CS 245
Midterm Exam – Winter 2017

This exam is open book and notes. You can use a calculator and your laptop to access course notes and videos (but not to communicate with other people). You have 75 minutes to complete the exam.

Print your name: _____

Remember: you can get one point of extra credit until Tuesday night by filling out the midterm course survey on Piazza!

The Honor Code is an undertaking of the students, individually and collectively:

1. that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;
2. that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.

While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

I acknowledge and accept the Honor Code.

Signed: _____

Problem	Points	Maximum
1		10
2		10
3		10
4		10
5		10
6		10
Total		60

Problem 1 (10 points)

Consider a hard disk that has 500 GB of hard disk space. It has the following performance characteristics:

- 5000 RPM rotation rate
- 200 cylinders, numbered from 1 to 200
- Takes $1 + (\tau/20)$ milliseconds to move heads across τ cylinders (e.g., from i to $i + \tau$).
- A constant transfer rate of 100MB/s on all tracks.

(a) Calculate the average time taken to read a 1MB block from the hard disk. Assume that, for an average read, the head travels half the cylinders and half a revolution. (A simpler assumption than what we discussed in class.)

1+((200/2)/20) = 6ms to move to the right cylinder
(0.5 rev/5000rev)*60000ms = 6 ms to move to right track
1/100 seconds to read 1MB = 10ms
Total: 22ms

Average read time for hard disk (ms):_____

(b) Consider a scenario where we use a single buffer to read 1MB blocks from this hard disk and process the blocks. Assume it takes 15 milliseconds to process a single 1MB block and the time it takes to read the block from disk is given from part a. Estimate the amount of time it takes to process 10 1MB blocks.

From above,221ms to read each block.
15 ms to process it.
(22+15)*10 = 370ms.

Time to read and process 10 1MB blocks (ms):_____

(c) Consider an identical scenario as in part b, but suppose we use two buffers to improve performance. Estimate the amount of time it takes to process 10 1MB blocks.

First block: 22ms to read.
Then, can begin fetching the next block using the second buffer.

By the time we finish processing, 15ms will have elapsed, and our read of the next block will be 7ms away from completion.
So, 22 (to read first block) + 15*10 (processing time) + 7*9 (delay waiting for new tuples) = 235 ms.

Time to read and process 10 1MB blocks using 2 buffers (ms):_____

(d) Suppose the 10 1MB blocks are contiguous on disk and therefore seek time and rotational delay are negligible, and the disk head is already on the correct cylinder. Assuming the double-buffer scenario from part (c), estimate the amount of the required to process 10 1MB blocks.

We can read blocks at a constant rate of 100MB/s.
Each block requires 1/100 seconds, or 10ms. The first block requires 10ms to read, then we can process and read the rest, without delay, due to double-buffering.
10+15*10 = 160ms.

Time to read/process 10 contiguous blocks with 2 buffers:_____

Problem 2 (10 points)

Suppose we have blocks of size 1024 bytes that we use to store fixed-length records. Each block has a 32 byte header used to store information including the number of records in the block.

(a) Suppose we have records consisting of a 12 byte header, and 3 fields of size 5 bytes, 6 bytes and 7 bytes respectively. Within each record, fields can start at any byte. How many records can we fit in a block?

1024-32 = 992 bytes available for records in each block
(12+5+6+7) = 30 bytes per record
992/30 = 33 records per block

Number of records:_____

(b) Suppose that we have 3 records, each with a 12 byte header and 500 bytes of data.

(i) How many blocks will we need to store these 3 records if no spanning is allowed?

As above, 1024-32 = 992 bytes available for records in each block
Each record is 512 bytes.
To avoid spanning, we must store each record on a separate block. 3 records = 3 blocks.

Number of blocks:_____

(ii) How much total free space is there in the blocks if no spanning is allowed? (Assume we are not storing anything else)

(992-512)*3 = 1440 bytes

Available space in bytes:_____

(iii) How many blocks will we need to store these 3 records if spanning is allowed? In addition to the record header, each time a record is split into fragments, each fragment needs its own header of 12 bytes.

Each record is 512 bytes. We can fit the first on one block, but the second record will need to span blocks one and two. After placing the first record, we have 992-512=480 free bytes in the first record. Spanning will cost an addition 12 bytes, so we are left with 480-12=468 bytes. Therefore, we place the remaining (512-468 = 44) bytes along with the 12 byte fragment header on the second block. This consumes a total of 56 bytes on the second block. We have 992-56 = 936 bytes remaining in the second block, so we can place the third record in the second block (leaving 936-512 = 424 bytes).

