Computer Science 409, Spring 2001: Prelim Solutions

Bo graded questions 2 and 3; Don graded questions 1 and 4; I graded question 5.

1. [6 points]

   (a) [3 points] Prove that $(n+1)^2 = \Theta(n^2)$ by giving the appropriate constants n_0, c, and d from the definition.

   Solution: It is immediate that $n^2 \leq (n+1)^2$ for all $n \geq 0$. It is also easy to show (a formal proof is by induction) that $(n+1)^2 \leq 4n^2$ for all $n \geq 1$. (It is also correct to say, for example, that $(n+1)^2 \leq 2n^2$ for $n \geq 3$.) Thus, we can take $n_0 = 1$, $c = 1$, and $d = 4$.

   (b) [3 points] Use the master theorem to solve the recurrence $T(n) = 4T(n/4) + 3n$.

   Solution: In the statement of the Master Theorem, $a = 4$, $b = 4$, and $f(n) = 3n$. Since $\log_4(4) = 1$ and $f(n) = \Theta(n^1)$, by the Master Theorem, it follows that $T(n) = \Theta(n \lg n)$.

2. [9 points]

   (a) [2 points] What is the BST property?

   Solution: The BST property says that if you place keys on a binary tree, the key of a node is greater than or equal to the keys on all the nodes on its left subtree, and less than or equal to the keys on all the nodes on its right subtree.

   (b) [4 points] Show the BSTs you would get at each step if you started with an empty BST and inserted the elements 22, 4, 44, 25, 56, and 48, in that order.
   Solution: Here are the trees you get, in order:

```
22     22     22        22          22           22
      /      /  \      /  \        /  \         /  \
     4      4    44   4    44     4    44      4    44
                          /            / \          / \
                        25           25           25   56
                                       \            \
                                        40           40
```

   (c) [3 points] Now what happens if you delete 22 from the BST?
   Solution: The successor of 22 is 25. Thus, we replace 22 by 25 and delete 25, to get

```
    25
   / \
  4   44
     /  \
    40   56
```

Grading: In part (b), 2 points were taken off for not giving the tree after each step. Othewrsie, 1 point was taken off for each mistake (although if you were consistent with an earlier mistake, no points were deducted for th later mistake). For part 9(c), one point was given if your answer was incorret but still a BST.

3. [12 points] Recopy and fill in the table below, showing the running time for each operation (using big O notation), for the appropriate data structure, if it contains n elements. Make clear whether it's the *expected* running time or an actual bound that does not depend on probabilities (for example, say "expected $O(n^2)$" or just "$O(n^2)$", if there are no probabilities involved). You may assume that the inputs are "nice" (that is, if your expected running time depends on certain standard assumptions about the input, such as all inputs are equally likely, then assume that these assumptions hold For SEARCH, you are given an element and either return the element if it is in the data structure or return NIL; for INSERT, you must insert the element into the data structure; for MAXIMUM, you return the largest element. Assume that the elements in the data structure are integers, so that MAXIMUM makes sense.

   Solution:

Data Structure	SEARCH	INSERT	MAXIMUM
Hash Table	expected $O(1)$	expected $O(1)$	$O(n)$
Skip List	expected $O(\log n)$	expected $O(\log n)$	$O(1)$
Binary Search Tree	expected $O(\log(n))$	expected $O(\log n)$	expected $O(\log n)$
Queue	$O(n)$	$O(1)$	$O(n)$

   Grading: We accepted $O(h)$ as the bound for for BST (but not for skip lists). However, you did have to specify the relationship between n and h. 1 point was deducted for each 2-3 times you omitted the word "expected" if it was required. If you wrote $O(1 + \alpha)$ instead of $O(1)$ for hashing, you lost 1 point.

4. [14 points]

   (a) What is one advantage of using a heap rather than a sorted array to implement a priority queue?

   Solution: Insert takes time $O(\log n)$ for a heap; it may take time $O(n)$ for a sorted array.

   (b) What is two advantages of using a heap rather than a binary search tree to implement a priority queue?

   Solution: Maximum takes time $O(1)$ for a heap; it takes expected time $O(\log n)$ for a binary search tree. Also, for a heap, Insert and ExtractMax are guaranteed to take time $O(\log n)$; for a BST, $O(\log n)$ is just the expected time (and even this assumes that the input is "nice").

   (c) What is one advantage of using hashing with chaining over hashing with open addressing?

   Solution: Hashing with chaining can deal with more elements than the size of the table. It also is much better at dealing with deletion.

   (d) What is one advantage of using hashing with open addressing over hashing with chaining?

   Solution: With hasing with open addressing, there are no pointers and linked lists to deal with. This saves space.

   (e) What is one advantage of the adjancy-list representation of a graph over the adjacency-matrix representation?

   Solution: It will take less space if the graph is sparse.

   (f) What is one advantage of the adjacency-matrix representation of a graph over the adjacency-list representation?

   Solution: The adjacency-matrix representation takes less space if the graph is dense. It can also easily represent weighted graphs.

   (g) What is one advantage of a skip list over a binary search tree?

   Solution: A BST can have height $O(n)$ (and thus perform very badly) if the input sequence has long increasing or decreasing subsequences. A skip list has expected running time $O(\log n)$ for insertion, search, and deletion for all input sequences. The expectation is taken over the coin tosses. Also, we can find the maximum and minimum for a skip list in time $O(1)$.

   Grading: We were fairly flexible here.

5. [9 points] Consider an ADT with the following operations: Insert, GetNext, and GetMax. GetNext returns the next item in FIFO (First-in First-out) order (that is, the item that has been on the list longest) and deletes it from the set. Thus, if only Insert and GetNext are used then the ADT acts like a queue. GetMax returns the item with the maximum key value and deletes it from the set. Thus, if only Insert and GetMax are used, then the ADT acts like a priority queue. Describe a data structure that implements this ADT efficiently. Assume that each of the three operations is done roughly

equally often. How long does each operation take? (You may use any data structure we discussed in class.)

Solution: The idea is to simultaneously use a priority queue, implemented as a heap, and a queue, implemented as a doubly-linked list. Thus, each element has a parent, a left child, and a right child (as in a priority queue), together with next and previous pointers (as in a linked list). To INSERT an element, we do the usual HEAP-INSERT (which takes take $O(\log n)$) and also put it at the head of the list (by setting its next pointer to the previous head of the list and its previous pointer to NIL) (which takes an additional constant number of steps). Thus, INSERT takes time $O(\log n)$. To do GETNEXT, we find the head of the queue (which takes a constant number of steps), delete it from the queue (by making the element next to it be the head of the queue), and then delete it from the heap (just as in Exercise 7.5-5). Thus, GETNEXT also takes time $O(\log n)$. Finally, for GETMAX, we find the maximum element on the heap and delete it (by switching it with the last element and using HEAPIFY); we also get delete it from the queue by fixing the pointers of its next and previous element (in constant time). Thus, GETMAX also takes time $O(\log n)$.

Grading: If you just use a single linked list, the running times for INSERT, GETNEXT, and MAX are $O(1)$, $O(1)$, and $O(n)$, respectively. Similarly for other variants. You got 5/9 if you did this (and analyzed it correctly). You got 8/9 if you used a BST instead of a priority queue (the expected running time is still $O(\log(n))$, but that's only an expectation, and depends on the tree being "nice".