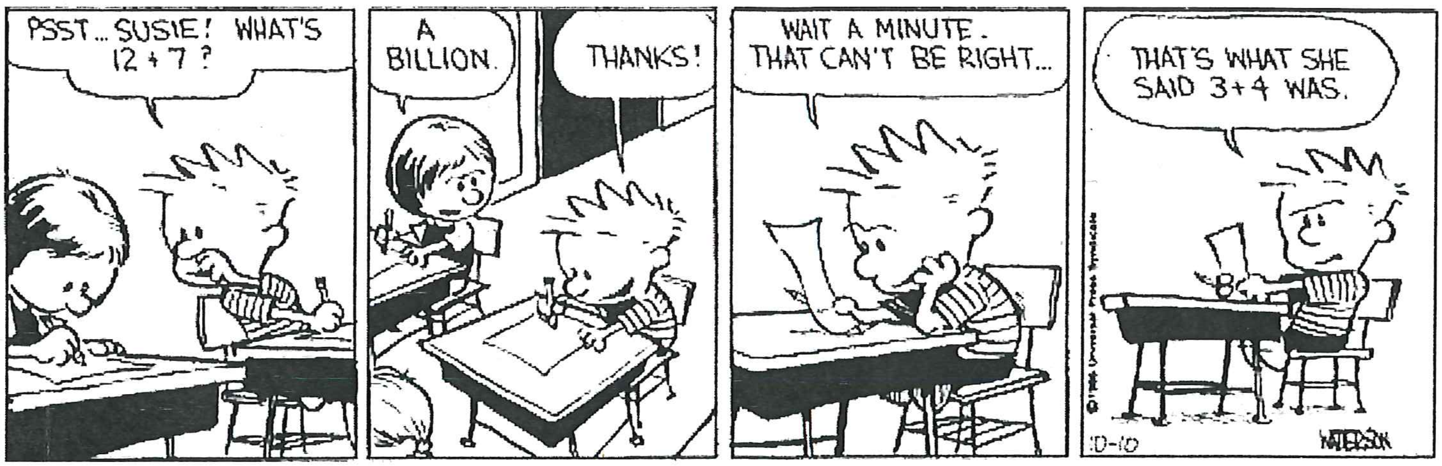
CS 564: Midterm Exam, Spring 2017

Please *print* your name below.

First Name:	Kirthanaa
Last Name:	Raghuraman

Instructions:

1. This is a closed book exam.
2. You have 75 minutes to complete this exam. The points on this exam total to **75**.



(For Instructor's Use)

Q1: Short Questions	**15** points	15
Q2: Join Processing	**20** points	17
Q3: Indexing and Sorting	**30** points	30
Q4: BadgerDB	**10** points	10
TOTAL	**75** points	73

+1

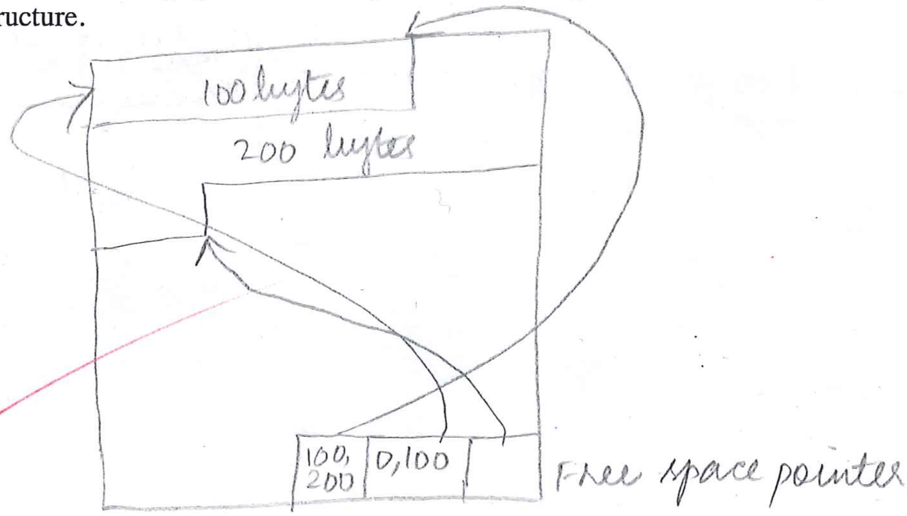
Question 1. [15 points] Short Questions

a. **[2 points]** In MapReduce, there are two steps, which can be written as:
Map: {(k1, v1)} → {(k2, v2)} // for every key-value pair in the input, output 0 or more key-values
Reduce: {(k2, [list-values-with-key-k2])} → <k3, v3> // Reduce to a final result, which is also key-value pairs

You have also learnt about relational operations like <u>selection</u>, <u>projection</u>, <u>join</u> and <u>aggregation</u>. Describe the Reduce operation in terms of the relational operations above.

