# CS150 Midterm Solutions

## I. **Storage: Disk, Files, Buffers** [17 points]

1. [5 points] Write down the letters of true statements *in alphabetical order.* (If none are true, write ∅.)
   A. When querying for a 16 byte record, exactly 16 bytes of data is read from disk. False, a page's worth of data is always read from disk.
   B. Writing to an SSD drive is more costly than reading from an SSD drive. True, writes to an SSD can involve reorganization to avoid uneven wear and tear.
   C. In a heap file, all pages must be filled to capacity except the last page. False, there is no such requirement. With clustered indices, it is common to keep heap file pages 2/3 full to accommodate inserts.
   D. If the file size is smaller than the number of buffer frames, a sequential scan of the file using either MRU or LRU (starting with an empty buffer pool) will have the same hit rate. True, our eviction policy doesn't matter because the file fits in memory.
   E. Assuming integers take 4 bytes and pointers take 4 bytes, a slot directory that is 256 bytes can address 64 records in a page. False, as shown in lecture, an entry in slot directory is 8 bytes, because a single entry consists of both a pointer and an integer (length).
   F. In a page containing fixed-length records with no nullable fields, the size of the bitmap never changes. True, the size of the records is fixed, so the number we can fit on a page is fixed.

2. [4 points] Write down the true benefits of using a record header for **variable** length records *in alphabetical order* (or if none are true benefits, write ∅.)
   A. Does not need delimiter character to separate fields in the records. True.
   B. Always matches or beats space cost when compared to fixed-length record format. False.
   C. Can access any field without scanning the entire record. True.
   D. Has compact representation of null values. True.

3. [8 points] Assume we have 4 empty buffer frames and the following access pattern, in which pages are immediately unpinned.

T A M E T E A M M A T E M E A T L I D

Use the replacement policy listed, and list the four pages in the buffer pool at the end, *in alphabetical order.* Hint: you don't need to draw a big chart for every access — look for patterns.

A. MRU. ADEM
B. LRU. DILT
C. Clock. (*Assume the clock hand starts on the first buffer and does not move unless a page needs to be replaced.).* DEIL

---

## II. **Sort/Hash** [16 points]

For this question, consider our table of Products from the previous Joins section, but assume:

- **[P] = 2000 pages**
- **$p_C$ = 100 tuples/page**
- **B = 40 pages** of buffer
- In all parts, we'll use **QuickSort** for our internal sort algorithm.
- **Include the cost of the initial scan *and* cost of writing output in your I/O calculations.**

1. [7 points] First, let's sort our table of Products (P) using the external algorithm we learned in lecture.
   A. How many passes are needed to sort this file? 3. ceil(log_39(ceil(2000/40)))+1 = 3. We accepted answers written as an expression, only if the ceilings were correct (i.e. your expression must evaluate to 3 and exactly 3).
   B. What is the I/O cost (in pages) of sorting this file? 12000. 2N*3 = 2*2000*3 = 12000
   C. Suppose we want to decrease the I/O cost of sorting this file, but we want to add the minimum number of buffer pages possible. Among the numbers on the answer sheet (1, 5, 10, 50) circle the *smallest* number of additional buffers to add that decreases the I/O cost. We accepted either 5 or 10. A lower IO cost would have a maximum of 2 passes, so B(B-1) = 2000, where B = 40+x. x=5 additional buffers is an approximate answer, whereas 10 additional buffers would be a fully correct answer.

2. [5 points] Given the resources at the top of Question III, answer the following two questions. Do not bother to simplify arithmetic expressions over constants, like 247*($36^3$+log(4)).
   A. What is the largest file size (in pages) that we can sort in 3 passes? 40(39)(39). Recall that we can sort B in one pass, B(B-1) in two passes, and B(B-1)^(k-1) in k passes.
   B. What is the smallest file size (in pages) that will require 3 passes to sort? 40(39) + 1. This is the largest file sortable in 2 passes, plus 1.

3. [4 points] Suppose I want to eliminate duplicates from our (unsorted) table of Products, using external hashing. Write down the letters of true statements. (If none are true, write ∅.)
   A. Deduplicating the file using hashing can have a higher IO cost than sorting the file (without deduplicating). True. Consider a file with very, very skewed data, and/or a set of very, very poor hash functions, such that many repartitioning passes are needed.
   B. Deduplicating the file using hashing can have a lower IO cost than sorting the file (without deduplicating). True. Consider an extreme case, where the entire file consists of duplicates - there will only be one tuple to output (and only one pass over the input).
   C. If external hashing recursively partitions on one partition, it will do so on all partitions. False. You only need to recursively partition oversized partitions.

