CSE 4/560
Databases and Query Languages
Homework 5
Total Marks 100

Your Name:
Your email ID:
Your UB Person ID:

1.[10]Suppose blocks hold either three records, or ten key-pointer pairs. As a function of n, the number of records, how many blocks do we need to hold a data file and:
(a) A dense index
(b) A sparse index?

2. [10] Suppose we have B-tree nodes with room for three keys and four pointers, as in the examples of this section. Suppose also that when we split a leaf, we divide the pointers 2 and 2, while when we split an interior node, the first 3 pointers go with the first (left) node, and the last 2 pointers go with the second (right) node. We start with a leaf containing pointers to records with keys 1, 2, and 3. We then add in order, records with keys 4, 5, 6, and so on. At the insertion of what key will the B-tree first reach four levels?

3. [15] Some hash functions do not work as well as theoretically possible. Suppose that we use the hash function on integer keys i defined by $h(i) = i^2 mod B$, where B is the number of buckets. a) What is wrong with this hash function if B =10?
b) How good is this hash function if B = 16?
c) Are there values of B for which this hash function is useful?

4. [20].Suppose that blocks can hold either ten records or 99 keys and 100 pointers. Also assume that the average B-tree node is 70% full; i.e., it will have 69 keys and 70 pointers. We can use B-trees as part of several different structures. For each structure described below, determine
(i) the total number of blocks needed for a 1,000,000-record file, and
(ii) the average number of disk I/O 's to retrieve a record given its search key.
You may assume nothing is in memory initially, and the search key is the primary key for the records.
a) The data file is a sequential file, sorted on the search key, with 10 records per block. The B-tree is a dense index.
b) The same as (a), but the data file consists of records in no particular order, packed 10 to a block.
c) The same as (a), but the B-tree is a sparse index.
d) Instead of the B-tree leaves having pointers to data records, the B-tree leaves hold the records themselves. A block can hold ten records, but on average, a leaf block is 70% full; i.e., there are seven records per leaf block.
e) The data file is a sequential file, and the B-tree is a sparse index, but each primary block of the data file has one overflow block. On average, the primary block is full, and the overflow block is half full. However, records are in no particular order within a primary block and its overflow block

5.[10]In an extensible hash table with n records per block, what is the probability that an overflowing block will have to be handled recursively; i.e., all members of the block will go into the same one of the two blocks created in the split?

6.[20] Suppose keys are hashed to four-bit sequences, as in our examples of extensible and linear hashing in this section. However, also suppose that blocks can hold three records, rather than the two-record blocks of our examples. If we start with a hash table with two empty blocks (corresponding to 0 and 1), show the organization after we insert records with hashed keys:
a)0000,0001,... ,1111, and the method of hashing is extensible hashing.
b) 0000,0001,... ,1111, and the method of hashing is linear hashing with a capacity threshold of 100 %.
c) 1111,1110,..., 0000, and the method of hashing is extensible hashing.
d)1111,1110,... , 0000, and the method of hashing is linear hashing with a capacity threshold of 75 %

7. [15] Suppose we have a relation R (x ,y ,z), where the pair of attributes x and y together form the key. Attribute x ranges from 1 to 100, and y ranges from 1 to 1000. For each x there are records with 100 different values of y, and for each y there are records with 10 different values of x. Note that there are thus 10,000 records in R. We wish to use a multiple-key index that will help us to answer queries of the form:
SELECT z FROM R WHERE x = C AND y = D;
where C and D are constants. Assume that blocks can hold ten key-pointer pairs, and we wish to create dense indexes at each level, perhaps with sparse higher-level indexes above them, so that each index starts from a single block. Also assume that initially, all index and data blocks are on disk.
a) How many disk I/O 's are necessary to answer a query of the above form if the first index is on x?
b) How many disk I/O 's are necessary to answer a query of the above form if the first index is on y?
c) Suppose you were allowed to buffer 11 blocks in memory at all times. Which blocks would you choose, and would you make x or y the first index, if you wanted to minimize the number of additional disk I/O 's needed?