Reduce operation - Aggregation ie, 'group by' all the value with same key 'k2' and return a value by aggregating the list of values with key 'k2'.

b. **[5 points]** Consider a slotted page organization with two records. One of size 100 bytes and the other of size 200 bytes. Assume pages are 1024 bytes. Draw a diagram to show the layout of the records and the slotted directory structure.



c. **[4 points]** Imagine that you have a hard disk drive that can be operated in one of two modes. In "Mode A," you can double the rotational speed of the disk, and in "Mode B," you can reduce the seek time by half. If your workload consisted of purely random I/Os, then which mode would you operate the disk in for highest performance? Why?

Your answer: ☐ Mode A ☑ Mode B (1-20ms)

Explain your answer: Seek time is the most dominant and time consuming than rotational speed. For random I/Os, seek time becomes higher and hence I would operate disk in Mode B.

d. **[4 points]** Mark True/False for the following:

Consider a B+-Tree index on attributes <a, b> (i.e. this is a B+-Tree index with a composite key).

The predicate "a = 10" matches the B+-Tree index	☑ True	☐ False
The predicate "a > 10 and b < 10" matches the B+-Tree index	☑ True	☐ False

Now consider a Hash index on attributes <c, d> (i.e. this is a Hash index with a composite key).

The predicate "c = 10 and d = 10" matches the Hash index	☑ True	☐ False
The predicate "c = 10" matches the Hash index	☐ True	☑ False

Question 2. [20 points] Join Algorithm

a) **[12 points]** Consider the following SQL equijoin query: SELECT * FROM R, S WHERE R.a = S.a
You can assume that both R and S have the same cardinality and both R and S have the same schema (i.e. their tuple sizes are similar). For each of the following, list the join algorithm that is likely to be the most efficient and why?

Clustered B+Tree index on R.a	Algorithm that you would use: Index Nested Loop Join Why? Since there is a B+tree index on R.a, we can probe this index for every tuple in S to check if there exists a value R.a = S.a. Since the tree is clustered, I can choose an Index nested loop join or even a block index nested loop join, if we want results in sorted order.
Clustered B+Tree index on R.a and a Clustered B+-Tree on S.a	Algorithm that you would use: Scan and do 'intersect' on both the lists Why? Similar to 'Sort Merge join' Since both the indexes are clustered, it is enough if we scan the leaf nodes from left to right in both trees and in merge step, keep track of equality & O/P records that match.
Unclustered B+Tree index on R.a 3	Algorithm that you would use: Block Index Nested Loop join SMJ or HJ Why? Since the index is unclustered, using a block index nested loop join is better as the records in table S are sorted among each block that is fetched. This helps in reducing the number of I/Os.

b) **[5 points]** Consider joining two relations X and Y using B buffer pool pages. Assume |X| > |Y|. Derive the minimum number of buffer pages required to join two relations using the sort-merge join algorithm in two passes.

No of sorted runs for X $\Rightarrow \frac{|X|}{2B}$ B = no of pages in buffer pool

$|||^{ly}$, no of sorted runs for Y = $\frac{|Y|}{2B}$

Both these should fit in buffer pool for merging.

$\frac{|X|}{2B} + \frac{|Y|}{2B} < B$ $\Rightarrow$ Since |X| is bigger relation, replace |Y| with |X|

$\frac{\cancel{2}|X|}{\cancel{2}B} < B$ $|X| < B^2 \Rightarrow \boxed{B > \sqrt{|X|}}$