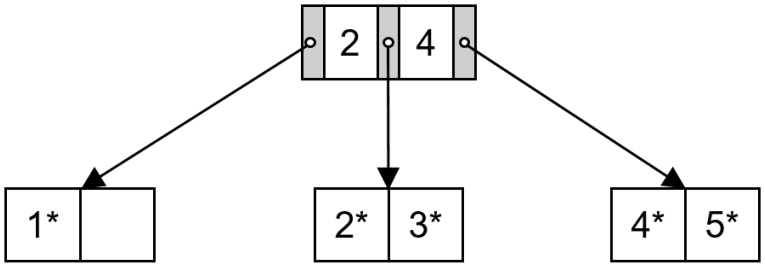
---

## III. **Indexes and B+ Trees** [18 points]

Note: for B+ tree page splits with an odd number of items, assume that the majority of the items is placed on the right-hand page after the split.

1. [6 points] Alphabetically, write down the letters of statements that apply (or write ∅.)
   A. All internal keys in a B+ tree also appear in its leaf nodes. False, a leaf node which has been copied up can be deleted.
   B. The height of a B+ tree increases whenever any node splits. False, height increases when root node splits.
   C. An ISAM index is similar to a B+ tree, but does not allow for insertion of new values. False.
   D. The column(s) we select for our index key must have a unique value for every row in the table. False.
   E. An Alternative 1 index may be either clustered or unclustered. False, it's clustered by design.

2. [7 points] The following B+ Tree (height=1) has order 1 (max fanout of 3) and each leaf node can hold up to 2 entries. Answer each of the follow questions **independently of each other**.




A. What value(s) would be in the root node if we were to insert 0? 2, 4
B. What value(s) would be in the root node if we were to insert 6? 4
C. Starting with the height 1 tree in the picture above, suppose we start inserting keys 6, 7, 8, … and so on. After inserting what key will the height of the tree become 3? 10

3. [5 points] Assume we are trying to construct a B+ Tree of order 2 (max fanout of 5). Each leaf node can hold up to 4 entries. We insert a total of 16 unique keys via bulk loading, with a fill factor of ¾.
   A. How many leaf nodes will there be? 6
   B. How many internal (non-leaf) nodes will there be? 3
   C. How many internal (non-leaf) nodes do we traverse to do an equality search? 2

---

## IV. **SQL** [17 points]

Consider the following schema:

| | |
|---|---|
| CREATE TABLE Location (<br>lname TEXT,<br>areacode INTEGER **PRIMARY KEY**) | CREATE TABLE People(<br>id INTEGER **PRIMARY KEY**,<br>pname TEXT,<br>areacode INTEGER **REFERENCES Location**) |
| CREATE TABLE Calls(<br>cid INTEGER **PRIMARY KEY**,<br>date DATETIME,<br>from INTEGER **REFERENCES People**, to<br>INTEGER **REFERENCES People**,<br>duration INTEGER) | |

You should assume that referential integrity is being enforced, and no NULL values appear.

For parts 1 and 2, fill in the blanks in the SQL queries.

1. [7 points] Names of pairs of people who called each other on 2014-12-25

```
SELECT p1.name, p2.name
   FROM People p1, People p2, Calls C
   WHERE C.from = p1.pid
      AND C.to = p2.pid
      AND C.date = '12-25-2014'
```

2. [7 points] Find the location to which the most phone calls have been made, and return its location name as well as the number of calls made to it.

```
SELECT L.lname as name, COUNT(*) as NumCalls
   FROM Location L, People P, Calls C
   WHERE C.areacode = L.areacode
      AND P.id = C.to
GROUP BY L.areacode
   ORDER BY NumCalls DESC
   LIMIT 1;
```

3. [3 points] On the answer sheet, alphabetically write down letters of the statement(s) that find the name of the location of the person who made the longest call. (If none match, write ∅.)

A.
```
WITH (SELECT P.areacode AS areacode, C.duration ASduration FROM calls C,
        People P
        WHERE C.from =P.id) AS DC,
SELECT  L.lname FROM
DC, Location L
WHERE L.areacode = DC.areacode
ORDER BY DC.duration DESC LIMIT 1;
```

B.
```
SELECT L.lname
FROM Location L, (SELECT P.areacode AS areacode,
                        C. duration AS duration FROM
                    Calls C, People P WHERE C.from
                    =P.id) AS DC
WHERE L.areacode = DC.areacode
ORDER BY DC.duration ASC;
```

C.
```
WITH (SELECT L.lname AS lname, L.areacode AS LAC, P.id AS pid FROM Location L
        FULL OUTER JOIN People P
            ON L.areacode = P.areacode) AS Names, SELECT
Names.lname
FROM Names, Calls C WHERE
C.cid = Names.pid ORDER BY
C.duration DESC LIMIT 1;
```

## V. **Joins** [17 points]

There are two tables: Candies and Stores.

