Lecture 4 exercise solutions

1.

Using the numbers from the slides: first, m = 128MiB / 16KiB = 2^{27} / 2^{14} = 2^{13}; then, $N \le M(m-1) \approx Mm$ = $2^{27} \cdot 2^{13} = 2^{40}$ = 1TiB.

2.

Printing all the keys sorted from the B^+-tree is simple. First, leftmost pointers are followed to get to the leftmost leaf and then the sibling pointers are followed to traverse all the leaves. To do the same in the B-tree, one has to perform a recursive inorder tree walk analogous to an inorder tree walk in a binary search tree. For each internal node, a recursive call and a printing of a key is repeated for each child, followed by the last recursive call (for the rightmost child).

The external-memory variant of tree-sort will be dominated by the insertions into the B-tree and it will have the running time of $\Theta(N \log_B N)$. We do not call this optimal, because an optimal external sorting algorithm (such as multiway merge sort) has the running time of $\Theta(N/B \log_{M/B} N/B) = \Theta(n \log_m n)$. Parameters such as B and M are not merely constants in our model. One of the main problems of tree-sort is that it does not use all the available main memory.

18.2-6.

The number of nodes accessed by the search is $O(\log_t n)$. The CPU cost of a binary search in a node is $O(\lg t)$. Thus the total cost is:

$$O(\log_t n \cdot \lg t) = O\left(\frac{\lg n}{\lg t} \lg t\right) = O(\lg n).$$

Note that it is the same as for a balanced binary search tree storing all n elements in main memory.

4.

The main problem with the nested-loop algorithm from the video is that it may leave the output slightly fragmented, with holes in it. As a batch of m-1 pages becomes cleaned in one iteration of the algorithm, they will have gaps where duplicates were removed. When outputting them, one has to be careful to compact them which you can do in main memory, but when you have written all of them, the last page may be left half-full. When you output the next batch of m-1 pages cleaned from the duplicates, you have to start from the middle of the half-full page, which you first have to read back to memory. Thus, it is better to keep this page in memory while it is not full, i.e., have always one page as an output buffer and write it out only when it is full. Then we would have one page as the input buffer of the inner loop, one page as the output buffer, and we would clean batches of m-2 pages.

In the video and the lecture, I give a quick analysis of the asymptotic running time. Counting precisely, the read I/Os will sum up as an arithmetic series (for simplicity, I assume m-1 divides n): $n + n\text{-}(m\text{-}1) + n\text{-}2(m\text{-}1) + \ldots + (m\text{-}1) = n(n+m\text{-}1) / 2(m\text{-}1)$. Write I/Os will add up to n. In total: $(n^2 + nm - n) / 2(m\text{-}1) + n \approx (n^2 + nm) / 2m + n$

The sorting based algorithm will do precisely $2n+2n$ I/Os for sorting and the same $4n$ I/Os for “unsorting” based on the position in the original order saved with each record before sorting. Thus $8n$ in total. Plugging

in the numbers we get ~27000 I/Os for the nested-loop algorithm, and 48000 for sorting-based algorithm. Thus, in this case, the nested-loop algorithm is faster.