CS222/CS122C – Principles of Data Management
Fall 2012, Department of Computer Science, UC Irvine
Midterm Exam
Max. Points: 100

Read the following instructions carefully:

- Total time for the exam is **75** minutes. No extra time will be awarded, so budget your time accordingly. The exam is closed books and closed notes.
- Please be succinct in your answers.
- If you find ambiguities in a question or in the schema, write down the interpretation you are taking, and then answer the question based on your interpretation.
- This exam contains **9** pages. You can use the last page as scratch paper.

NAME: **Student ID:**

Part	Points	Score
1	30	
2	10	
3	15	
4	10	
5	15	
6	20	
Total	100	

Problem 1: Short Questions (30 points)

(1) (**4 pts**) Why do database systems allow only one clustered index on a table?

With a clustered index, the data has to be sorted by the index key. One copy of the data cannot have multiple orders.

(2) (**6 pts**) Give an example of a database operation to show that the LRU replacement policy is bad for the buffer manager of a database system.
Nested loop join. If the inner relation has M pages but the buffer only has M-1 pages.

(3) (**5 pts**) In our project 2, one requirement (for extra credits) is to implement the following functions without touching the existing records in a table.

RC dropAttribute(…);
RC addAttribute(…);

Briefly describe a correct way to implement them.

Use schema versioning. In each record, there is a field to indicate the schema version. DropAttribute and addAttribute only update the schema and add a new schema version.

(4) (**5 pts**) What is the role of the buffer manager in a DBMS? Why it is important?

In a DBMS, the buffer manager reads data from persistent storage into memory as well as writes data from memory into persistent storage. The main goal of the buffer manager is the minimization of physical disk I/Os since the cost of reading a page is very expensive compared to reading a page from the memory. Thus, we prefer to maintain a buffer of pages in memory to cache some pages for faster access.

(5) (**5 pts**) What's the main difference between an ISAM index and a B+ tree?
ISAM:

- Only leaf pages modified; overflow pages needed.
- Overflow chains can degrade performance unless the size of data set and data distribution stay constant.

B+ tree is a dynamic structure.

- Inserts and deletes are handled gracefully by leaving the tree balanced.

(6) (**5 pts**) Describe three ways to manage the space of pages within a record file.

1. Double linked list. Maintain a linked list of page descriptors for free pages, and also a linked list of page descriptors for full pages.

2. Bitmap. Keep a bitmap, which uses a bit for each page (for example, 0 means full, 1 means to have free space).

3. Directory. Keep a directory to store space info and page pointers.

Problem 2 (10 Points)

a) (**4 pts**) What times does the time of each **random** disk IO include?

Seek time + Rotational delay + Transfer time

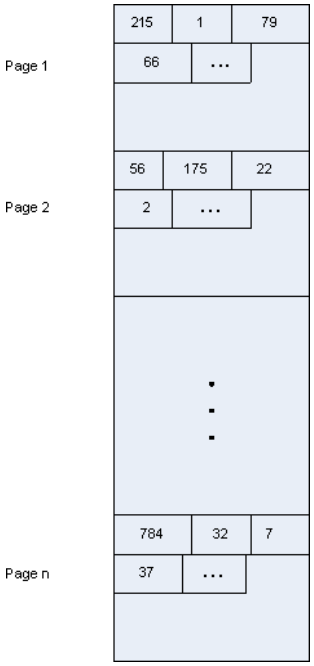
b) (**6 pts**) When we analyze the cost of a database operation (e.g., the time of inserting a record to a random heap file), why do we care about the type of each disk IO (random IO versus sequential IO)?

The seek time is the dominant cost of a disk IO cost. In sequential disk IO, we only have to do one seek and thus we pay the cost only once. While in random disk IO, we have to seek every time we access a new page. Thus, the cost of doing a sequential IO is much cheaper than doing a random IO.

Problem 3 (15 Points) In class we covered three file organizations for records. For each of them, explain its main idea using a diagram that includes pages of records with keys. Also discuss its advantages and disadvantages.

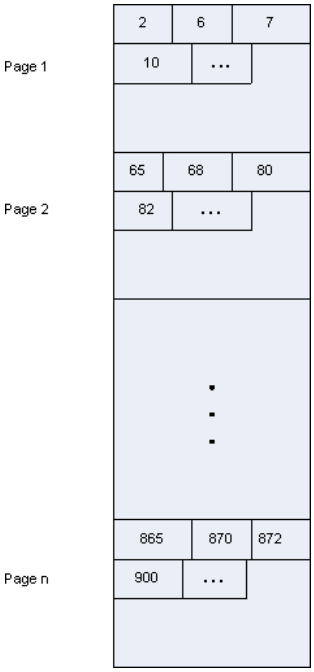
a) (**5 pts**) Random heap file

In a heap file, records are appended to the file without considering any order. Therefore, insert operation is very fast. On the other hand, point search, range search, and delete operations are going to be costly operations, since we have to scan the file.



