Implementation of DBMS
Exercise Sheet 7
Klingemann, WS 2024 / 2025

1) Suppose that blocks can hold either three records, ten key-pointer pairs, or fifty pointers.
a) We use the indirect bucket scheme. If each search-key value appears in 10 records, how many blocks do we need to hold 3000 records and its secondary index structure? How many blocks would we need if we did not use buckets but a key-pointer pair for each record?
b) We want to retrieve all records for a particular search key. In doing so, we sequentially scan the key-pointer pairs in the index from the beginning until we find the search key value we are looking for. We assume that records with the same search key never share a block. We also assume that buckets never extend across different blocks and that key-pointer pairs with the same key are all in the same block. How many blocks do you have to inspect on average, for the two scenarios in a)?

2) We would like to sort the tuples of a relation R(A, B, C). Initially, the tuples are stored in secondary storage in a random order. At the end, they have to be in secondary storage sorted according to the values of attribute A (search key).
The following information is known about the relation.

- The relation R has 100000 tuples.
- The size of a block is 4096 bytes. Blocks have a header of 60 bytes.
- The sizes of attributes are 32 bytes for A, 200 bytes for B and 140 bytes for C. Records have a header of 28 bytes.
- Each block holding R tuples is full of as many R tuples as possible. We use unspanned storage for the records and the key-pointer pairs in c).
- A record pointer is 8 bytes.

Answer the following questions based on the information above.
a) If we use the 2PMMS sorting algorithm with two passes, what is the minimum amount of main memory (in number of blocks) required?
b) What is the cost of the two pass sorting algorithm in terms of number of I/Os?
c) Consider the following variant of the sorting algorithm. Instead of sorting the entire tuples, we just sort the pairs <key, recordPointer> for each tuple. As in the conventional two pass sorting algorithm, we sort chunks of <key, recordPointer> in main memory and write the chunks to disk. In the merge phase, <key, recordPointer> entries from different chunks are merged. The record pointers are used to recover the rest of the tuple (from the original copy of R) and write the sorted relation to the disk.
What is the cost in terms of number of I/Os?