Name_____

Section You Usually Attend

UNIVERSITY OF CALIFORNIA
Department of EECS, Computer Science Division

CS186 Garofalakis/Hellerstein
Fall 2005 Midterm Exam

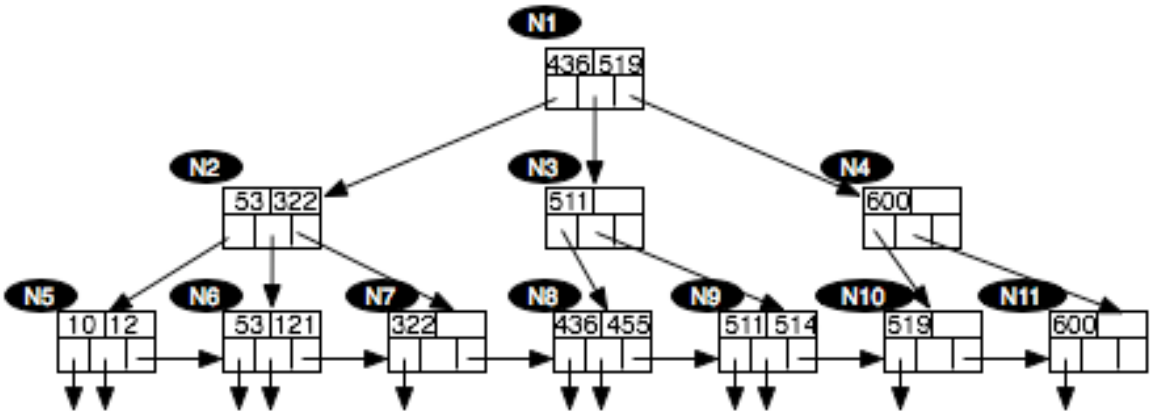
Midterm Exam: Introduction to Database Systems

This exam has five problems and one extra credit question, worth different amounts of points each. Each problem is made up of multiple questions. You should read through the exam quickly and plan your time-management accordingly. Before beginning to answer a problem, be sure to read it carefully and to *answer all parts of every problem!*

You **must** write your answers on the exam. Two pages of extra answer space have been provided at the back in case you run out of space while answering. If you run out of space, be sure to make a "forward reference" to the page number where your answer continues.

Good luck!

1. B-trees [9 points]
Consider the B+ tree of order $n = 2$ (two values, three pointers) shown in the Figure. The nodes are labeled by ovals from N1 to N11.

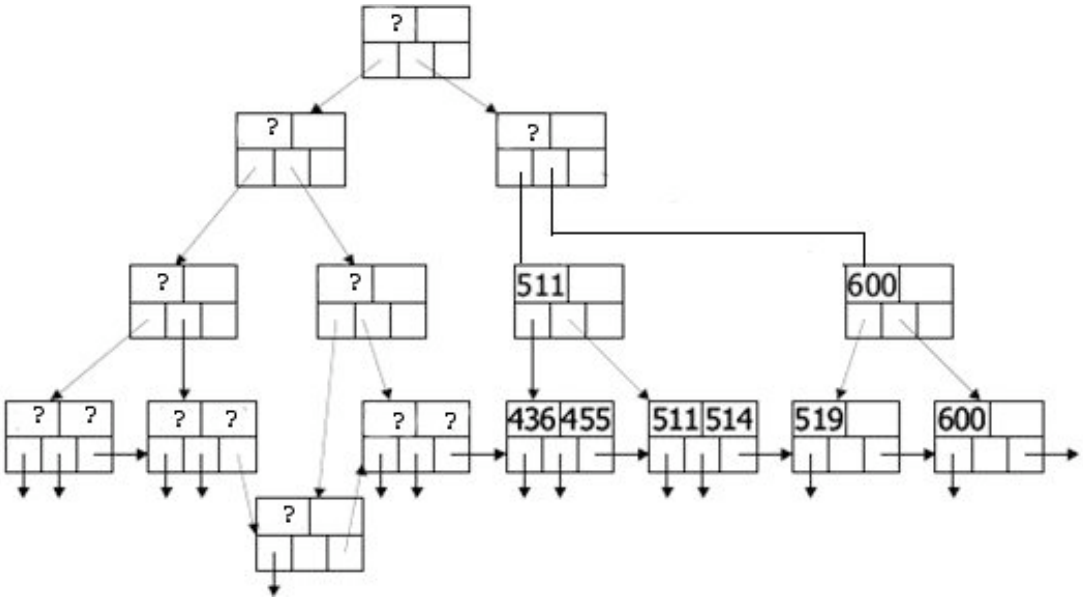


A. [3 points] In the space below, draw a new version of the node(s) that would change after the insertion of the key 435. Be sure to identify nodes by their labels.

