Problem 1.

How many possible plans are there for an \( n \)-way join query \( R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n \), if we use only one type of asymmetric binary join operator in our plans? Your answer should be a closed-form or recurrence formula. Also, compute your answer for \( n = 7 \).

Remember to consider all bushy plans—not just left-deep ones. For example, three possible plans for \( n = 3 \) are shown below. There are a total of 12 plans for \( n = 3 \).

![Possible plans for n=3](image)

Problem 2.

Consider relations \( R(A, B, C) \), \( S(C, D) \), \( T(D, E) \) with the following statistics:

- \( |R| = 100; |\pi_A R| = 100; |\pi_B R| = 10; |\pi_C R| = 50; \)
- \( |S| = 500; |\pi_C S| = 30; |\pi_D S| = 100; \)
- \( |T| = 400; |\pi_D T| = 400; |\pi_E T| = 150. \)

Estimate the number of the tuples returned by the following queries:

(a) \( \sigma_{A = 10} R \)
(b) \( \sigma_{A = 10 \text{ AND } B = \text{\'Bart\'}} R \)
(c) \( \sigma_{A = 10 \text{ OR } B = \text{\'Bart\'}} R \)
(d) \( R \bowtie S \)
(e) \( R \bowtie S \bowtie T \)

Problem 3.

Consider relations \( \text{Employee(eno, ename, pno, salary)} \) and \( \text{Project(pno, pname, location, budget)} \), where the key attributes are underlined. Furthermore, \( \text{Employee.pno} \) references \( \text{Project.pno} \). The most common queries on \( \text{Project} \) use the set of simple predicates \{location = 'RTP', location = 'NYC', budget < 1000, budget \geq 3000\}.

(a) Compute the primary horizontal fragments of \( \text{Project} \) based on the given set of simple predicates.
(b) Suppose that the horizontal partitioning of Employee is derived from Project. Transform the following SQL query into a relational algebra plan over the fragments, pull up union and join, push down selection and projection, and simplify the plan as much as possible.

```sql
SELECT ename, pname
FROM Employee, Project
WHERE Employee.pno = Project.pno
AND location = 'RTP' AND budget < 2000;
```

Problem 4.

Consider the general fragment and replication join algorithm discussed in lecture. Suppose that \( P \) sites are available to process \( R \bowtie S \). The algorithm partitions \( R \) into \( m \) fragments \( R_1, R_2, \ldots, R_m \) of size \( |R|/m \) each, and \( S \) into \( n \) fragments \( S_1, S_2, \ldots, S_n \), of size \( |S|/n \) each, where \( m \cdot n = P \). Each site receives a copy of \( R_i \), a copy of \( S_j \), and computes \( R_i \bowtie S_j \) locally. This problem explores the optimal choice of \( m \) and \( n \).

(a) If the cost of sending \( t \) tuples from one site to another is \( c \cdot t \), what is the total communication cost of the algorithm (assuming that the site storing \( R \) and \( S \) does not participate in join)?

(b) If the cost of computing \( R_i \bowtie S_j \) locally at a site is \( k \cdot (|R_i| + |S_j|) \) (e.g., if we use sort-merge join), what is the optimal choice of \( m \) and \( n \)?

(c) If the cost of computing \( R_i \bowtie S_j \) locally at a site is \( k \cdot |R_i| \cdot |S_j| \) (e.g., if we use nested-loop join), what is the optimal choice of \( m \) and \( n \)?

Problem 5.

This problem explores why semijoin reducers do not work with cyclic joins. Consider an \( n \)-way join \( R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie \ldots \bowtie R_n(A_n, A_1) \). Note that \( R_1 \) joins with \( R_1 \) on \( A_1 \), making this \( n \)-way join cyclic. Your job is to construct a database instance in which:

- \( R_i \neq \emptyset \) for any \( i \).
- \( R_i \bowtie R_j = R_i \) for any \( i \) and \( j \); that is, pair-wise semijoins cannot reduce anything.
- \( \bowtie_{1 \neq j} R_i \neq \emptyset \) for any \( j \); that is, any \( (n - 1) \)-way join is non-empty. Here \( \bowtie_{1 \neq j} R_i \) is a short hand for \( R_1 \bowtie \ldots \bowtie R_{j-1} \bowtie R_{j+1} \bowtie \ldots \bowtie R_n \).
- \( \bowtie_i R_i = \emptyset \); that is, the final \( n \)-way join is empty. Here \( \bowtie_i R_i \) is a short hand for \( R_1 \bowtie \ldots \bowtie R_i \bowtie \ldots \bowtie R_n \).