Question 1  

Suppose you have two clustered relations $R(A,X,Y)$ and $S(B,C,Z)$. You have the following indexes on $S$.

- A non-clustering B-tree index on attribute $B$ for $S$.
- A clustering B-tree index on attribute $C$ for $S$.

Assume that both indexes are kept entirely in memory always (i.e., you do not need to read them from disk). Also, assume that all of the tuples of $S$ that have the same value of attribute $C$ are stored in sequentially adjacent (i.e., contiguous) blocks on disk. That is, if more than one block is needed to store all of the tuples with some value of $C$, then these blocks will be located sequentially on the disk.

You have the following information about $R$ and $S$:

- 100 tuples of $R$ are stored per block on disk. Assume that blocks of $R$ are laid out contiguously on disk.
- $T(R) = 360,000$ (number of tuples of $R$). The values of attribute $A$ in $R$ range from 1 to 360,000. Assume that $A$ is a key of $R$, so each tuple in $R$ has a unique value of $A$ in $[1, ..., 360,000]$.
- 5 tuples of $S$ are stored per block on disk.
- $T(S) = 1,200,000$ (number of tuples of $S$).
V(S,B)= 1200, i.e., there are 1200 distinct values of attribute B in S. Assume that these values are distributed uniformly in S, so each value of B occurs T(S)/V(S,B) = 1000 times in S. Furthermore, assume that these values range from 1 to 1200. That is, for each value v in [1,...,1200], there are 1000 tuples in S with S.B = v.

V(S,C)= 120,000, i.e., there are 120,000 distinct values of attribute C in S. Assume that these values are distributed uniformly in S, so each value of C occurs T(S)/V(S,C) = 10 times in S. Furthermore, assume that these values range from 1 to 120,000. That is, for each value v in [1,...,120,000], there are 10 tuples in S with S.C = v.

You want to execute the following query:

```
SELECT * 
FROM R, S 
WHERE R.A = S.B AND R.A = S.C 
```

We present you with two indexed-nested-loop-join plans:

**Plan 1:**

For every block BLK of R, retrieved using a scan of R

For every tuple r of BLK

   Use the index on B for S to retrieve all of the tuples s of S such that s.B=r.A
   
   For each of these tuples s, if s.C=r.A, output r.A, r.X, r.Y, s.B, s.C, s.Z

**Plan 2:**

For every block BLK of R, retrieved using a scan of R

For every tuple r of BLK

   Use the index on C for S to retrieve all of the tuples s of S such that s.C=r.A
   
   For each of these tuples s, if s.B=r.A, output r.A, r.X, r.Y, s.B, s.C, s.Z

Note that both plans read R one block at a time, and retrieve all S tuples that join with tuples in the current block of R (using one of the indexes on S) before reading the next block of R.

a. Analyze each of these plans in terms of their behavior regarding accesses to disk. For each plan compute the number of sequential accesses and the number of random accesses to blocks on disk. Given that random accesses are at least an order of magnitude costlier than sequential accesses, which of the plans performs better?

b. Assume all statistics remain the same except for the number of tuples of S stored per block on disk, which now reduces to 2 (from 5). How does this change your answer to (a)?

c. Let the variable X represent the number of tuples of S stored per block on disk. Assuming all other statistics remain the same as before, what values of X in [1,...,10,000] will make the worse plan of (a) perform better than the other?

d. Which plan is better if both indexes are non-clustering, and everything else remains as specified originally in the question? Note that now tuples of S that have the same value of attribute C are not stored in contiguous blocks on disk.

2
e. Which plan is better if both indexes are non-clustering, and \(V(S,B) = 180,000\)? There are 180,000 distinct values of attribute B in S. Assume that these values range from 1 to 180,000 and are distributed uniformly in S. \(V(S,C)= 120,000\) as before.

f. Suppose everything remains as specified originally in the question except that values of attribute B come from the domain 1-3,600,000. (That is, the domain is positive integers 1,2,3 and so on up to 3.6 million.) Assume that the values of attribute B in S are distributed uniformly in this domain, and \(V(S,B) = 1,200,000\). Which plan is better in this scenario?

**Question 2**

Points 10

A set of indexes is called a **covering index set** for a query if the query can be evaluated using these indexes only (i.e, without fetching any data records). For queries Q1 and Q2 below:

(a) Give a minimal covering index set

(b) Give an efficient technique (need not be a query plan) to evaluate the query using your minimal covering index set from (a)

(c) Compute the number of disk blocks read by your technique from (b)

Queries Q1 and Q2 are as follows:

**Q1:**

\[
\text{SELECT R.a} \\
\text{FROM R, S} \\
\text{WHERE R.a = S.a}
\]

**Q2:**

\[
\text{SELECT DISTINCT R.a} \\
\text{FROM R, S, T} \\
\text{WHERE R.a > S.a AND S.a >= T.b}
\]

Note that SQL’s DISTINCT operator used in Q2 will eliminate duplicates from Q2’s result. DISTINCT is discussed in Section 6.4.1 of the textbook.

