanation]**

1) Part 1

a) ***Non-Blocking*** since the first instance of each identical group of tuples can be reported as soon as it is found.
b) ***Non-Blocking*** since the tuples within each group can be detected once value X changes, and hence the previous group can be reported.
c) ***Blocking*** since no group can be reported until all input tuples have need seen
d) ***Blocking*** since no tuple can be reported until all input tuples have need seen
e) ***Non-Blocking*** since the operator can follow the order from the B-tree index and start producing tuples as they appear in the leaf level of the B-tree index.
f) ***Non-Blocking*** because most join algorithms can start producing the joined tuples before they consume the entire input from the two sides R and S.
g) ***Non-Blocking*** since the bag union produces the total tuples from both R and S, and hence the tuples can be streamed one by one from R and then from S (or the reversed order).

2) Assume relation R is 1,000 blocks and relation S is 150 blocks, and the available memory buffers are 200. Moreover, for Point (e), the R.X index size is 70 blocks. For each of the items above, discuss: [6 **Points each Item, 2 for the number of passes, 2 for the memory constraint, 2 for the I/O cost (if needed)]**
   a. Whether the operator can be done in one pass or not.
   b. If it can be done in one pass, what are the size constrains?
   c. If it cannot be done in one pass, then how many passes are needed? Describe the algorithm that uses that number of passes you suggest? What will be the I/O cost?

2) Part 2

a) It can be done in one pass if the distinct tuples can fit in roughly 199 Blocks. That is the size constraint to be done in one pass. The I/O cost will be B(R) = 1000 I/O. If the distinct tuples are larger, then we can use two-pass sort-based or hash-based duplicate elimination algorithms. Certainly two-pass is enough since $B(R) < M^2$. The I/O cost in this case = 3 B(R) = 3000 I/O.

b) Can be done in one pass, and there are no memory constraints. The I/O cost = B(R) = 1000 I/O.

c) Can be done in one pass if the groups can fit in roughly 199 blocks. That is the size constraint to be done in one pass. The I/O cost will be B(R) = 1000 I/O. If the groups are larger, then we can use two-pass sort-based or hash-based grouping algorithms. Certainly two-pass is enough since $B(R) < M^2$. The I/O cost in this case = 3 B(R) = 3000 I/O.

d) Cannot be done in one pass since R does not fit in memory. We need two-pass sort algorithm to sort R. That is enough since $B(R) < M^2$. The I/O cost in this case = 3 B(R) = 3000 I/O.

e) It can be done in one pass by following the entries in the leaf nodes in the index. The I/O cost will be reading the leaf nodes of the index (roughly 70 I/Os) and then retrieve the records from R.
The worst case is that each pointer will cause an I/O ➔ 70 + T(R)
** Note1: No need to read the internal nodes of the index in this case, so the I/Os of the index is less 70 (but it is a small fraction, so using 70 is an approximation).
** Note 2: This example shows that in some cases, a single pass (with random I/Os) can be worse than a 2-pass algorithm.

f) Can be done in one pass where S fits in memory, and then read R one block at a time. So we will use 151 blocks for the input data. The I/O cost = B(S) + B(R) = 1150 I/O.

g) Can be done in one pass. We only need 1 buffer in memory to process both R and S one block at a time. The I/O cost = B(S) + B(R) = 1150 I/O.

Problem 2 (Query Processing Strategies & Their Costs) [44 Points (11 each)]

Consider the condition join $R1 \bowtie_{R1.a=R2.b} R2$, given the following information about the relations to be joined. The cost-metric is the number of IOs. The cost of writing the result would be the same independent of the particular join method used, hence we henceforth can ignore it. Given:

- R1 has 10,000 tuple, 10 tuples per block ➔ B(R1) = 1,000 blocks
- R2 has 2,000 tuple, 10 tuples per block ➔ B(R2) = 200 blocks
- The available memory buffers are 52

1. Assume we use a block-oriented nested loop join.
   a. Which relation you suggest to be the outer relation? [3 points]
   b. What is the cost of the join if we use the outer relation as you suggested? [4 points]
   c. What is the cost of the join if we use the other relation (not what you suggestion) as the outer one? [4 points]

a) The outer relation should be the smaller one (Relation R2).
b) The I/O cost will be: B(R2) + 200/51 x B(R1) = 200 + 3922 = 4122 I/Os
c) The I/O cost will be: B(R1) + 1000/51 x B(R2) = 1000 + 3922 = 4922 I/Os

Note: If you divide by 50 instead of 51 (where you take into account the 1 output buffer) that is correct too.

2. Assume we use a sort-merge join, and we use the "Efficient Sort-Merge" algorithm covered in class where we merge the sorting and joining together
   a. What is the cost of the join algorithm? [6 points]
   b. What is the minimum number of buffers needed for the cost to remain unchanged, i.e., Can we use less than 52 buffers and still have the same cost that you calculated in 2.a? [5 points]

a) The I/O cost will be: 3 (B(R1) + B(R2)) = 3600 I/Os
b) The smallest M where the algorithm can work in 2 passes and we keep the I/O cost as is should be the smallest M such that M^2 = B(R1) + B(R2) ➔ smallest M = 35 Buffers

3. Assume we use a hash-join, and we will do a simple hash-join.
   a. What is the cost of the join algorithm? [6 points]
   b. What is the minimum number of buffers needed for the cost of the hash join to remain unchanged, i.e., Can we use less than 52 buffers and still have the same cost that you calculated in 3.a? [5 points]

a) The I/O cost will be: 3 (B(R1) + B(R2)) = 3600 I/Os
b) The smallest M where the algorithm can work in 2 passes and we keep the I/O cost as is should be the smallest M such that M^2 = Min(B(R1), B(R2)) ➔ M2 = B(R2) ➔
   smallest M = 15 Buffers

4. Assume we use an index-join with R2 as the outer relation, and we have an index on R1.a. Assume that the index fits in memory. Moreover, on average we get 5 R1 tuples matching every R2 tuple.
   a. What is the cost of the join algorithm? [11 points, 3 for your assumptions, 8 for your result]
   Here you might need to make assumptions i.e. one or both tables are clustered (or not), or index is clustered or not, index in memory or not, etc.

You need to make some assumptions here (Based on the assumptions the answer may differ).

Assume R2 is clustered ➔ we need for the outer relation B(R2) = 200 I/Os

Assume the index is in memory ➔ No I/O for reading it
Note: If you assume the index is not in memory, then the I/O will be for each probe. Unless you state you will be done once and kept in memory for the rest of the query.

Assume the index is clustered ➔ each 5 tuples will fit in one page = 1 I/O (roughly)

The cost is = B(R2) + T(R2) x 1 = 200 + 2000 = 2200 I/O. [5 points]