CPSC 221: Sample Final Exam Questions for Practice

Here are some sample final exam questions for practice, with no guarantee (of course) that these questions—or remarkably similar ones—will appear on the final exam.

Note that we also have the textbooks, labs, midterms, and theory assignments with more practice questions and solutions. It may be useful for you to revisit the sample midterm questions and the theory assignments—and their respective solutions.

We plan to give you a variety of questions on the final exam—some shorter ones, some longer ones, some C++ programming, some proofs, etc., with good coverage throughout the course.

Question 1

a) What is the worst-case running time for inserting n items into an initially empty hash table, where collisions are resolved by chaining? Would it make a difference if you inserted new nodes at the start of the chain rather than at the end (or vice-versa)? Assume that n is greater than m, where m is the number of cells in the hash table.

b) What would be the worst-case running time if each sequence (each hash chain) was stored in *sorted* order?

c) How would your answer to (b) change—if at all—if we stored the root of a binary search tree (BST) (instead of the start of a linked list) in the hash table? In other words, we insert new nodes into the BST tree rooted at hash cell location $h(k)$, where k is the key and h is the hash function.

d) Show (by contradiction) that assuming that m is a prime and $0<h_2(k)<m$, then double hashing probing sequence for key k visits all slots of the table during the first m attempts.

Question 2

Suppose that each row of an $n \times n$ array A consists of 1's and 0's such that, in any row of A, all the 1's come before any 0's in that row. Describe a function running in $O(n \log n)$ time for counting the number of 1's in A.

Question 3

We want to sort an array of size n that contains items that are either 0 or 1.

a) Use Big-Θ notation to describe the asymptotic worst-case number of comparisons made by Insertion Sort to sort such an array. (No proof is necessary for this part.)

b) Describe a worst-case input for (a), in terms of the number of entries *n*, and prove that this results in the Big-Θ expression found in (a).

c) Use Big-Θ notation to describe the asymptotic worst-case scenario for the number of comparisons made by Quicksort to sort such an array. Justify your answer (but no formal proof is necessary).

Question 4

Let A be a collection of unordered objects stored in the form of an array, vector, singly-linked list, or doubly-linked list. Describe (in words) an efficient function for converting A into a set. That is, remove all duplicates from A. What is the running time of this method?

Question 5

Consider 6-tuples of the form `(band_name, rating, lyrics, lifestyle, generosity, friendship)` where `band_name` is a string, and the other 5 attributes are integers having values ranging from 1-100 (e.g., 1 = bad or unfavourable, 100 = good or favourable). For these 5 other attributes, rock music fans have collectively decided (via social media) upon the following numeric values for the band: (1) how much the fans like the band (`rating`), (2) how clean the band's lyrics are, (3) how clean the lifestyles of the band members are, (4) how generous the band members are in giving their money to charity (e.g., children's hospital, seniors' services, disaster relief work, homelessness, youth community programs), and (5) the quality of friends that band members hang around with.

Each string is 30 characters long, and each numeric value (each integer) is 4 bytes long. Let us assume that all pointers in the B+ tree are 8-bytes long.

Suppose we want to create a B+ tree structure that holds 10,000 bands, subject to the following criteria:

- Each node or page in the tree is 4096 bytes long. This is true for both internal and leaf pages.
- We need to store three 8-byte pointers in each node to identify the node's parent, left sibling, and right sibling. Some of these pointers might be null, of course.
- *In the leaf pages*, which is where we store the key-value pairs (also called *data entries*) in the index: We want to store `band_name` as the key along with all 5 integers: (i.e., fields (1)-(5) above will collectively be the data "value" for the key. Thus, a data entry might consist of a 6-tuple (six fields) like: ("Big B+ Tree Band", 50, 25, 73, 51, 14). Note that since all of the data will be contained in the

leaf pages, there is no pointer to a data record on disk for the band (because there isn't any other information to store about the band).

- Data entries cannot span pages. In other words, you cannot store part of a data entry on page *k* and the rest on page *k* + 1.

a) Compute the maximum number of data entries that fit on a leaf page, assuming we want an even number of data entries.

b) Compute the number of leaf pages we need, if they are filled to capacity (subject to the constraints in (a)).

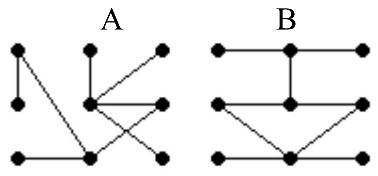
c) Compute the maximum number of (key, child-pointer) entries that fit on an internal page, assuming we want an even number of such entries.

d) Compute the number of internal nodes we need at each level of the B+ tree, if we fill the internal nodes to capacity (subject to the constraints in (c)).

e) Given the constraints in (a)–(d), suppose we knew that there were 3 levels in the tree (root, level-one, and leaf). What is the *minimum* number of unique bands that must be in our tree?

Question 6

Consider the following two graphs, A and B:



Where possible:

a) Re-draw each graph as an *m*-ary (max. *m* children per parent) rooted tree.

b) Assuming the trees in A aren't already balanced, re-draw each tree as a *balanced* tree (if possible), where "balanced" means that all of the leaves appear on no more than 2 adjacent levels: level *k* – 1 and (possibly) level *k*, where k is the furthest level from the root.

Question 7

Suppose in a group of *n* people, everyone has a first name, middle name, and last name. Suppose there are *f* unique first names, *m* unique middle names, and *l* unique last names.

What is the *minimum* number of people needed in order to guarantee that two people have the *same* first, last, and middle names? In other words, that is the smallest number of people for which there *must* be two individuals with the same first, middle, and last names.

Question 8:

In matrix multiplication, the product of two $n \times n$ matrices A and B is one $n \times n$ matrix C whose i,j-th entry $c_{i,j}$ is computed as follows:

$$\sum_{k=1}^{n} a_{i,k} * b_{k,j}$$

If we wrote an algorithm that multiplied matrices in this way, how many scalar multiplications (i.e., * operations) would the algorithm perform when multiplying one $n \times n$ matrix by another $n \times n$ matrix? Use Big-Θ notation. You do not need to prove your result; it suffices to give us a Big-Θ expression, and a brief justification of your result.