Make the following assumptions about relations R(a,b), S(a,b), and T(a,b) (Note: you may not need all this information to compute the number of disk blocks accessed):

- R.a is the primary key of R, S.a is the primary key of S, and T.a is the primary key of T.
- All relations are clustered.
- \(B(R) = 1000\), \(B(S) = 10,000\), and \(B(T) = 100,000\)
- \(T(R) = 10,000\), \(T(S) = 50,000\), and \(T(T) = 300,000\). (\(T(T)\) denotes the number of tuples in relation T.)
- There are clustering B-tree indexes on R.a, S.a, and T.a. There are non-clustering B-tree indexes on R.b, S.b, and T.b.
For simplicity of computation, assume that all indexes contain two levels, with the root node in the first level and some number of leaf nodes in the second level. The indexes on R.a and R.b contain 25 leaf nodes each; the indexes on S.a and S.b contain 250 leaf nodes each; and the indexes on T.a and T.b contain 2500 leaf nodes each.

Assume that root nodes of all indexes are always in memory so that access to a root node never incurs an I/O.

**Question 3**

Figure 15.7 in the textbook gives the pseudocode for an iterator for a tuple-level nested loop join. For sample pseudocode for other iterators, see the iterators for TableScan and Select in Lecture Slides #7.

(a) Consider the query $R_1 \bowtie R_2 \bowtie R_3$. Let the number of tuples in the three relations be: $|R_1| = 3$, $|R_2| = 6$, $|R_3| = 4$. Assume that whenever a join is performed during query execution, the join has selectivity $= \frac{1}{2}$ (i.e., half of the tuples in the cross-product satisfy the join predicate). Suppose the iterator technique is used, the joins are iterator-based tuple-level nested-loop joins, and the relations are accessed using iterator-based TableScans. If the join is executed as the left-deep tree $(R_1 \bowtie R_2) \bowtie R_3$, how many times is the GetNext iterator function called totally? (Figure 1 shows the left-deep plan.) Assume for this question and for (b) below that once an iterator returns “EOT” (End-Of-Tuples), no further GetNext call will be invoked on that iterator. (The EOT terminology is from the Lecture Slides. The textbook uses “NotFoundError” for the same purpose.) Your answers should include the GetNext calls which return “EOT”.

(b) If the join in (b) is executed as the right-deep tree $R_1 \bowtie (R_2 \bowtie R_3)$, how many times is the GetNext iterator function called? (Figure 2 shows the right-deep plan.)

![Figure 1: Example left-deep plan](image)

**Question 4**

The following information is available about relations R and S: 

4
Relation R is clustered and the blocks of R are laid out contiguously on disk. \( B(R) = 1000 \) and \( T(R) = 10,000 \).

Relation S is clustered and the blocks of S are laid out contiguously on disk. \( B(S) = 500 \) and \( T(S) = 5000 \).

\( M = 101 \) blocks.

For simplicity, we will assume that a random access can be done on average in time \( t_r = 20 \) ms, and a sequential access can be done on average in time \( t_s = 1 \) ms. For example, scanning five contiguous blocks on disk, assuming the first access is random, incurs a cost \( t_r + 4t_s \).

How will you extend the “Efficient” Sort-Merge Join algorithm (that we learned in class) to minimize cost when our cost model distinguishes between random accesses and sequential accesses? (Note that in class we did not distinguish between random and sequential accesses.) Compute the cost of your algorithm.

**Question 5**

The following information is available about relations R and S:

- Relation R is clustered and the blocks of R are laid out contiguously on disk. \( B(R) = 1250 \) and \( T(R) = 10,000 \).
- Relation S is clustered and the blocks of S are laid out contiguously on disk. \( B(S) = 1000 \) and \( T(S) = 5000 \).
- \( M = 101 \) blocks.

a. For this question assume that our cost model is the same as the one we have been using in class, namely, the total number of blocks read or written, excluding the writes for the final output. Compute the number of buckets and the cost for the most efficient Hybrid Hash Join of relations R and S. (A similar example is worked out in Example 15.10, Section 15.5.6, of the textbook.)

b. Suppose everything in the question remains the same except now \( M = 51 \). Compute the number of buckets and the cost for the most efficient Hybrid Hash Join of relations R and S.
c. When our cost model is the total number of blocks read or written, then the Hybrid Hash Join strategy—where we reduce the number of buckets appropriately, and keep one bucket in memory—clearly has lower cost than a regular hash join. If our cost model distinguishes between random accesses and sequential accesses, then does this advantage of Hybrid Hash Join over the regular hash join increase, decrease, or remain the same? Describe your reasoning.

Question 6  

Points 5

Describe a query execution scenario where the Most-Recently-Used (MRU) buffer page replacement strategy clearly outperforms both the Least-Recently-Used (LRU) and the First-In-First-Out (FIFO) strategy. Note that performance of a page replacement strategy is characterized in terms of the percentage of block accesses that can be satisfied from the buffer, i.e., without going to the disk. You should give a clear analysis of the performance of MRU, FIFO, and LRU for the query execution scenario that you give.