CSE-4411M

Assignment #1

1. (10 points) **Buffer Pool & Operations.** *We're on the same page.* [SHORT ANSWER]

a. (6 points) Consider that you have a buffer pool that can hold three pages. There are 26 pages in the database on disk, labelled as $A \ldots Z$. Consider each of the sequences of page requests (*access patterns*) below and say the number of I/Os that would be used for the **LRU**, **MRU**, and **Clock** replacement policies. (Spaces in the access patterns are only for readability.) Assume that

- you start with an empty buffer pool each time;
- for **Clock**, the frame pointer starts with frame 0 each time;
- no page is written on (made *dirty*);
- the page is pinned and immediately unpinned each time; and
- the timestamp is updated each time the page is pinned (for **LRU** and **MRU**).

access pattern	**LRU**	**MRU**	**Clock**
ABC DAB CD			
JKL MLJ NK			

b. (4 points) Consider that an operation that uses a given B+ tree index repeatedly for equality searches (*probes*); e.g., first for *aardvark*, then for *badger*, then for *capibara*, and so forth. Assume that the values for which the operation probes are in sorted order.

What replacement policy would be most beneficial for this, and why?

2. (10 points) **Index Logic.** *That doesn't make sense!* [Short Answer]

Consider the following schema.

Employee(<u>e#</u>, name, salary, d#)
  FK (d#) refs **Department**
Department(<u>d#</u>, name, location, budget)
Project(<u>d#</u>, <u>title</u>, budget, leader)
  FK (d#) refs **Department**
  FK (leader) refs **Employee** (e#)

Dr. Dogfurry says he has made the following indexes on the above schema.

- **Employee**:
  - **A.** unclustered hash index on e#, name
  - **B.** clustered tree index on d#, salary
- **Department**:
  - **C.** clustered hash index on d#
  - **D.** unclustered tree index on name, location
- **Project**:
  - **E.** clustered tree index on d#, title
  - **F.** clustered hash index on leader

---

a. (6 points) You do not trust what Dr. Dogfurry has said about the indexes. There are logical impossibilities, and indexes that do not make sense. (That is, another index would be better in *every* situation than that one.)

Identify at least three of these problems.

b. (4 points) An index is usually created by the system when a primary key is declared for a table. The reason is that integrity checking would be too expensive without an index.

   Consider a table like **Employee** with a one-field key e#. What type of index would be more appropriate for this? Why?

   What if the primary key were *composite*—consisting of multiple fields—such as title, year, studio for a table, say, **Movie**?

3. (15 points) **Indexes & Index Mechanics.** *Index Zoology.* [Exercise]

   Consider the data values *giraffe* (29), *capibara* (17), *badger* (5), *impala* (41), *elephant* (13), *aardvark* (21), *hedgehog* (29), *donkey* (25), and *fox* (33). The number after each represents its hash value using the hash function **h**.

   a. (4 points) Consider a linear hash index that has hash buckets that have room for two entries each, and that starts with a single bucket. (So initially, *zero* bits are used to find an item.)

      Show how the linear hash index would look after adding the nine entries from above (*giraffe*, *capibara*, ...) in that order using **h** as the hash function. Assume a split rule that splits once per overflow occurrence.

b. (4 points) Consider a B+ tree index of order one that initially consists of an empty root node.

   Show a possible resulting B+ tree adding the nine entries from above (*giraffe*, *capibara*, ...) in that order. (Of course, the hash values do not apply here.) Use usual split rules and redistribution.

c. (2 points) Are there any anomalies in the indexes created in 3a and 3b?
If so, what are they, and what might be done to fix them?

As the database administrator, you observe that there are many queries on the table **Student** that have an equality predicate on major (e.g., major = 'Computer Science') and a range predicate on gpa (e.g., gpa $\geq$ 6.5).

d. (3 points) What type of index—tree versus hash, clustered versus unclustered—with what search key would you suggest to create, and why?

e. (2 points) If you also knew that the queries often need to return st# (**Student**'s primary key), name, and gpa—but just those columns—and that your index had to be unclustered, what might you propose?

4. (15 points) **External Sorting.** *Do run, do run, do run.* [Analysis]

   The standard method works in passes. *Pass 0* sorts blocks of the file into runs. Subsequent passes are *merge passes*. Assume we have B buffer frames allocated for the job. Within a merge pass, each merge step merges $B-1$ runs from the previous pass to produce a new run. The procedure ends in the pass that merges the final $B-1$, or fewer, runs into a single run.

   We need not think in terms of separate merge passes, however. Rather, we can think of it just as a sequence of merge steps. The standard external sort routine can be implemented by having each merge step take the $B-1$ *oldest* runs produced—or fewer, if fewer than $B-1$ runs remain—to merge together into a new run. We can call this choice of runs to merge *least recently made*. We repeat the merge step until just one run remains.

   Dr. Datta Bas has a suggestion that he believes is an improvement over the standard external sort routine. Pass 0 is the same as before (say by quicksort). But his choice of runs to merge each time in a merge step is *most recently made*. That is, each merge step will take the most recently made $B-1$ runs—or fewer, if fewer remain—to merge.

---

   a. (5 points) Consider the buffer pool allocation B to be 5, and a file of 100 pages.

      Calculate the I/O cost to sort them by the external sort algorithm using the merge selection of *least recently made*, which is *almost* the standard method as described in the textbook.

b. (5 points) Again, consider the buffer pool allocation B to be 5, and a file of 100 pages. Calculate the I/O cost to sort them by the external sort algorithm using the merge selection of *most recently made*, Dr. Bas's suggested method.

c. (5 points) Is Dr. Bas's version more efficient or less efficient than the standard external sort routine, in general?

Explain convincingly.