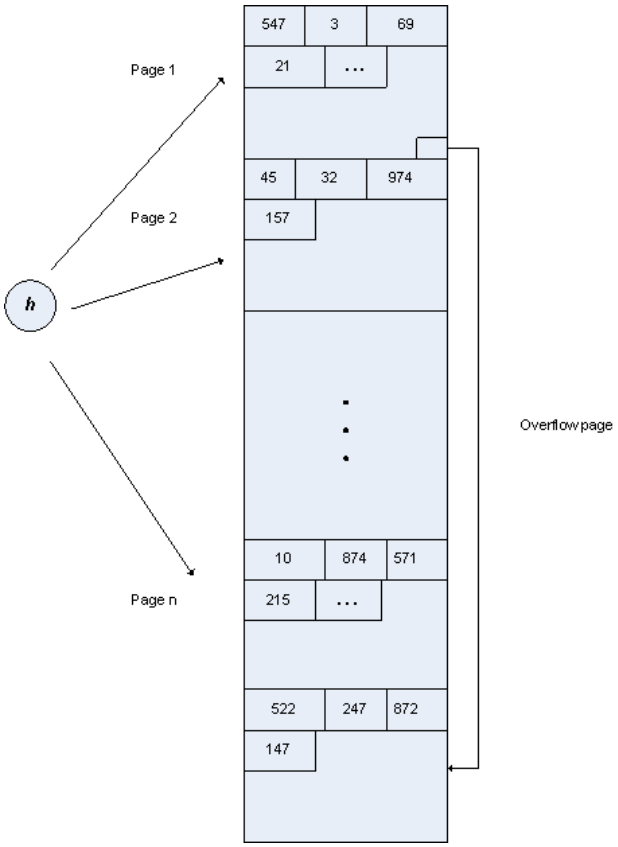
b) (**5 pts**) Ordered sequential file

In ordered sequential file, the records are sorted based on the key. Whenever we want to insert a record, we must do a binary search to find the correct location of that record and also do a shift operation to prepare enough space for the new record. Delete operation has similar cost to insert. On the other hand, point and range search operations can be done very efficiently without the need to scan the file by doing a binary search.



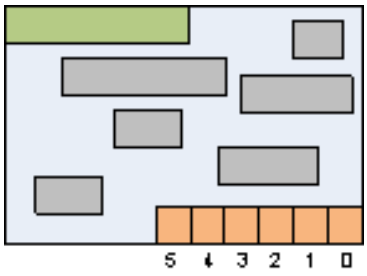
c) (**5 pts**) Hashed file

In a hashed file, we apply a hash function on key of the record to be inserted to determine the page that will be used for insertion. Note that we might have an overflow pages connected to some pages. Insert, delete, point search can be done very efficiently. For scan operation we might read more pages than a heap or sorted file since some pages can be empty. Range search is very slow since we must scan the hashed file.

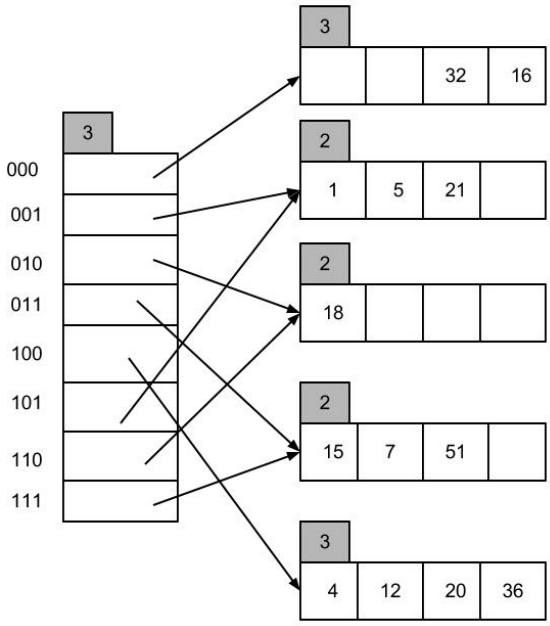


Problem 4 (10 Points) Draw a diagram to explain a suitable on-disk layout for a page of variable-size records that is maintained in a random heap file. Be sure to indicate what control information is stored per page and say what an RID would consist of.