Two blocks.

Number of blocks:_____

(iv) How much total free space is there in the blocks if spanning is allowed as in (iii)? (Assume we are not storing anything else)

Per above, we have completely filled the first block and have 424 bytes left in the second block.

424 bytes

Available space in bytes:_____

(c) Now consider blocks of size 1024 bytes that we use to store variable-length records. Each block has a fixed 32 byte header used to store information including the number of records in the block. In addition to this fixed header, the header contains variable number of 2 byte pointers to each record in the block. Records can start at any byte offset and are packed as densely as possible. Which of these following combinations of records can be stored in a single block? Circle all that apply.

(i) 32 records of 23 bytes each

(ii) 30 records of 20 bytes each and 10 records of 10 bytes each

(iii) 5 records of 11 bytes, 10 records of 13 bytes and 20 records of 17 bytes

(iv) 2 records of size 496

With N records, we have 1024-32-2*N free bytes.

i: CIRCLE
32 records require 23*32 bytes of space. 1024-32-2*32-23*32=192 bytes free

ii: CIRCLE
40 records total
1024-32-2*40-30*20-10*10 = 212 bytes free

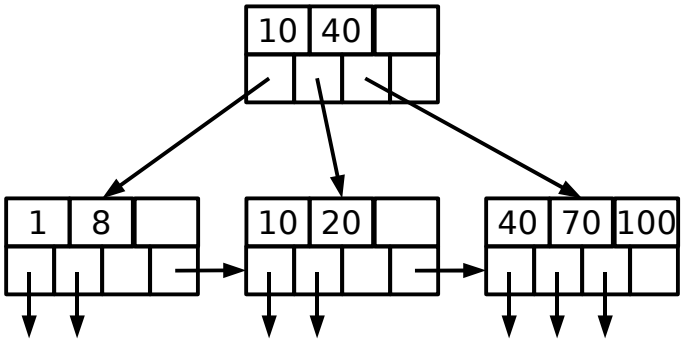
iii: CIRCLE
35 records total
1024-32-2*35-5*11-10*13-20*17 = 397 bytes free

iv: NOT CIRCLED
2 records total
1024-32-2*2-496*2 = -2 (out of space!)

Problem 3 (10 points)

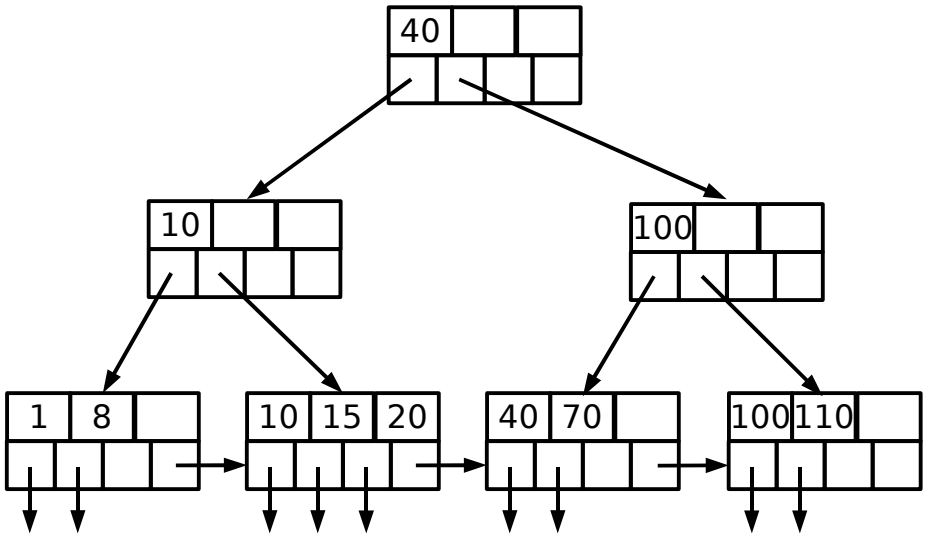
In this problem, if a B+tree node needs to be split, the new node on the left should contain the larger number of non-empty pointers. If a B+tree node needs to coalesce with its sibling, coalesce the node with its left sibling.

