CSE-4411(A) Test #1

Sur / Last Name:
Given / First Name:
Student ID:

- **Instructor:** Parke Godfrey
- **Exam Duration:** 75 minutes
- **Term:** Fall 2005

Answer the following questions to the best of your knowledge. Be precise and be careful. The exam is closed-book and closed-notes. Write any assumptions you need to make along with your answers, whenever necessary.

There are five major questions. Points for each question and sub-question are as indicated. In total, the exam is out of 50 points.

If you need additional space for an answer, just indicate clearly where you are continuing.

Regrade Policy

- You should not write anything on the exam paper after the exam period, if it is to be considered for regrading. Write any subsequent notes on separate paper.
- Regrading should only be requested in writing. Write what you would like to be reconsidered. Note, however, that an exam accepted for regrading will be reviewed and regraded in entirety (all questions).

Grading Box	
1.	/10
2.	/10
3.	/10
4.	/10
5.	/10
Total	/50

1. (10 points) **Buffer Pool Manager.** *How popular do you want to be?* [analysis]

   Dr. Mark Dogfurry, the infamous database designer at *Fanatics-R-Us, Inc.*, has come up with a new idea to improve the performance of the buffer manager in database systems. He suggests that space for two *long integers* (say 64 bits each) be reserved on each data page. The first will be for a field called *hits*, and the second for a field called *creation*.

   When the page is created, its *creation* field is set to the current date (say measured in seconds from midnight 1 January 1970 as is done in UNIX). Thus, *creation* is a time-stamp for the page. The *hits* field is initially set to zero.

   Every time the page is pinned, its *hits* field is incremented by one. (When the page is unpinned, however, the page's *hits* field is *not* likewise decremented.) Thus the *hits* field counts how many times the page has been pinned over its lifetime.

   Dr. Dogfurry claims a page's *popularity* can be measured as

   $$pop = hits/(now - creation)$$

   where *now* is the time-stamp for the present moment.

   a. (3 point)

   b. (3 points) Design a buffer manager replacement policy that beneficially employs pages's *popularity*. Why would this likely be a good policy?

c. (4 points) List several disadvantages to Dr. Dogfurry's *popularity* scheme.

d. (3 points) Present *in brief* a modification of Dr. Dogfurry's popularity scheme that might make for a good buffer pool replacement strategy based on popularity, but would not suffer from the problems you identified in your answer to Question 1c.

2. (10 points) **General.** *Rock, scissors, paper.* [multiple choice]

   For each of the following, choose *one* best answer.

---

a. (1 point) Consider an extendible hash with 2^{10} directory slots. It has *at most* how many buckets?
   A. 1
   B. 10
   C. 11
   D. $2^{10} - 2$
   E. 2^{10}

---

b. (1 point) Consider an extendible hash with 2^{10} directory slots. It has *at least* how many buckets?
   A. 1
   B. 2
   C. 10
   D. 11
   E. 2^{10}

---

c. (1 point) Why are overflow chains in a linear hash of length one, on average?
   A. The question is nonsense. Overflow pages are not needed in a linear hash.
   B. A bucket is split when it overflows by doubling the directory.
   C. Buckets are split round robin; this has the amortized effect that overflows never get longer than one, on average.
   D. If a record is hashed to a bucket that is overflowed, the record is rehashed.
   E. Overflows are redistributed immediately.

---

d. (1 point) How many distinct search keys exist for table **T** with the five attributes A, B, C, D, and E?
   A. 1
   B. 5
   C. 32
   D. 120
   E. 325
   F. 3125

---

e. (1 point) Variable length fields mean records are variable length. This has the consequence that
   A. slot#'s cannot be determined as fixed addresses on the page, so a slot directory on each page is necessary.
   B. the buffer pool manager must support variable length frames.
   C. different records from the same table can have different numbers of fields.
   D. the different fields of the same record must be kept on different pages.
   E. B+ tree indexes are not possible for these records because index entries would be variable length.

---

f. (1 point) B+ trees are more suitable than balanced binary search trees for database indexes because
   A. it is impossible to maintain the balance of a binary tree when it is disk based rather than main-memory based.
   B. the tree is shallower because of higher fan-out.
   C. all the leaves are the same distance from the root.
   D. the tree grows by splitting up, not by adding new leaves below.
   E. binary search trees cannot accommodate composite search keys, but B+ trees can.

---

g. (1 point) Using replacement sort instead of quicksort for pass zero of the external sort algorithm has the advantage that
   A. it is faster than quicksort (even though they are the same $\mathcal{O}$-wise).
   B. it allows for block reads whereas quicksort does not.
   C. it produces runs twice as long, on average, as the use of quicksort does.
   D. it reduces the number of I/O's for pass zero (compared with using quicksort).
   E. it reduces the number of I/O's for subsequent passes (compared with using quicksort).

