Midterm 2

ECE 608
April 7, 2004, 7-9pm

Name: ______________________________

Read all of the following information before starting the exam:

- **NOTE:** Unanswered questions are worth 30% credit, rounded down. Writing any answer loses this "free" credit, which must be earned back by the quality of the answer. If you wish a question to be treated as unanswered, but you have written on it, clearly write "DO NOT GRADE" in the answer area. In a multi-part question, unanswered *parts* are worth 30%.
- Show all work, clearly and in order, if you want to get full credit. I reserve the right to take off points if I cannot see how you arrived at your answer (even if your final answer is correct).
- No calculators, or materials other than pen/pencil and blank paper are allowed except those we distribute during the exam.
- Please keep your written answers brief; be clear and to the point. Points will be deducted for rambling and for incorrect or irrelevant statements.
- This test has XX problems, each of which is approximately equal in value. Multi-part problems divide the problem score approximately equally among the parts.
- Good luck!

1. Define a "word" to be a finite sequence of letters ('a' to 'z'). Describe how to sort a collection of words in $O(n)$ worst-case time into alphabetical order, where n is the total length of all the words in the collection. Prove that your sort is $O(n)$ worst case.

Solution:

Use a counting sort to sort the words based on their first letter. Then, for each initial letter, recursively sort the words with that first letter with the sort we are designing here, but with the first letter of each word removed—if one of those entries was just one letter long then do not include it in the recursion but place it at the beginning of the results (when the recursion returns, place the first letter back on the front of each word). The base case of the recursion is when the set of words to sort is empty.
Analysis: Each counting sort call is $O(k+26) = O(k)$ when k words are sorted. Let $T(n)$ be the worst-case cost for sorting total length n. For each letter a, let n_a be the total length of the words starting with letter a, and c_a be the number of words starting with a. The sort requires recursive calls for each letter a, each of cost $T(n_a - c_a)$. Also, there is a divide and recombine cost for each subproblem of size $O(c_a)$. Also, there is a counting sort call at cost $O(\sum_a c_a)$. These last two kinds of cost can be combined as one charge or $O(\sum_a c_a)$.
The recurrence for the runtime of the sort is thus $T(n) = \sum_a T(n_a - c_a) + k(\sum_a c_a)$, for some k.
Here, we know that $\sum_a n_a + c_a = n$, because every letter is either the first letter of a distinct word or one of the remaining letters passed on, and $\sum_a c_a \geq 1$, because there is at least one word. We show by substitution that $T(n)$ is $\leq dn$ for some constant d, for all $n \geq 1$.
In the base case, $n = 1$, and the cost is $k(1)$, so we can take $d \geq k$.
In the recursive case, we have $T(n) \leq \sum_a d(n_a - c_a) + k(\sum_a c_a) \leq \sum_a d(n_a) \leq dn$, as desired.

2. Given a set X of n distinct real numbers $x_1, \ldots, x_n$, with corresponding weights $w_1, \ldots, w_n$ summing to one, the weighted median is the least number $x \in X$ such that the total weight in the set $\{y \in X \mid y \leq x\}$ is at least one half. Describe in detail how to compute the weighted median of a set of n distinct real numbers in $O(n)$ worst-case time. You do not need to prove the run-time bound.

Solution:

Design a routine WSELECT that takes an array of elements and a target weight w and returns the least element such that all the weights of elements $\leq$ that element total w or more.

WSELECT operates as follows (following SELECT from the book):

1. Divide the input element into groups of 5 (with $\leq$ one smaller group)
2. Use insertion sort to sort each group and find its median
3. Use SELECT from the book to find the median of the medians x
4. Partition the input array around x
5. Compute the total weight t of the elements $\leq x$
6. Return x if $t - weight(x) < w \leq t$.
7. Else, recursively invoke WSELECT on:
    (a) The elements $\leq x$ if $w < t$. Use the same target weight w.
    (b) The element $> x$ if $w > t$. Use target weight $w - t$.

3.

a. Why would a random-number generator make a poor hash function?

solution: unable to reproduce results, so can't find keys in table.

b. Define and explain the motivation for universal families of hash functions. Explain how such families respond to this motivation.

Solution: A universal family of hash functions is one where the expectation of collision for any pair of keys by a uniformly randomly chosen hash function from the family is no more than the reciprocal of the table size m. The motivation here is to protect against an adversary causing a lot of collisions on purpose, assuming the adversay has access to our source code but not our runtime memory contents. Universal hash families allow the hash function to be chosen at runtime so that the adversary, knowing only the family (from the source code) but not the hash function, will not be able to achieve better than $1/m$ chance of a collision for any key pair.

c. Give as many reasons as you can why the following hash function on strings of characters would perform poorly: the hash function which treats each character of the string as the integer encoding it and returns the sum of those integers.

Solution:

1. Permutations of the same characters have same hash values.
2. For a finite size table, hash may fall outside table. Or the table size required is unbounded.
3. Strings of comparable length have nearby hash-values.
4. Hashes will not vary much in the most significant bits.

4.

a. State the uniform hashing assumption and argue that double hashing does not satisfy it.

Solution:

Uniform hashing assumption is that each key is equally likely to have any of the $m!$ permutations as its probe sequence.
Double hashing does not satisfy this assumption because only m^2 different rehash sequences are possible, one for each pair of hash values from the two hash functions.

b. Describe a scenario when perfect hashing might be useful.

Solution:

Any application that has a static set of known keys can benefit from perfect hashing. Examples are: set of reserved words in a programming language, or the set of filenames on a CD-ROM.

c. Explain why secondary hash tables are needed in implementing perfect hash tables. . . what happens if we just use a single hash table?

Solution:

The secondary hash tables are needed in order to obtain a collision free hashing mechanism while keeping the total storage space requirement within $O(n)$. If we use a single hash table, we can still obtain the same collision guarantees except that it would require $O(n^2)$ storage space.

5.

a. Characterize the set of keys that occur on the path to a particular leaf node in a binary search tree in terms of the behavior of “bounded online max” or “bounded online min” algorithms running on the key sequence used to construct the tree.

Solution:

The set of keys that occur on the path to a leaf is the union of the set of keys that cause an update in the value of “bounded online max” and the set of keys that cause an update in the value of ”bounded online min” before that leaf was inserted in the tree.

b. Describe an n-node binary tree that has average node depth $\Theta(\lg n)$ but maximum node depth $\omega(\lg n)$ and argue that these properties hold of the tree you describe.

See solution to homework problem CLR12.4-2 (homework 8 problem 7).

6.

a. The total path length in a tree is the sum of the depths of the nodes in that tree. Write a recurrence defining the average total path length for a randomly built binary search tree. You do *not* need to solve the recurrence.

See solution to homework problem CLR12-3 (homework 8 problem 10)

b. What algorithm we have studied shows the same recurrence? Describe the analogy between that algorithm and the construction of a randomly built binary search tree.

See solution to homework problem CLR12-3 (homework 8 problem 10)