# Solution — Homework 2

The following estimations can be drawn from the information:

1. About 50 data pages are programmers who work for the production department (from: 5% . . . )
2. About 100 pages are programmers (from: 10% . . . )

This will mean roughly that we will need to screen through about 50 pages and return the different names. Since the memory is about 51 pages, we could do the screening in main memory. So, we will need to find the fastest way to get these 50 pages.

**a:** Since we only have a clustered index on TITLE, we can use this index to find the set of PROGRAMMER; this will cost us (at most) 2 page IO to get to the first tuple of programmers; thereafter, we have no other way than scan the 100 pages of programmers to find the production programmer; the memory is enough for us to sort and output the different names. So, if the size of the output is $K$ pages, we will need $102 + K$ IO operations ($2 + 100 + K$).

The query execution tree in (a) (see the accompanied file).

**b:** Similar to the above, we can use the index to search for PRODUCTION + PROGRAMMER (this is the prefix of the index and hence it is possible to do so). Thereafter, we only need to scan 50 pages to output the different names (because the index contains ENAME, we don't even need to do the sorting in the main memory). Thus, if the size of the output is $K$ pages, we will need $52 + K$ IO operations ($2 + 50 + K$).

The query execution tree is given in (b) (see the accompanied file).

**c:** Since the index is on DEPT, ENAME, TITLE, we cannot use the index like in **b**. We can use the index to search for PRODUCTION, then do the selection and projection on the fly as we have done in **a**. In the worst case, the cost is the same as in **a** plus 1 since the index is now 3-level index. However, we can do the following: we can search for the department in the index (whose size is supposely much smaller than the original data), thereafter, we can access the data and output it. Thus, the cost can be reduced to $3+$ the number of index pages containing index of production programmers + K.

The query execution tree is similar to (a) with differences in the index and the method of searching for the department, if applicable.

**d:** We could use the hash index to find the PRODUCTION department. We then search the tuples pointed to by these indices to get the programmers. Thereafter, we do the projection and selection (of different names) in memory. Since the index is unclustered, each tuple results in a page IO. So, the cost is 1.2 IO (the "magic number" of finding something using hash index) to find the bucket and 1000 for the PRODUCTION department (10 departments, 10000 tuples), i.e., we need $1002 + K$ pages IO.

If we do a direct scan (scan the data from begin to end) and do the projection and sorting in-memory, it will cost us ony $1000 + K$ pages IO.

If we use the ENAME index, it will require us to scan through the file anyway. However, we do not need to do the sorting and projection in-memory. So, this is likely the best of all three possibilities.

Since scan through seems to be the best, not much to draw for the query execution tree!