Sample solutions

Question 1.

a) R1 Join R2 = 10,000 tuples (since C is the key, we have key-foreign key join)
(R1 Join R2) Join R3 = 10,000 tuples (same reason)

Each output tuple has 6 attributes.

b) There are a total of 3! = 6 possible plans – two of these involve cross products:

- (R1 Join R2) Join R3
- (R2 Join R1) Join R3
- (R2 Join R3) Join R1
- (R3 Join R2) Join R1

c) 1 page can hold 10 tuples of R1
=> can hold 10 tuples of R2 & (3*10)/2 = 15 tuples of R3

given 100 buffer pages. assume input/output buffers not included (for ease of computation – if you use 98 buffers, that's fine too your answer will be slightly different)

Plan 1: (R1 Join R2) Join R3

R1 Join R2 = 10,000/10 + [(10,000/10)/100]*(15,000/10)
= 16,000 I/Os

Note that there are 10,000 tuples, each 5 attributes.
=> number of tuples per page = (3*10)/5 = 6

Cost for writing out output = 10,000/6

(R1 Join R2) Join R3 = (10,000/6) + [(10,000/6)/100]*(7500/15)
= 10,167

Total = 16,000 + 10,000/6 + 10,167 (ignoring the final output cost since it is the same for all schemes).

You should compute the cost for the other 4 plans, and pick the one that is minimum.

One observation is that R3 JOIN (R1 JOIN R2) actually has a lower cost! But, this plan is not within the space of left deep trees!

Question 2.

Avoiding cross product is in general a very good heuristics because cross products are expensive. Moreover, it significantly reduces the plan space (consider R1 JOIN R2 JOIN … R20). There are many many plans that involve one, if not more, cross products (and these plans are likely to be bad plans). So, the optimization overhead can be reduced.

However, it has been shown that cross products are not always bad.

Counter-example:
Consider 3 tables:
R1(A, B): tuple size = 10 bytes, 10 tuples
R2(D, E): tuple size = 10 bytes, 10 tuples
R3(B, E, F, G, H, …): tuple size = 1000 bytes, 10000 tuples

Query: R1 JOIN R2 JOIN R3

Avoiding cross product: (R1 JOIN R3) JOIN R2 or (R2 JOIN R3) JOIN R1
With cross product: (R1 JOIN R2) JOIN R3

If R1 JOIN R3 and R2 JOIN R3 is large, then with cross product can be cheaper.

Question 3.
a. We cannot push down b+c->x, because both b and c are used in the join. Notice that if we replaced b+c by their sum before joining, we would join tuples whose sum b+c agreed, but that had different values of b and c. What we can push down is: (a) The elimination of a from R; (b) The elimination of e from S; (c) the replacement of c+d by y in S. Thus, the answer is:

$$\pi_{b+c\rightarrow x,\, y}\,(\pi_{b,\, c}\,(R) \bowtie \pi_{b,c,c+d\rightarrow y}\,(S))$$

Note that, in principle, there is no need to push down (c+d->y) since there is no savings. This is done purely because the question asks that we push down as much as possible.

b. answer can be derived in a similar manner as above.

Question 4.
Ans: In step (i), we break S into S/B sorted sublists. Step (ii) requires that we use T buffers to hold T. Step (iii) needs an additional one buffer for R, and S/B buffers for the sorted sublists of S. Thus, 1+T+S/B <= B. If we neglect 1, and multiply throughout by B, we have $B^2$ >= S+BT.