(a) Consider the following B+tree of with $n = 3$ pointers per block. (We are using the textbook's way of depicting B+trees here.):



We subsequently perform the following operations in order: insert 15, insert 30, insert 110, delete 30. Draw the tree after these operations.

Final B+tree:



Now consider a B+tree with 10 pointers per block and depth 5:

(b) If our minimum node fill factor is 5 record pointers or non-leaf child pointers, what is the minimum number of **record pointers** the tree can contain?

If filled to the minimum five pointers, the first four levels give us $5*5*5*5 = 625$ leaf nodes; if leaf nodes are half full of record pointers, they contain 5 records each. So, $625*5 = 3125$ record pointers.

(c) What is the maximum number of **record pointers** the tree can contain? Recall that each leaf node has a next-leaf pointer, and, for this question, next-leaf pointers count towards the 10 pointers per block limit.

If full, the first four levels give us $10*10*10*10 = 10000$ leaf nodes; to maximize record pointers, we should fill leaf records with the maximum $10-1 = 9$ record pointers. So, $10000*9 = 90000$ record pointers.

Problem 4 (10 points)

Consider relations $R(A, B)$, $S(B, C, D)$ and $T(C, D)$. The following sub-problems ask you to rewrite relational algebra expressions. You can assume that the relations contain sets (not bags). If the requested rewrite is *not* feasible, state so and briefly explain why. Also, make sure there are no unneeded expressions in your rewrite, e.g., $\pi_{CD}T$ and $\sigma_{A \neq A}R$ are unneeded.

(a) State whether the following expression is feasible. If so, rewrite the following expression (including the projection, if necessary) by pushing the projection as far down as possible:

$\pi_{AD}[\sigma_{C=5}(R \bowtie S)]$

$\pi_{AD}[\sigma_{C=5}(\pi_{AB}R \bowtie \pi_{BD}S)]$

(b) State whether the following expression is feasible. If so, rewrite the following expression so it does not contain a union operator and contains one selection operator (instead of two):

$[R \bowtie (\sigma_{C=2}S)] \cup [(\sigma_{A=1}R) \bowtie S]$

$\sigma_{(C=5 \vee A=1)}[R \bowtie S]$

(c) State whether the following expression is feasible. If so, What is the *minimum* number of operators required to express the query represented by the following expression?

$\sigma_{D=2}[(\sigma_{C=2}S) \bowtie (\sigma_{C=1}T)]$

Zero; if we select all rows with $C = 2$ from S and all rows with $C = 1$ from T, then join on C and D via natural join, we are guaranteed to have no output so we can avoid running the query entirely.

(d) Give two reasons why pushing down selections can improve query execution speed. Give one reason why pushing down projections can improve query execution speed.

Selection: need to read less data from disk; can avoid work later on in query processing;
Projection: need to read less data from disk and also transfer less data as we process it

Problem 5 (10 points)

Consider performing a natural join on two relations $R(X, Y)$ and $S(Y, Z)$. Relation R has 30,000 records stored contiguously in blocks. One block of memory can hold 30 records from relation R. Relation S has 5,000 records **not** stored contiguously in blocks. One block of memory can hold 25 records of relation S. There are 50 main-memory buffers.

Calculate the I/O cost for each of the following algorithms, using an analysis similar to the one done in class. Ignore the I/O cost of writing the final join output to disk. Unless stated otherwise, the records in the relations are not sorted.

(a) Iteration Join Algorithm, R First: The algorithm reads 50 blocks of R into memory, then reads all of S (1 record at a time) into memory, joining with R. This process is repeated until all records are processed.

Each pass through R processes $30 * 50 = 1500$ records of R. There are $30,000$ records in R, requiring 20 passes. Each pass requires $5,000$ I/Os to read S. So $(5000 + 50) * 20 = 101,000$ I/Os total.

Number of IOs: _____

(b) Merge Join: Assume R and S are not sorted. First sort R by Y using two-pass merge-sort, writing all of the records in R into a sorted file. Then sort S by Y the same way. Finally, perform a merge-join on the sorted relations. (Note: in this implementation, we do *not* apply the optimization where we directly join the sorted runs in memory!)

To sort each relation, we need to read it in, sort each chunk, write each chunk out, then read in the chunks and write out the sorted, merged chunks (4 full passes through each relation, and the first read of S is not contiguous). Producing these sorted runs requires $4 * 30000/30 = 4000$ I/Os for R and $5000 + 3 * (5000/25) = 5600$ I/Os for S. Next, we merge the sorted relations, requiring one more pass through each (1000 and 200 respectively). Total is $4000 + 5600 + 1000 + 200 = 10,800$ I/Os.

