Question 1  
Consider the following query over relations $R_1$–$R_4$:

$$R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$$

Suppose there are three possible access methods for each $R_i$ and two possible join methods for each join. Assume that all combinations of access and join methods are feasible, and that both join methods are asymmetric (e.g., the two join methods could be Nested-Loop join and Hash join, both of which are asymmetric).

1. How many different left-deep plans are there for this query?

2. How many different bushy plans are there for this query? Note that a plan that is not left-deep or right-deep is bushy.

3. How would your answer to (1) change if there is only one join method, but this join method is symmetric (e.g., the join method could be Sort-Merge join, which is symmetric)? Compute the number of different left-deep plans in this case.

Question 2  
Consider the following heuristic for pruning the space of plans during query optimization: *cross products are never formed unless there exists no plan involving only joins*. Construct a scenario where this heuristic fails, i.e., it removes the optimal physical plan from consideration.
An *scenario* consists of a query, statistics for input relations of the query, and statistics for any intermediate relations that may occur in physical plans for the query. Statistics for a relation $R$ consists of $B(R)$ (the number of blocks of $R$) and $T(R)$ the number of tuples of $R$. For simplicity, assume:

1. The only physical join operator available in the system is tuple-based, nested loop join.
2. Only pipelined plans are considered.
3. An optimal plan is one that minimizes the total number of $\text{getNext()}$ calls over all the operators in a physical plan (and not one that minimizes disk I/O).

**Question 3**  
Points 20

Consider the following heuristic for pruning the space of plans during query optimization: *only plans with left-deep join trees are considered; those with right-deep or bushy join trees are pruned.* Construct a scenario where this heuristic fails. For simplicity, assume:

1. The only physical join operator available in the system is tuple-based, nested loop join.
2. Plans could be pipelined or materialized.
3. An optimal plan is one that minimizes the total number of $\text{getNext()}$ calls over all the operators in a physical plan (and not one that minimizes disk I/O).

**Question 4**  
Points 20

Consider the join of four relations $R_1 \Join R_2 \Join R_3 \Join R_4$. We have not shown the join predicates since they are not relevant to this problem. Consider two plans for joining these relations: one using a left-deep join tree (Figure 1) and one using a right-deep join tree (Figure 2). $X_1, X_2, X_3, X_4$ represent various intermediate relations produced in the plans. All the join operators are tuple-based, nested loop joins. The plans are fully pipelined. Only 4 blocks of memory are available. We have $B(R_1) = B(R_2) = B(R_3) = B(R_4) = 1000$ blocks, and $T(R_1) = T(R_2) = T(R_3) = T(R_4) = T(X_1) = T(X_2) = T(X_3) = T(X_4) = 10000$ tuples. What is the number of disk I/Os for the left-deep plan and the right-deep plan?

**Question 5**  
Points 20

Consider a SQL query of the form:

```
Select <project-list>
From  R1, R2, ..., Rn
Where  <predicates>
```

A useful representation of this query involving joins is as an undirected *graph* as described below: There are $n$ nodes in the graph, one corresponding to each of the $n$ relations $R_1, \ldots, R_n$. There is an edge between nodes $R_i$ and $R_j$ if there is join predicate involving $R_i$ and $R_j$ (i.e., of the form $R_i.X = R_j.Y$). As a more concrete example, Figure 3 shows the graph for the following SQL query.
Assume we use the following heuristics (described in the class) to prune the space of join trees:

1. Plans with cross products are ignored.
2. Only plans with left-deep join trees are considered/allowed.

Figure 4 shows the graphs corresponding to five different queries (A,B,C,D,E). For each query indicate the number of join trees that remain after applying the above heuristics. Note that queries D and E are general: they are defined in terms of a general n.
Figure 4: Example query graph