Exercise 1 (query evaluation): 
Consider the join R><S, given the following information about the relations to be joined. The cost metric is the number of page I/Os unless otherwise noted, and the cost of writing out the result should be uniformly ignored.

Relation R contains 10,000 tuples and has 10 tuples per page.
Relation S contains 2000 tuples and also has 10 tuples per page.
Attribute b of relation S is the primary key for S.
Both relations are stored as simple heap files.
Neither relation has any indexes built on it.
52 buffer pages are available.

1. What is the cost of joining R and S using a page-oriented simple nested loops join? What is the minimum number of buffer pages required for this cost to remain unchanged?

2. What is the cost of joining R and S using a block nested loops join? What is the minimum number of buffer pages required for this cost to remain unchanged?

3. What would be the lowest possible I/O cost for joining R and S using any join algorithm, and how much buffer space would be needed to achieve this cost? Explain briefly.

4. How many tuples does the join of R and S produce, at most, and how many pages are required to store the result of the join back on disk?

5. Would your answers to any of the previous questions in this exercise change if you were told that R.a is a foreign key that refers to S.b?

Exercise 2:
Consider the following relational schema and SQL query. The schema captures information about employees, departments, and company finances (organized on a per department basis).

```
Emp(eid: integer, did: integer, sal: integer, hobby: char(20))
Dept(did: integer, dname: char(20), floor: integer, phone: char(20))
Finance(did: integer, budget: real, sales: real, expenses: real)
```

Consider the following query:
```
SELECT D.dname, F.budget
FROM Emp E, Dept D, Finance F
WHERE E.did = D.did AND D.did = F.did AND D.floor = 1
AND E.sal >= 59000 AND E.hobby = 'reading'
```

1. Identify a relational algebra tree (or a relational algebra expression if you prefer) that reflects the order of operations a decent query optimizer would choose.
2. List the join orders (i.e., orders in which pairs of relations can be joined to compute the query result) that a relational query optimizer will consider. Briefly explain how you arrived at your list.

3. Suppose that the following additional information is available: Unclustered B+ tree indexes exist on Emp.did, Emp.sal, Dept.floor, Dept.did, and Finance.did. The system’s statistics indicate that employee salaries range from 10,000 to 60,000, employees enjoy 200 different hobbies, and the company owns two floors in the building. There are a total of 50,000 employees and 5,000 departments (each with corresponding financial information) in the database. The DBMS used by the company has just one join method available, index nested loops.

(a) For each of the query’s base relations (Emp, Dept, and Finance) estimate the number of tuples that would be initially selected from that relation if all of the non-join predicates on that relation were applied to it before any join processing begins.

(b) Given your answer to the preceding question, which of the join orders considered by the optimizer has the lowest estimated cost?
Let \( M = 1000 \) be the number of pages in \( R \), \( N = 200 \) be the number of pages in \( S \), and \( B = 52 \) be the number of buffer pages available.

1. Basic idea is to read each page of the outer relation, and for each page scan the inner relation for matching tuples. Total cost would be

\[
\text{Total Cost} = N + (N \times M) = 200,200
\]

The minimum number of buffer pages for this cost is 3.

2. This time read the outer relation in blocks, and for each block scan the inner relation for matching tuples. So the outer relation is still read once, but the inner relation is scanned only once for each outer block, of which there are \( \{\#\text{pages in outer} / (B - 2)\} \)

\[
= 200 / 50 = 4
\]

Total cost would be

\[
\text{Total Cost} = N + M * \{N / (B - 2)\} = 4,200
\]

If the number of buffer pages is less than 52, the number of scans of the inner would be more than 4 since \( =200 / 49 \) is 5. The minimum number of buffer pages for this cost is therefore 52.

3. The optimal cost would be achieved if each relation was only read once. We could do such a join by storing the entire smaller relation in memory, reading in the larger relation page-by-page, and for each tuple in the larger relation we search the smaller relation (which exists entirely in memory) for matching tuples. The buffer pool would have to hold the entire smaller relation, one page for reading in the larger relation, and one page to serve as an output buffer.

\[
\text{Total Cost} = M + N = 1,200
\]

The minimum number of buffer pages for this cost is \( N + 1 + 1 = 202 \).

4. Any tuple in \( R \) can match at most one tuple in \( S \) because \( S.b \) is a primary key (which means the \( S.b \) field contains no duplicates). So the maximum number of tuples in the result is equal to the number of tuples in \( R \), which is 10,000.

The size of a tuple in the result could be as large as the size of an \( R \) tuple plus the size of an \( S \) tuple (minus the size of the shared attribute). This may allow only 5 tuples to be stored on a page. Storing 10,000 tuples at 5 per page would require 2000 pages in the result.

5. The foreign key constraint tells us that for every \( R \) tuple there is exactly one matching \( S \) tuple (because \( S.b \) is a key). The Sort-Merge and Hash Joins would not be affected, but we could reduce the cost of the two Nested Loops joins. If we make \( R \) the outer relation then for each tuple of \( R \) we only have to scan \( S \) until a match is found. This will require scanning only 50% of \( S \) on average.

For Page-Oriented Nested Loops, the new cost would be

\[
\text{Total Cost} = M + \{M * (N/2)\} = 101,000
\]

and 3 buffer pages are still required.

For Block Nested Loops, the new cost would be

\[
\text{Total Cost} = M + (N/2) * \{M / (B - 2)\} = 3,000
\]

and again this cost can only be achieved with 52 available buffer pages.
Answer 2:
The answer to each question is given below.

1. \[ \pi_{D,dname,F.budget} \left( \left( \pi_{E.did} \left( \sigma_{E.sal \geq 59000, E.hobby = 'reading'}(E) \right) \right) \bowtie \pi_{D.did,D.dname} \left( \sigma_{D.floor = 1}(D) \right) \right) \bowtie \pi_{F.budget,F.did}(F) \]

2. There are 2 join orders considered, assuming that the optimizer only consider left-deep joins and ignores cross-products: (D,E,F) and (D,F,E)

3. (a) The answer to each relation is given below.
   * Emp: card = 50,000, E.sal \geq 59,000, E.hobby = "reading"
   resulting card = 50000 \times \frac{1}{50} \times \frac{1}{200} = 5
   * Dept: card = 5000, D.floor = 1
   resulting card = 5000 \times \frac{1}{2} = 2500
   * Finance: card = 5000, there are no non-join predicates
   resulting card = 5000

   (b) Consider the following join methods on the following left-deep tree: (E \bowtie D) \bowtie F).
   The tuples from E will be pipelined, no temporary relations are created. First, retrieve the tuples from E with salary \geq 59,000 using the B-tree index on salary; we estimate 1000 such tuples will be found, with a cost of 1 tree traversal + the cost of retrieving the 1000 tuples (since the index is unclustered) = 3 + 1000 = 1003. Note, we ignore the cost of scanning the leaves. Of these 1000 retrieved tuples, on the fly select only those that have hobby = "reading", we estimate there will be 5 such tuples.
   Pipeline these 5 tuples one at a time to D, and using the B-tree index on D.did and the fact that D.did is a key, we can find the matching tuples for the join by searching the Btree and retrieving at most 1 matching tuple, for a total cost of 5(3 + 1) = 20. The resulting cardinality of this join is at most 5.
   Pipeline the estimated 3 tuples of these 5 that have D.floor = 1 up to F, and use the Btree index on F.did and the fact that F.did is a key to retrieve at most 1 F tuple for each of the 3 pipelined tuples.
   This costs at most 3(3 + 1) = 12. Ignoring the cost of writing out the final result, we get a total cost of 1003 + 20 + 12 = 1035.