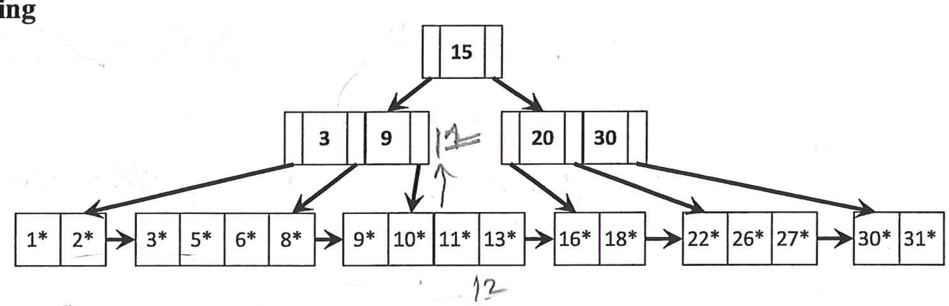
c) **[3 points]** Again, consider joining two relations X and Y using B buffer pool pages. Assume |X| > |Y|. Under some circumstances, a block nested loops join algorithm will be just as efficient as a hash join algorithm. Precisely state the mathematical criteria for when this happens.

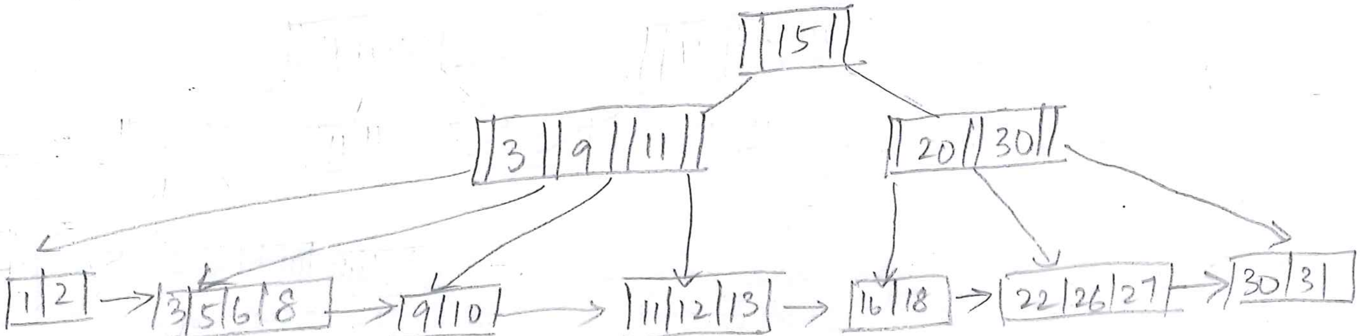
When the smaller relation Y's hash table can fit into the buffer pool, Block nested loop = hash join

ie. $B > \sqrt{F * |Y|}$

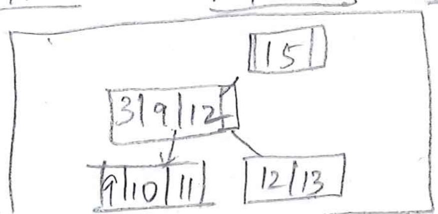
Question 3. [30 points] Indexing and Sorting

a) **[7 points]** Consider the B+-tree shown on the right with order 2 (i.e. the maximum number of keys in any node is 4). Insert an entry with the key **12** in the B+-tree. Assume that the insert **favors splitting** the node when it becomes full. Show the final B+-tree.





or



b) **[2 points]** In a B+-Tree, generally the maximum number of entries in a leaf node is more than the maximum number of entries in a non-leaf node.

Your answer: ☐ True ☒ False

Brief explanation: The maximum number of entries in leaf node is usually lesser than non-leaf node as leaf nodes store <RID, key> values while non-leaf nodes store <key, page no>, RIDs are longer than page no and hence more bytes will be required to store <key, RID> than <key, page No>.

c) **[6 points]** Consider a table with an attribute that has 33 distinct values. (You can assume that there are no null values.) Queries with an equality predicate on this attribute are very common on this table, so building a bit-based index is important for performance. Which type of bit-based index would you build on this attribute? Why?

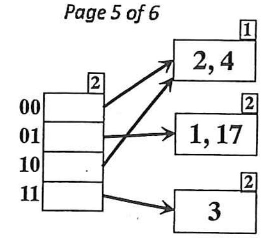
Your answer: ☒ Bitmap ☐ Bit-Sliced

Brief explanation: I would prefer bitmap over bit sliced since queries with equality predicate are common and doing this would be easier in bit map than bit slice index. In the bitmap index, there will be 34 indexes for the attribute (33 value +1 Null index). To check for equality, we just need to access the index for that particular value. Computing RID from bit position is easier in bitmap. (Assumption: discrete valued attribute).

