Problem 1.
1.1.

50,000 tuples

Notice that 'A' is NOT the key in neither relation, since there are obviously duplicates.

Also, no need for the students to give explanations - here are the explanations, anyway:

Since 'A' is not the key, we use the corresponding formula from the book on p. 398:

\[
\text{#tuples} = nr \times ns \times \min(\frac{1}{V(A,r)}, \frac{1}{V(A,s)}) = \\
= 1,000 \times 10,000 \times \min(\frac{1}{100}, \frac{1}{200}) = \\
= 10^7 / 200 = 50,000
\]

1.2:

10 tuples

The formula used is

\[
\text{#tuples} = \frac{nr}{V(A,r)} = \frac{1000}{100}
\]

1.3:

29.8 tuples

The formula used is

\[
\text{#tuples} = nr \times \text{selectivity}(A=15 \text{ or } B=12) = \\
= 1,000 \times (1/100 + 1/50 - 1/100 \times 1/50)
\]

Problem 2. \(r_1\) needs 800 blocks, and \(r_2\) needs 1500 blocks. Let us assume \(M\) pages of memory. If \(M > 800\), the join can easily be done in 1500 + 800 disk accesses, using even plain nested-loop join. So we consider only the case where \(M < 800\) pages.

a) nested-loop join:

Using \(r_1\) as the outer relation we need \(20000 \times 1500 + 800 = 30,000,800\) disk accesses,
If \( r_2 \) is the outer relation we need \( 45000 \times 800 + 1500 = 36,001,500 \) disk accesses.

b) Block nested–loop join:

If \( r_1 \) is the outer relation, we need \( \lceil \frac{800}{M-1} \rceil \times 1500 + 800 \) accesses, if \( r_2 \) is the outer relation we need \( \lceil \frac{1500}{M-1} \rceil \times 800 + 1500 \) disk accesses.

c) Merge-join:

Assuming that \( r_1 \) and \( r_2 \) are not initially sorted on the join key, the total sorting cost inclusive of the output is \( B_s = 1500(2 \lceil \log_{M-1}(1500/M) \rceil + 2 + 800(2 \log_{M-1}(800/M) \rceil + 2 \text{ disk accesses. Assuming all tuples with the same value for the join attributes fit in memory, the total cost is } B_s + 1500+800 \text{ disk accesses.}

d) Hash-join:

We assume no overflow occurs. Since \( r_1 \) is smaller, we use it as the build relation and \( r_2 \) as the probe relation. If \( M > 800/M \) i.e. no need for recursive partitioning, then the cost is \( 3(1500+800) = 6900 \) disk accesses, else the cost is \( 2(1500+800) \lceil \log_{M-1}(800)-1 \rceil + 1500+800 \) disk accesses.

**Problem 3.** There is a serializable schedule corresponding to the precedence graph below, since the graph is acyclic. A possible schedule is obtained by doing a topological sort, that is, T1, T2, T3, T4, T5.

**Problem 4.** a) lock and unlock instructions:

\( T_{31} : \)

- lock- S(A)
- read(A)
- lock-X(B)
- read(B)
- if A = 0
  - then B := B+1
- write (B)
- unlock (A)
- unlock(B)

\( T_{32} : \)

- lock-S(B)
- read(B)
- lock-X(A)
- read(A)
- if B = 0
  - then A :=A+1
- write(A)
- unlock(B)
- unlock(A)

b) Execution of these transactions can result in deadlock. For example, consider the following partial schedule
<table>
<thead>
<tr>
<th></th>
<th>T_{31}</th>
<th>T_{32}</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock-S(A)</td>
<td></td>
<td>lock S-(B)</td>
</tr>
<tr>
<td>Read (A)</td>
<td></td>
<td>read (B)</td>
</tr>
<tr>
<td>Lock-X(B)</td>
<td></td>
<td>Lock-X(B)</td>
</tr>
</tbody>
</table>

The transactions are now deadlocked.

**Problem 5.** Large bounding boxes tend to overlap even where the region of overlap does not contain any information. The figure 21.17 a) shows a region $R$ within which we have to locate a segment. Note that even though none of the four segments lies in $R$, due to large bounding boxes, we have to check each of the four bounding boxes to confirm this. A significant improvement is observed in the figure 21.17 b), where each segment is split into multiple pieces, each with its own bounding box. In the second case, the box $R$ is not part of the boxes indexed by the R-tree. In general, dividing a segment into smaller pieces cause the bounding boxes to be smaller and less wasteful of area.

![Figure 21.17 a) : Representation of a Segment by One Rectangle](image1)

![Figure 21.17 b) : Splitting each Segment into Four Pieces](image2)