Solution

1.

a) [CARTS] + [CARTS] * [CONTENTS] = 1,000 + 1,000 * 5,000 = **5,001,000**

b) [CARTS] + [CONTENTS] + ([CARTS] –1) * ([CONTENTS] – 999)
= 1,000 + 5,000 + (999 * 4,001)
= 6000 + 4,001,000 – 4,001 = **4,002,999**

c) [CONTENTS] + $\lceil [CONTENTS] / 1{,}000 \rceil$ * [CARTS]
= 5,000 + 5 * 1,000 = **10,000**

d)
1) Hash Join, since √[CARTS] fits in memory (2 passes)
2) Sort-Merge Join, since √ [CONTENTS] Does not fit in memory (3 passes)
3) Block Nested Loops Join, 20 passes of CONTENTS!

2.

a) $\pi_{\texttt{D.dname, F.budget}}((\pi_{\texttt{E.did}}(\sigma_{\texttt{E.sal >=59000, E.hobby="yodelling"}}(E)) \bowtie \pi_{\texttt{D.did, D.dname}}(\sigma_{\texttt{D.floor = 1}}(D))) \bowtie \pi_{\texttt{F.budget, F.did}}(F))$

Notice how the above relation is derived. For each relation in the FROM list, Emp, Dept, and Finance we look for:

- Where clauses that affect only that particular relation
- Fields that can be projected out.

The goal is to reduce the tuple-set to as small as possible before doing a join.

b) If we only consider left-deep joins as System R does. Then the following 6 join plans are possible:
1. ((D E) F) ; This means we first join D and E, then join their result with F
2. ((E D) F)
3. ((D F) E)
4. ((F D) E)
5. ((E F) D)
6. ((F E) D)

Notice that we have 2 join conditions in the where clause, E.did = D.did AND D.did = F.did, which means (E join D), and (D join F) have equijoin condition, but E join F would be equivalent to a cross product (assuming we have a semi-smart optimizer that does not infer the join condition between E and F). So we toss away plans 5 and 6. The only plans we can consider are plans 1-4. For simplicity we can ignore the internal permutations between 1-2 and 3-4. So we really have 2 join orders, ((D E) F) and ((D F) E))

c)

i) For each of the query's base relations 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.

- Emp: Initial Cardinality = 50,000,
  With selection conditions E.sal >= 59,000, E.hobby = "yodelling"
  Resulting cardinality = 50000 * 1/50 * 1/200 = 5
  - The 1/50 Red. factor comes from the fact that 60000 – 59000 / 60000 – 10000 = 1/50 (from the sal >= 59000 term).
  - The 1/200 Red. Factor comes from the fact that 200 unique hobbies exist, and we want only yodelling.

- Dept: Initial card. = 5000 with selection conditions D.floor = 1
  resulting card. = 5000 * 1/2 = 2500
  - The ½ factor comes from the fact that there are 2 floors.
- Finance: Initial card. = 5000, there are no selection conditions
  resulting card. = 5000

ii) First let's consider the cost of the ((E, D) F) join order.

- The tuples from E will be pipelined, no temporary relations are created.
- First, we can use the fact that there is a B-tree index on salary to retrieve the tuples from E with salary >= 59,000. (We can't do any pre-selection with E.hobby because a B-tree on hobby does not exist.)
- We estimate that (50000/50) = 1000 such tuples selected out, with a cost of 1 tree traversal (say 3 I/Os to get to the leaf) + the cost of scanning the leaf pages (1000/200 + 1 - 1 = 5) + the cost of retrieving the 1000 tuples (since the index is unclustered each tuple is potentially 1 disk I/O) = 3 + 5 + 1000 = 1008.
- Of these 1000 retrieved tuples, do an on-the fly select out only those that have hobby = "yodelling", we estimate there will be (1000/200) = 5 such tuples.
- Now we are ready to do our first join with D.
- Pipeline these 5 tuples from E one at a time to D. By using the B-tree index on D.did and the fact the D.did is a key, we can find the matching tuples for the join by searchig the D.did B-tree and retrieving at most 1 matching tuple per tuple from E(because D.did is a key).
- The cost of E njoin D is hence total cost of index nested loop. 5*(tree traversal of D.did Btree + record retrieval) = 5*(3 + 1) = 20.
- The above step will only give at most 5 tuples, since we expected only 1 matching D tuple for each E tuple.
- Now select out the [5/2] = 3 tuples that have D.floor = 1 on the fly and pipeline it to the next level F. (This is done after E join D is done).
- Use the B-tree index on F.did and the fact that F.did is a key to retrieve at most 1 tuple for each of the 3 pipelined tuples. This cost is at most 3(3+1) = 12.
- Ignoring the cost of writing out the final result, we get a total cost of 1008 + 20 + 12 = 1040.
- If we look at the other join orders; namely - ((D E) F), ((D F) E), ((F D) E), we see that any of the DF joins will incur high initial cost. Since the D join F will suffer from their initial tuple retrieval costs of 2500 for D and 5000 for E (as calculated in part b). The remaining plan is ((D E) F). But again to select from D first, we have an initial tuple retrieval cost of 2500 I/Os.
- So ((E D) F) is the best join order.