2 | 17
2 | 8 - 1
2 | 4 - 0
2 | 2 - 0
1 - 0

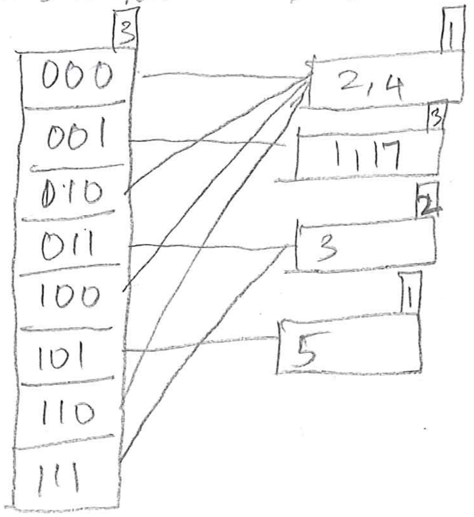
10001

d) **[7 points]** Consider the extendible hashed index shown on the right. Assume that the hash function is h (K) = K mod 32, and that 2 records fit on a page. Only the keys are shown in the bucket pages. Redraw the figure shown above to show the index structure after inserting a record with key 5. In case you need it $5 = 101_2$ (i.e. the binary form for 5 is 101).



5

Add 5 to bucket ⇒ Bucket full - split



e) **[8 points]** Consider sorting a file with 1024 pages and using replacement/tournament sort in the first pass (one that produces sorted runs).

i) Complete the following sentence: If the tournament sort has is allowed 64 buffer pool pages to hold it its data structure, at the end of the first pass, there are ___8___ runs, each on average of size ___128___ pages.

$\frac{1024}{64 \times 2} = \frac{2^{10}}{2^7}$ $\qquad N/2B \qquad 2B$

ii) What is the smallest number of buffer pool pages that can allow this file to be sorted in 2 passes, i.e. after the first pass, the second pass (i.e. the merge pass) produces a final fully-sorted file. Show how you derive your answer.

After first pass, we have $\frac{N}{2B}$ sorted runs to merge.

To sort in 2 passes, in second pass all the $\frac{N}{2B}$ sorted runs must fit in buffer pool. with B-1 pages (1 for output)

$$\Rightarrow \frac{N}{2B} < B - 1$$

$$N < (B-1)(2B)$$

$$N < B^2$$

$$\Rightarrow \boxed{B > \sqrt{N}}$$

Question 4. [10 points] BadgerDB

a) **[3 points]** In BadgerDB's buffer manager, how many times can we pin the same page in memory (without any interleaving unpin calls)? Explain.

We just need to pin one page once. As long as there are no un-pin calls, the page will already be in buffer pool but for each request for the page, the pin count is incremented.

b) **[2 points]** In BadgerDB, if the buffer pool has M frames, what is the maximum number of pages that can be pinned at any given instant?

M

c) **[5 points]** Your friend, Joe, suggests making the following modification to the BagderDB buffer manager replacement algorithm: When the replacement algorithm encounters a page that is marked as dirty, *always* skip that frame (i.e. don't use that frame to bring in a new page). Do you think Joe's idea will work? Explain your answer.

No, it wont, in some cases.

If all the pages in the buffer pool are marked dirty, then there will be no way we can select a replacement frame.

[Extra Credit] [2 points] Consider counting the number of bits in in a bitmap index that has 1024 bits, efficiently. Let **b** be a void* pointer that points to this bitmap in memory. Write the C++ code to count the number of bits that are on (i.e. set to 1) in the bitmap **b**, when you have at most 1MB of additional memory to use. You can assume that you are given a magic function **InitiateCountMap** that takes as input the # bits (**k**), and returns an array of 16 bit unsigned integer values with each array entry corresponding to the number of ones in the binary representation of the corresponding array index. The first few entries of this array are [0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2...]

```
unsigned int count = 0, *arr = InitiateCountMap(8);
    for(i = 0; i < 1024/8 ; i++8) //For each byte
{
    int val = 0;
      for (j=7; j<=0; j--) //Traverse byte by byte
        val += pow(2, j) * bitmap[i*8+(7-j)];
    count += arr[val];
}
return count;
```