**CPS 216 Fall 2001**
**Homework #3**
Due: Thursday, November 8

**Please note:** *Some problems in this homework may require you to review some simple calculus and statistics.*

**Problem 1.**

Consider a secondary $B^+$-tree index. Each internal node contains $m$ child pointers, and each leaf node contains $m$ pointers to records. There are a total of $N$ records indexed by the $B^+$-tree. We wish to choose the value of $m$ that will minimize search times on a particular disk drive, given the following information:

(1) For the disk holding the index, the time to read a given block into memory is given by $(30 + 0.01m)$ milliseconds. The 30 milliseconds represent the seek time and the rotational delay of the read; the $0.01m$ milliseconds is the transfer time.
(2) Once the block is in memory, a binary search is used to find the right pointer. The time to process a block in main memory is $(a + b \log_2 m)$ milliseconds, for some constants $a$ and $b$.
(3) The main memory time constant $a$ is negligible compared to the disk seek time and rotational delay of 30 milliseconds.
(4) You can assume that the index is completely full and perfectly balanced, and that $N$ is conveniently a power of $m$.

Questions:

(a) What value of $m$ minimizes the time to search for a given record? An approximate answer is okay.
(b) What happens as the disk latency (30 milliseconds) decreases? For instance, if this constant is cut in half, how does the optimal $m$ value change?

**Problem 2.**

To build a hash index for a multi-attribute search key, we can use an approach called *partitioned hashing*. The partitioned hash function is really a list of hash functions, one for each attribute in the search key. Suppose that we wish to build a partitioned hash index on $R(A, B)$ with $2^n$ buckets numbered 0 to $2^n - 1$. In this case, the partitioned hash function consists of two hash functions $h_A$ and $h_B$. Hash function $h_A$ takes a value of $A$ as input and produces a result with $n_A$ bits, and $h_B$ takes a value of $B$ as input and produces a result with $n - n_A$ bits. The two results are concatenated together to produce the result of the partitioned hash function, which is then used to index the buckets. To locate records with $A = a$ and $B = b$, we simply go directly to the bucket numbered $h_A(a)\ h_B(b)$ (in binary).

(a) Which buckets do we have to examine in order to locate all records with $A = a$?

(b) Suppose we are given a query mix. Each query in this mix will either ask for records with a given value of $A$, or it will ask for records with a given value of $B$ (but never both). Furthermore, with probability $p$, the value of $A$ will be specified. Give a formula in terms of $n$, $n_A$, and $p$ for the expected number of buckets that must be examined to answer a random query from the mix.
(c) Find the value of $n_A$, as a function of $n$ and $p$, that minimizes the expected number of buckets examined per query.

**Problem 3.**

Recall that the number of I/O's (not counting the output cost) required to compute $R \times S$ using the version of the block-based nested-loop algorithm presented in the lecture is $B(R) + \lceil B(R)/(M-1) \rceil \cdot B(S)$, where $B(R)$ is the number of blocks taken by $R$, $B(S)$ is the number of blocks taken by $S$, and $M$ is the number of available memory blocks. Here, by turning off double buffering on $S$, we divide $B(R)$ by $M - 1$ instead of $M - 2$.

(a) Show how to modify the algorithm to require even fewer I/O's. (Hint: Consider reusing buffers.)
(b) Give a lower bound on the number of I/O's (again, not counting the output cost) required to compute $R \times S$. Justify your answer. (Hint: An acceptable answer is $B(R) + B(S)$ (why?), but you can get a much tighter bound than that.)

**Problem 4.**

Suppose that we are using a disk where the time to move the head to a block is 100 milliseconds, and it takes ½ milliseconds to read one block. In other words, it takes $k/2$ milliseconds to read $k$ consecutive blocks once the head is positioned. Suppose we want to use the basic two-pass hash join algorithm (without the hybrid hashing trick) to compute $R \bowtie S$, where $B(R) = 1000$, $B(S) = 500$, and $M = 101$. To take advantage of sequential I/O's, we want to use as few partitions as possible (assuming perfect hash functions), and read and write as many blocks as we can to consecutive positions on disk. Counting 100.5 milliseconds for a random disk I/O and $100 + k/2$ milliseconds for reading or writing $k$ consecutive blocks from or to disk:

(a) How much time do the disk I/O's take?
(b) How much time would a sort-merge join take under the same conditions, assuming that we write each sorted run to consecutive blocks on disk?