Name_____

B. [6 points] Shown below is the B+ tree after the insertion of the key 57 in the B+ Tree from part (B), i.e., the B+ Tree formed after inserting both 435 and then 57 into the tree of part (A). Fill in the keys shown as '?' – be sure to make them legible!

(Note: Assume that while reallocating an odd number of index entries after a node splits, the ***left*** node of the split gets one more entry than the right.)



Name_____

2. Query Languages [24 points]

The following is part of the schema for the blog software used on the CS186 home page (in fact, the SQL DDL is a subset of what is in the blog's source code!) The XML DTD on the right is a rough translation of the SQL schema. Please use this information to answer the questions below.

SQL DDL

```
create table entries (
  id integer primary key,
  title varchar(200),
  timestamp integer,
  body text,
  authorid integer references authors
);

create table authors (
  realname varchar(255),
  username varchar(20),
  authorid integer primary key
);

create table comments (
  id integer primary key,
  entry_id integer references entries,
  parent_id integer references comments,
  timestamp integer,
  com_title varchar(150),
  com_author varchar(80),
  body text
);
```

XML DTD

```
<!ELEMENT entrylist (entry*)>
<!ELEMENT entry (body, author, timestamp,
      comment*)>
<!ATTLIST entry type CDATA "title">

<!ELEMENT body(#PCDATA)>

<!ELEMENT author (#PCDATA)>
<!ATTLIST author type CDATA "username")>

<!ELEMENT timestamp(#PCDATA)>

<!ELEMENT comment(body, timestamp,
      comment*)>
<!ATTLIST comment type CDATA
      "com_author">
<!ATTLIST comment type CDATA "com_title">
```

A. SQL

a. [4 points] Find blog entries posted by an author with realname 'Ted', and return the title, timestamp and body of the entry.

b. [6 points] For each entry in the blog, return the same fields as in part (a), for those entries with more than 2 comments.

c. [6 points] The "parent_id" field in the comments table tracks the nesting of "comments on comments": when a new comment is posted in response to an old comment, the parent_id of the new comment is the id of the old comment. (You may assume that comments made directly on blog entries have parent_id = 0).

Find the id of each comment and the id of its "grandparent" comment; if it does not have a grandparent, omit it from the answer.

Name_____

B. XML

a. [4 points] Write an XPATH query that returns all comments associated with entries entitled “Midterm”.

```
/entry[@title="Midterm"]/comment
```

b. [4 points] Write an XPATH query to find the “grandparent” comment for each (nested) comment; if the comment does not have a grandparent, omit it from the answer.

```
/entry/comment/comment//comment/../..
```

Name_____

3. Sorting. [8 points]
We would like to sort the tuples of a relation R(column1, column2, column3, column4) on a the sort key (column1, column2, column4). The following information is known about the relation.

· The relation R contains 100,000 tuples.
· The size of a block on disk is 4000 bytes.
· The size of each R tuple is 400 bytes.
· The size of each field in R is 4 bytes.
· A record pointer is 4 bytes.

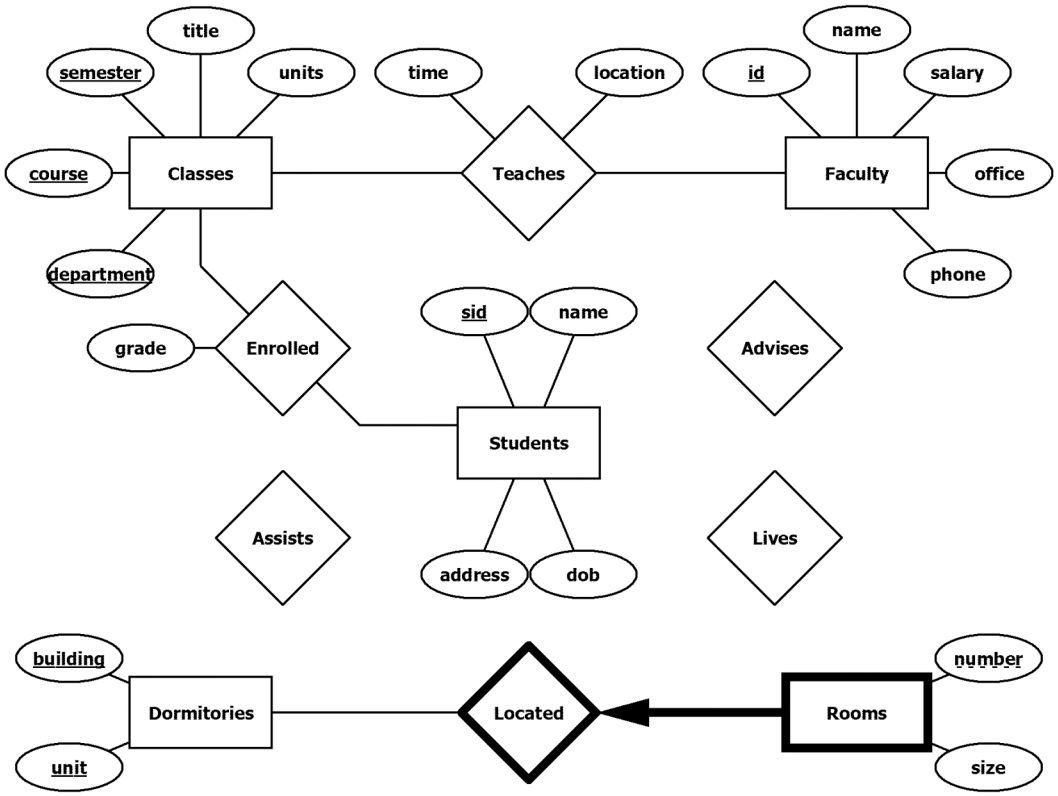
Answer the following questions based on the information above.