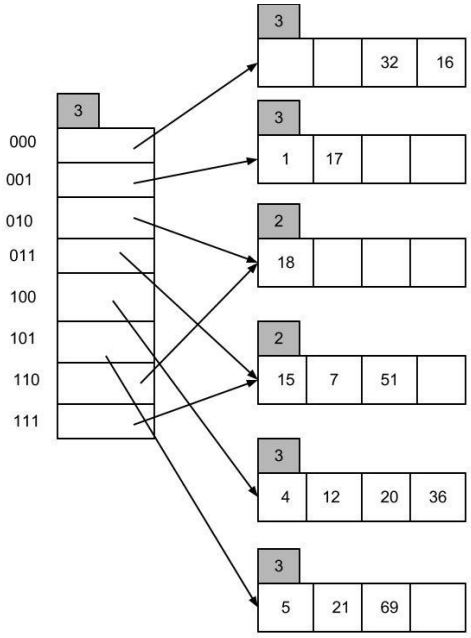
- The page header will store the following information:
  - Number of records (slots) in the page.
  - Free space offset.
  - Total free space in the page (good to have).
- For each record we must know its length in order to be able to read it from the page, we can maintain the record's length in its slot or as a metadata at the beginning of the record.
- The RID would consist of pageNo/slotNo.



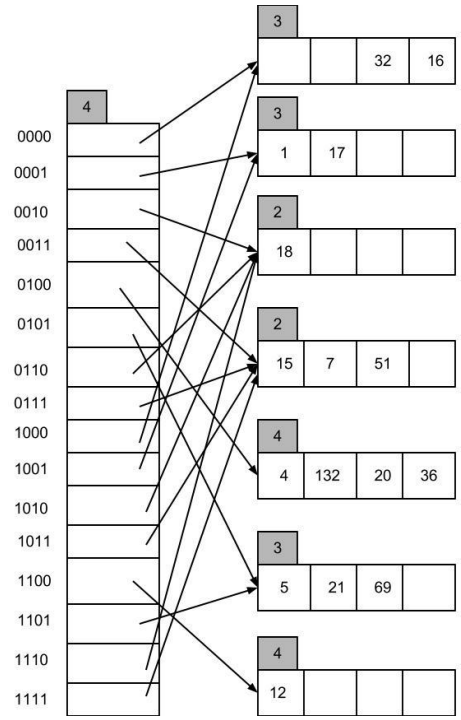
Problem 5 (15 points) Consider the extendible hashing index in the following figure.



(a) **(8 points)** Show the index after inserting entries with hash values 17 and 69.



(b) **(7 points)** Show the index after inserting an entry with hash value 132.



Problem 6 (20 points) We are building an index for an application with 2^{25} records. Records may have duplicate keys. We are considering two options.

(i) A standard B+ tree with a fanout (i.e., "2*d") n = 2^7. (The records themselves are stored separately from the index.) Nothing special is done for duplicate keys, e.g., if a key appears twice in the file, that key will appear twice in leaf node(s), each with a pointer to one of the records.

(ii) A B+ tree with a fanout (i.e., "2*d") n = 2^7 where keys appear only once. The pointer associated with key *K* points to a bucket that contains all the pointers to the records with key *K*. These pointers are stored contiguously on disk, so they can be read with a *single* IO, no matter how many pointers there are for *K*. (The records themselves are stored separately from the buckets.)

Suppose both trees are as full as allowed by *n*, and the levels close to the leaf nodes are as full as possible. In addition, assume that the root of the B+ tree always resides in main memory, and that none of the other disk structures are buffered in memory when a search starts. Also, in evaluating the cost of a query, ignore the IOs needed to fetch the records themselves, as this cost will be the same for either option.

(1) **(5 points)** For Option (i), assuming no duplicate records, how many IOs are necessary for finding (the pointer to) a record given its key?

Since records are packed as much as possible in leaf pages, the tree will have height of 4. Since the root page is cached, only 3 Disk IOs are needed.

(2) **(5 points)** For option (i), assuming that each record has exactly 2^8 duplicates, how many IOs are necessary for finding (the pointers to) *all* the records with a given key? Given a lower and an upper bound for this number.

There are exactly 2^8 duplicates for every record; thus, all duplicates records of the same key can be maintained in exactly 2 leaf pages. This means that the total disk IOs for both the best and worst cases is 4 disk IOs (root page is cached).

(3) **(5 points)** For Option (ii), assuming no duplicate records, how many IOs are necessary for finding (the pointer to) a record given its key?

Same reasoning for question (1) applies here. The only difference is that we need one additional disk IO to go to the block that has the pointer. So 4 disk IOs are needed.

(4) **(5 points)** For option (ii), assuming that each record has exactly 2^8 duplicates, how many IOs are necessary for finding (the pointers to) *all* the records with a given key?

Since there are too many duplicates in the tree, option (ii) will excel in this case. The tree will have a height of 3 instead of 4. To retrieve all the duplicates for a single key K, we need only 3 disk IOs (2 disk IOs to traverse the tree, and one to go the block that has the pointers).

You may use this page as your scratch paper.