Candies(candy_id int, candy_name text, manufacturer text)
Stores(store_id int, store_name text, candy_id int)

You want to see which stores carry your favorite candies, but you are unsure of which join algorithm you should utilize to perform the query below.

```
SELECT C.candy_id, COUNT(*)
FROM Candies C, Stores S
WHERE C.candy_id = S.candy_id
GROUP BY C.candy_id;
```

[C] = 100. [S] = 60. B = 25.
$p_C = 5$. $p_S = 10$

We have 100 pages of Candies and 60 pages of Stores. The Candies table has 5 records per page, and the Stores table has 10 records per page. Be sure to choose the inner and outer relations such that you minimize the I/O cost. You have 25 buffer pages at your disposal.

S: outer.

1. [2 points] What is the I/O cost of a block nested loops join between Candies and Stores?

[S] + ⌈[S]/(B-2)⌉* [C] = 50 + ⌈50/(20-2)⌉* 80 = 50 + 3*80 = **290**

60 + ⌈60/23⌉ * 100 = 60 + 3×100 = 360

2. [3 points] What is the I/O cost of an index nested loops join between Candies and Stores?
(There is an index on Stores.candy_id, and it takes an average of 3 I/O's to find a matching tuple.)

Note that our index is on Stores!
[C] + $p_C$*[C]*3 = 80 + 5*80*3 = **1280**

100 + 5×100×3 = 100 + 1500 = 1600

3. [4 points] What is the I/O cost of a sort-merge join between Candies and Stores?

We perform Pass 0 of external merge sort on Candies and Stores. This splits Candies into 4 runs (each contains 20 pages), and it splits Stores into 3 runs (2 runs contain 20 pages, 1 run contains 10 pages). Next, we apply the "important refinement" discussed in the lecture slides. The 7 runs can all be streamed into memory one page at a time, and we perform the merge step of external merge sort at the same time as the join step.
3[C] + 3[S] = 3(80) + 3(50) = **390**

3×(100 + 60) = 480

4. [5 points] What is the I/O cost of a grace hash join between Candies and Stores?
(Assume we have hash functions that can partition the data evenly.)

Each table has more than B pages and less than $B^2$ pages.

B < . < $B^2$
25 < < 625

3x (100+60) = 480

So, the cost is 3[C] + 3[S], which we already calculated to be **390**

5. [3 points] Suppose we add the following clause to the query: "ORDER BY C.candy_id". Which join is best? Mark only one answer.
   A. Block Nested Loop Join
   **B. SortMerge** (since we would like our output to be sorted.)
   C. Join
   D. Grace Hash Join
   E. All of the above are equally efficient.

---

## VI. **Relational Algebra** [15 points]

As shown in the following figures, there are three instances corresponding to three relations: Sailors, Reserves and Boats.

| *sid* | *sname* | *rating* | *age* |
|---|---|---|---|
| 22 | Dustin | 7 | 45.0 |
| 29 | Brutus | 1 | 33.0 |
| 31 | Lubber | 8 | 55.0 |
| 32 | Andy | 3 | 25.0 |
| 58 | Rusty | 10 | 35.0 |
| 74 | Horatio | 9 | 35.0 |

Fig. 1. An instance of Sailors

| *sid* | *bid* | *day* |
|---|---|---|
| 22 | 101 | 10/10/98 |
| 22 | 104 | 10/7/98 |
| 31 | 102 | 11/10/98 |
| 31 | 103 | 11/6/98 |
| 64 | 102 | 9/8/98 |
| 74 | 103 | 9/8/98 |

Fig. 2. An instance of Reserves

| *bid* | *bname* | *color* |
|---|---|---|
| 101 | Interlake | blue |
| 102 | Interlake | red |
| 103 | Clipper | green |
| 104 | Marine | red |

Fig. 3. An instance of Boats

Use relational algebra to describe the following queries.
(Recall that $\pi$ is project, $\sigma$ is select, and $\bowtie$ is join.)

1. [2 points] Find the names of sailors who have reserved boat 104.

$$\pi_{sname}((\sigma_{bid=104} Reserves) \bowtie Sailors)$$

2. [4 points] Find the names of sailors who have reserved a green boat.

$$\pi_{sname}((\sigma_{color='green'}Boats) \bowtie Reserves \bowtie Sailors)$$

3. [4 points] Find the colors of boats reserved by Dustin.

$$\pi_{sname}((\sigma_{sname='Dustin'}Sailors) \bowtie Reserves \bowtie Boats)$$

4. [5 points] Find the names of sailors who have reserved at least one boat.

$$\pi_{sname}(Reserves \bowtie Sailors)$$

---