A. [4 points] If we want to sort in two passes (using only Phase 0 and Phase 1), we need to know the minimum number of blocks B of main memory required. Provide (i) a formula for computing B correctly, and (ii) also give an *integer* value of B that guarantees a 2-pass sort.

B. [4 points] Assume we have sufficient memory to perform the sort in two passes. What is the cost, in terms of number of disk I/Os, of sorting relation R? Include the cost of the writing the sorted file to the disk in the end in your calculations.

Name_____

4. Entity-Relationship Modeling [12 points]



A. [6 points] Make additions or changes to the entity-relationship diagram shown above to reflect the following, labelling the changes with the corresponding letter:

[a] Students are not required to live in the dorms; but they are assigned to at most one room if they do. To ensure that living space does not go unused, it is required that every room be assigned to a student.

[b] Each building in a dormitory unit has exactly one student assigned to it as a "resident assistant".

[c] Every student must be assigned to some faculty member for advising; a student may have more than one advisor, for instance, if the student studies in multiple departments.

B. [6 points] Consider moving the attribute "semester" out of the entity "Classes" and separately into both of the relations "Teaches" and "Enrolled". What would the effect of this change (in terms of what will be representable) be on "Classes", "Teaches", and "Enrolled", if any?

Teaches:

Classes:

Enrolled:

Name_____

5. Query Optimization and Selectivity Estimation [14 points]
You are working for *Poodle Inc.*, a hi-tech company that employs relational database technology to revolutionize web search and information discovery. *Poodle Inc.* uses the following database schema to store web-page information collected through their web crawler:

webpages(*url*: text, *author*: text, *content*: text, *pagesize*: integer)
links(*fromUrl*: text, *toUrl*: text)

(Underlining denotes key attribute(s); *fromUrl*, *toUr*l are foreign-key references to the **webpages** relation.)

The latest crawl has loaded the **webpages** table with 10^6 tuples in 10^5 disk blocks, and the **links** table with 10^7 tuples in 10^4 blocks. The value of the *pagesize* field of **webpages** ranges between 0 -- 20,000 (bytes), and the number of distinct page *author*s is 10^4.

(a) [6 points] Consider the following query for identifying the URLs of **webpages** that are pointed to by "small" **webpages** authored by John Doe:

```
SELECT  L.toUrl
  FROM  webpages P, links L
 WHERE  P.url = L.fromUrl
   AND  P.pagesize < 5000
   AND  P.author = "JOHN DOE"
```

Under some reasonable assumptions, give an estimate for the selectivity of this query. Be sure to explicitly state your assumptions.

(b) [8 points] Now, assume that you want to retrieve all **webpages** with *pagesize* of more than *K* bytes authored by authors with names beginning with the letter *N or higher*, using the following SQL query:

```
SELECT  *
  FROM  webpages P
 WHERE  P.pagesize > K  AND P.author >= "N%"
```

Other than a complete file scan of **webpages**, you also have the option of using an *unclustered* B+-tree index on the *pagesize* attribute. The height of this B+tree (from root to data-entry nodes) is 3 levels. Assuming *zero benefit from buffering* of data pages during the unclustered index scan, and making similar assumptions to (a), determine the "crossover" point for *K* – that is, the value of *K* at which using the unclustered index becomes better/worse than the file scan (in terms of the number of page IOs).

Extra Credit [8 points]

Consider a query joining *n* different relations, and a system with only one join algorithm. Compute the number of distinct possible left-deep and bushy query plans for this query. Consider the *complete space* that includes possible cross products. (For the case of bushy plans, you don't need to give a closed-form formula – a recurrence relation will do.)

Name_____

Name_____