---

h. (1 point) A relational database system implements an external sort routine because
   A. it is significantly faster than other standard sorting routines like quicksort.
   B. it is used by tree indexes whenever a new record is inserted.
   C. it is needed to sort files too large for main memory.
   D. it can sort entirely on disk without using main memory.
   E. standard sort routines cannot sort for compound search keys.

---

i. (1 point) Page size is determined by
   A. the programmer.
   B. the operating system.
   C. the Internet.
   D. the database system architecture.
   E. the hardware.

---

j. (1 point) All the following are design assumptions made in the design of relational database systems except which of the following?
   A. Many records fit per page.
   B. Each table can fit in main memory.
   C. Tables have many records but few columns (in comparison).
   D. I/O is expensive compared with CPU operations.
   E. Main memory is volatile.

3. (10 points) **Index Mechanics.** *To hash or not to hash.* [short answer]

a. (3 points) As you know, it is not possible to have *two* clustered indexes on the same table. This is because, by design, the database system keeps a *single* copy of the data records themselves.

You are working on a database design with your boss, Dr. Dogfurry. He says that for table **T**—with attributes A, B, C, D, and E—two clustered indexes are required: A+B+C and D + B.

What can you do to effectively have two "clustered" indexes? That is, any query that would benefit from one or the other index above, *if clustered*, would be nearly as efficient to evaluate under your "fix".

b. (3 points) Consider a hash index based on linear hashing. The basic *split rule* is to make a new hash cell whenever an overflow bucket is generated.

Should this policy be modified when there are many duplicates? For instance, when an overflow page has occurred because of a new search-key entry, but the search-key value is identical to the search-key values already in the bucket, should a split be made? Why or why not?

Whenever we create a primary key constraint, e.g.,

```
alter table T add primary key (a, b);
```

the database system creates automatically an index on the key (e.g., A + B).

c. (2 points) Why does the system do this?

d. (2 points) Assume that the table will be very large (that is, it will have many records). Should the index that the system creates be a tree index or a hash index? Briefly, why?

4. (10 points) **Index Usage & I/O's.** *May I borrow an I/O or two?* [exercise]

```
select st#, name
    from Student
    where age > 24 and major = 'CompSci'
```

There are 50,000 records in **Student**. Its primary key is st#. The file that stores the records of **Student** has 20 records per page, on average.

Assume that the predicate "major = 'CompSci'" alone matches 2,000 records, the predicate "age > 24" alone matches 20,000 records, and together—as a conjunction—they match 800 records.

The following are the indexes available for the table **Student**. Each is a tree index of type alternative #2.

A. st#
- clustered
- 100 data entries are on a leaf page (on average)
- fan out: 120, depth: 2 (not counting the leaf layer)

B. major + name
- 75 data entries are on a leaf page (on average)
- fan out: 100, depth: 2 (not counting the leaf layer)

C. age + st# + name + major
- 50 data entries are on a leaf page (on average)
- fan out: 60, depth: 2 (not counting the leaf layer)

D. major + age
- 90 data entries are on a leaf page (on average)
- fan out: 120, depth: 2 (not counting the leaf layer)

For each index, state whether it can be used to evaluate the query. If so, calculate the I/O cost of using it.

Also calculate the cost of a filescan.

Which is the best solution? Index **A**, **B**, **C**, **D**, or a filescan?

(Space for the answer to Question 4.)

5. (10 points) **External Sort.** *Dog sort.* [analysis]

   Assume that $B + 1$ buffer frames are available for sorting.

   Dr. Dogfurry has designed a new external sort algorithm:

```
Phase 0 (quick sort phase):
    while there are unprocessed pages from the input file
        read in B + 1
            (or the remaining number of unprocessed pages, if less than B + 1)
            unprocessed pages
        quicksort over the records of the read-in pages
        write out the sorted run
Phase 1 (merge phase):
    while number of unprocessed runs > 1
        merge the B oldest unprocessed runs into a new run
```

   An *unprocessed* page is one that has not yet been sorted yet. An *unprocessed* run is one that has not been merged into a new run yet. Each run is timestamped. So the oldest run is the one that was first created.

   He is proud of the new algorithm because it sorts a file in just two *phases*, whereas the basic external sort algorithm may need to make more than two *passes*.

   a. (5 points) Is Dr. Dogfurry's algorithm the same as, nearly the same as, or different than, the standard external sort algorithm? Explain.

b. (5 points) The algorithm above can be said to use a *least recently made* (LRM) policy: for each merge, it chooses the B *oldest* unprocessed runs to merge.

Dr. Dogfurry claims that, actually, a *most recently made* (MRM) policy would be far better. I.e.,

merge the B *newest unprocessed* runs

Explain whether the MRM version of his algorithm is more efficient or less efficient than the standard external sort algorithm.

(Scratch space.)

(Scratch space.)

(Scratch space.)

Relax. Turn in your exam. Go home.