Homework 4 – Answer

11.5

Observe that the size of the output is at most 50 pages (5% are programmers who work for the production department). So, at most, a file scan (i.e., 1000 pages) will be the worst case scenario.

a. Since the only index is on *Title*, we can use it to find all *Programmers*, then select from this group those work for the production department. The index is clustered, so we need to find the first programmer (2 if the index is integrated, 3 if it is unintegrated). Then, we need to scan 100 pages (10% are programmers). As observed above, the output is at most 50 pages and can fit in the memory, we can select the programmers with E.Dept='Production' on the fly (pipelining) and keep the result in the main memory. Sorting to get 'distinct' values can also be done in the memory. Thus, we need 102 (or 103) page transfers.

b. Note that the condition in the WHERE clause of the query contains a prefix of the index, we can use the index to find the first person with E.Title='Programmer' and E.Department='Production' and then scan 50 pages (size of output). Thus, it will be the level of the index (2) plus 50 (or plus 1 if index is unintegrated). Notice that we even do not need to sort since the data is already sorted. Answer: 52 or 53 page transfers.

c. This is similar to the first case, we can use the index to find people in the 'Production' department, select the programmers, and the project the name on the fly. Thus, the cost will depend on how many pages of employees working for the 'Production' department are in the table. Since there are 10 departments, this number is at most 103 (or 104) pages.

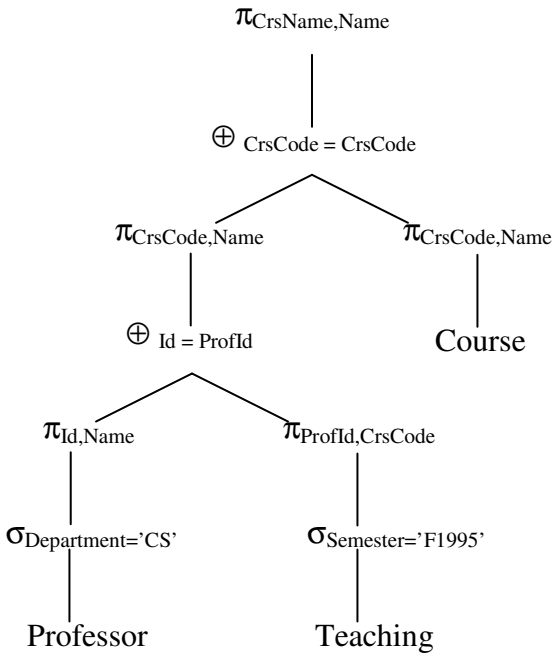
d. If we were to use the hash index on Dept, it will give us the bucket of employees working for the 'Production' department. However, the index is unclustered, so we need to get 1 page per index entry in the bucket which is around 1000 rows. So, this is worse than the file scan.

   If we were to use the Ename index, we will have to scan the whole table anyway. The advantage of this method is that we do not need to do the sorting. So, this can be use as well.

11.7

a. $\pi_{\text{CrsName,Name}}(\sigma_{\text{Semester='F1995' AND P.Department='CS'}}$
   $(\text{Professor} \oplus_{\text{Id= ProfID}} \text{Teaching} \oplus_{\text{CrsCode=C.CrsCode}} \text{Course}))$

Here, $\oplus$ stands for the join operator.



Since there are 100 pages in Professor, 2000 in Teaching, and 400 pages in Courses and with such large relations, doing straight joins is going to take many pages transfers, so we try selections first.

Selection on Department in the Professor relation can use the clustered 2-level index and yields 100/50 = 2 data pages. The cost: 2 page transfers to search the index and 2 to fetch the data pages of Professor. In total, 4 page transfers.

It is tempting to select Teaching on Semester, yielding 20000/200 = 100 rows (10 pages), but the index on Semester is unclustered, so it may take us 100 page transfers to find the relevant rows, not counting the cost of the index search.

So, instead, we will use index-nested loops to join the 2 selected pages of Professor with Teaching. Since the index on ProfId in Teaching is clustered, we can fetch all rows in Teaching that match any given row in Professor in two page transfers (20000 teaching rows per 1000 professors yields 20 rows per professor, i.e., 2 pages). For each such fetch, we need to search the B+ tree index on ProfId. We can keep the top level of that index in main memory, so there is only one index page fetch per search. Thus, the join takes 1 (to fetch the top level of index) + 20 (professors) * 3 (index search plus fetching the matching rows) = 61.

The intermediate join has 20 * 20 = 400 rows, each row twice the size of the row in either Professor or Teaching. So, only 5 rows fit in a page for a total of 80 pages. However, we can select the result of the join on Semester on the fly, reducing the size to just 2 rows, 1 page.

We could also apply projection on Name and CrsCode, but this will not give us any more gains. Since we are talking about just a 1-page result, there is no need to write it on disk – we can keep it in main memory.

Now, since Course is sorted on CrsCode, we can use binary search on Course (for each of the two rows in the join result) to find the matches. This would take us $\lceil \log_2 400 \rceil = 9$ page transfers per search, 18 page transfers in total.

Thus, the total cost is 4+61+18=83 page transfers.

Note that joining Teaching with Course first would require us to scan Course at least once (since there is no index on CrsCode in either relation). So the cost would be at least 400 pages – by far higher than the 83 pages that we came up with.