Number of IOs: _____

(c) Hash Join: We first hash records from R (using the Y attribute) into 50 buckets. (No R buckets are kept in memory.) We then hash S into 50 buckets in a similar fashion. Then we join each pair of Ri and Si buckets. Our hash function is (miraculously!) perfectly balanced, and buckets are equally sized.

We read and write each relation once to hash them, followed by a read through each relation bucket by bucket. All but the first read of S are contiguous. R requires $3 * 30000/30 = 3000$ I/Os and S requires $5000 + 2 * 5000/25 = 5400$ I/Os. Total: 8400 I/Os.

Number of IOs: _____

(d) Extra Credit (1 point): In class, we had discussed that for the hash join, the size of a bucket needs to fit within main memory. Namely, $(x/k) < m$ where x is the number of blocks in the relation, k is the number of buckets, and m is the number of memory buffers (not including the one used to read in from disk). In the scenario where the bucket size is too large, describe a change to the hash-join algorithm to rectify this problem. Assume k is fixed to be m. You may not simply increase k nor increase m.

Basic idea: recursively hash until buckets are sized m or smaller — for example, hash records into k buckets. Then, for each bucket that has more than m records, hash the bucket contents using a second hash function to produce k more buckets. Repeat/recurse until no buckets larger than m exist, then join each respective pair of buckets.

Problem 6 (10 points)

State if the following statements are true or false. Please write TRUE or FALSE in the space provided.

1. The first and second levels of a secondary index are dense.

   FALSE. Building a dense second level on top of a dense first level is redundant.

   ANSWER:_____

2. Using the notation from lecture: storing a hash table constructed via linear hashing requires 2^i blocks.

   FALSE. We only use m blocks at a time, and $m \leq 2^i$.

   ANSWER:_____

3. If we store our data in a columnar form (and without compression), then aggregating a single column of integers from a relation with K integer columns (e.g., `SELECT SUM(a) FROM table;`) stored on disk should execute approximately K times faster than executing the same query over the data stored in row form.

   TRUE. We will only have to read $\frac{1}{K}$ of the data blocks when data is stored in columnar form.

   ANSWER:_____

4. For range queries, merge join is always faster than iteration (nested loops) join.

   FALSE. For small relations (small N), N^2 (IOs required to perform nested loops join) may be smaller than $N \log(N) + N$

   ANSWER:_____

5. Given two relations R and S with indexes on their primary keys, the fastest way to join R and S by primary key is by using iteration (nested loops) join.

FALSE. The fastest join method depends on the size of R and S and selectivity of the join condition. For example, hash join could be faster for relations that have few matches.

ANSWER:_____

6. Using histograms for cardinality estimation always results in more accurate results than simply using the full domain or distinct value assumptions in class.

FALSE. There is no guarantee that a histogram will be more accurate. Say we perform cardinality estimation for a selection $a = 10$. If there are 1000 records in the table, 100 distinct values of a, and a high degree of skew in the histogram so the bucket for $a \in (0, 20)$ has 400 records, then we'd estimate $\frac{1000}{100} = 10$ according to the distinct value assumption and $\frac{400}{20} = 20$ using the histogram assumption. This is to say that, in expectation, histograms will be more accurate, but they are not guaranteed to be.

ANSWER:_____

7. If we don't use overflow blocks, increasing the number of bits used in extensible hashing by one always doubles the directory size.

TRUE. This follows from the basic algorithm for extensible hashing.

ANSWER:_____

8. Each primary key lookup in a B+tree containing N records and P pointers per node will require $log_P(N)$ node traversals.

TRUE. B+tree record pointers are stored in leaf nodes, so we will need to traverse down to the leaf.

ANSWER:_____

9. Each primary key lookup in a B-tree (not B+tree) containing N records and P pointers per node will require $log_P(N)$ node traversals.

FALSE. Unlike a B+tree, a B-tree stores some record pointers in non-leaf nodes, so lookups can be faster.

ANSWER:_____

10. LRU is a good replacement policy for caching B+tree nodes.

FALSE. LRU will cache leaf and non-leaf nodes equally, despite the fact that non-leaf nodes are much more likely to be accessed. As discussed in class, a better caching scheme is to cache the root and upper non-leaf nodes of the tree.

ANSWER:_____