Sample solutions

Question 1: Short questions (15 points)

a) (3 pts)
Operators can be easily connected to generate a plan since they have the same interface.

b) (3 pts)
The subplan generates intermediate results sorted on the attribute A. In addition, the attribute A is used later in the remaining plan, i.e., it’s used in a join, group by, distinct, or order by operator.

c) (3 pts)
These results could be pipelined to the next operator without going to the disk.

d) (3 pts)
(1) ((R JOIN S) JOIN T) JOIN W;
(2) (R JOIN (S JOIN T)) JOIN W;
(3) (R JOIN S) JOIN (T JOIN W).

e) (3 pts)
Approach (1) uses space more efficiently since it stores a duplicate key only once in data entries. A main drawback is that it needs to maintain overflow pages when the list of rid’s cannot fit in one page.

Question 2: B+ Tree (10 points)

a) (2 pts)
Fewer I/O’s, Fewer splits & rebalancing.

b) (4 pts)
c) (4 pts)
Assume the condition “deptID = 10 or 11” can be handled as a “range”. The plan of using 
<deptID, gpa> has fewer IOs.

Question 3: External Sort (13 points)

a) (4 pts)
Pass 0: 300 sorted runs of 31 pages each;
Pass 1: 10 sorted runs of 930 pages each;
Pass 2: 1 sorted file of 9300 pages
5 * 9300 = 46500

b) (4 pts)
2 * log(A) * (A * B * C).
Each record needs to be pushed into and popped out of the heap (priority queue), and each 
push/pop operation has a cost of log(A). If we combine the pop of the previous top element and 
the push of the new element into one step, we may get rid of the “2 *” in the formula.

c) (5 pts)
We can allocate some space in memory to store a directory of pointers pointing to each record.
Then when you sort the variable-length records, we swap the pointers instead of the actual 
record.

If the directory is small, then we need at least one buffer page. If there are too many records 
and one buffer page cannot hold that many pointers, we need more buffer pages as needed.

Question 4: Join (18 points)

a) (4 pts)
200 + 3000 * (1 + 20,000/3,000)
Assumption: we only need to access one leaf node of the Order.cid B+ tree.

b) (4 pts)
Need 1 partition pass.

\[(2 \times 2 - 1) \times (200 + 500) = 2100\]

c) (5 pts)

Suppose R is the smaller relation.

In step 1, read the records of R page by page. For each record, apply a hash function \( h \). If the hash value \( h(x) \) is 0, keep it in memory to build a hash table. Otherwise, pass it to the disk through a buffer page.

In step 2, read the records of S page by page. For each record, apply the same hash function \( h \). If the hash value \( h(x) \) is 0, do a lookup in the in-memory hash table and find matching results, which are output as results though one page buffer. Otherwise, pass it to the disk through one page buffer.

Repeat steps 1 and 2 for those passed-over records on the disk using a sequence of different hash functions, until the remaining disk records of R can fit into memory. Then load these pages into memory to build a hash table, and scan the disk pages of S to do the in-memory join.

d) (5 pts)
Suppose R is the smaller relation.

In step 1, read the records of R page by page. For each record, apply a hash function $h$. If the hash value $h(x)$ is 0, keep it in memory to build a hash table for partition $R_0$. Otherwise, pass it to the buffer page $i = h(x)$, which will eventually be flushed to the disk to generate partition $R_i$.

In step 2, read the records of S page by page. For each record, apply the same hash function $h$. If the hash value $h(x)$ is 0, do a lookup in the in-memory hash table of $R_0$ and find matching results, which are output as results though one page buffer. Otherwise, pass it to the buffer page $i = h(x)$, which will eventually be flushed to the disk to generate partition $S_i$.

Use the second step of grace join to join each $(R_i, S_i)$ partition pairs.

This join algorithm combines the idea of “keeping one partition in memory as a hash table to reduce their disk IOs” from Simple Hash Join and the idea of “partitioning those remaining records using multiple output buffers” from Grace Hash Join.

**Question 5: Distinct (12 points)**

a) (3 pts)

select count(distinct name)

from Applicant
b) (3 pts)
We need to two merge passes.
Thus, the total is $1000 + 400 \times 4 = 1000 + 1600 = 2600$

c) (3 pts)
Same to b), 2600

d) (3 pts)
For b, eliminate duplicate results during sort/merge phase.
For c, eliminate duplicate results during partitioning/building phase.

Question 6: Cost Estimation (12 points)
a) (3 pts)

```
   100
  0-10
   
   155
  10-20
   
   25
  20-30
   
   20
  30-40
```

b) (3 pts)
35

c) (3 pts)

```
   100
  0-10
   
   100
  10-15
   
   100
  16-40
```

d) (3 pts)
$\frac{4}{24} \times 100$
Question 7: System-R Optimizer (20 points)

a) (2 pts) Write the meaning of the query in plain English.

For each supplier, for all its products with a price more than $200 and sold to a California customer, compute their total price.

b) (6 pts)
- **Products(pid, pname, price)**
  1) B+ tree scan on pid, interesting order on pid; kept;
  2) B+ tree search on price for the condition “price > 200”; kept if its cost is less than the access method 1).
  3) B+ tree search on pid for a constant, kept for a later index-based join

- **Customers(cid, cname, email, state)**
  1) B+ tree scan on cid, interesting order on cid; kept;
  2) B+ tree search on cname for the condition “state=CA”; kept if its cost is less than the access method 1);
  3) B+ tree search on cid for a constant, kept for a later index-based join

c) (4 pts)
1) Products join Buys: For each access method on Products kept from the previous step, for each access method on Buys kept from the previous step, consider all possible valid join methods: block nested loop join, index-based join, sort merge join, hash join, etc.
   Estimate the cost of each subplan. Use the interesting order from the previous method, if any, when estimating the cost of a sort-merge join. For each interesting order, select the join method with the smallest cost, and remove those subplans that are dominated by another subplan in terms of both cost and interesting order(s).
2) Repeat the same step for Buys join Products;

d) (4 pts)
1) (C, CP, P, PS )
2) ( CP, P, PS, S)
3) (C, CP, PS, S)

Notice that CP and PS can still join their “pid” attributes. If your answer assumes CP and PS do not join when P is not included, we can also accept an answer that does not include “3)”.

System-R optimizer does not consider cross products. So a subset that produces a cross product is wrong.
e) (4 pts)
- Interesting orders
- Left-deep trees only
- Avoid Cartesian products
- Dynamic programming to do plan enumeration
- Selection push down
- Deal with group by at the end
- Deal with nested subqueries as blocks, and optimize them separately (not shown in the example)

Question 8: Extra-Credit Question (15 points)

a) (5 pts)
Sort the new records. Perform a full scan over the original B+ tree. Merge the results of sorted new records with the results from scanning B+ tree to bulk load a new B+ tree.

b) (5 pts)
Use a tombstone record to indicate that a key has been deleted.

c) (5 pts)
Perform a full scan over multiple B+ trees to get sorted records. During the full scan, use a priority queue to merge results so that a key can be returned at most once. If a key has been deleted (as a tombstone), we ignore that key.

For the sorted records, we bulkload a new B+ tree.