

## Problem Set 3—Due Wednesday, February. 4 3:15

### Important Instructions:

Write clearly (if you can't, type it). Illegible answers won't get points.

**(20) Problem 1.** Problem 11-2.3 on page 229. First briefly describe how you would do insertions and deletions to maintain a sorted list, and how you would now do searches. Then analyze the performance.

**(35) Problem 2.** In our analysis of successful lookups in hashing we assumed that all keys were equally likely to be looked up. We now consider hashing with chaining under the (common) setting where some items are more likely to be looked up than others. For this we assume that we have keys  $k_1, \dots, k_n$  which are inserted into a hash table of size  $m$ , and as before we assume *simple uniform hashing* for our analysis. We now assume that the probability we lookup key  $k_i$  is  $p_i$ . We also assume here that all lookups are successful (so  $p_1 + \dots + p_n = 1$ ).

**a)** Argue that we will minimize the expected lookup time by sorting the  $p_i$  values so  $p_1 \geq p_2 \dots \geq p_n$  and then inserting the items in the reverse of this order (i.e. first insert  $k_n$ , the key with the smallest  $p_i$ , and insert  $k_1$  last). You are only being asked for a qualitative argument here.

**b)** Suppose instead of inserting them in the correct order (as suggested in a)) we simply keep each chain in order (so the elements in each chain are in decreasing order of  $p_i$  value: as is shown in problem 1 we can do this if the  $p_i$  values are known. Would this be as good as the approach of part a)?

**c)** We now consider specific values of the  $p_i$ 's: Let  $p_i = \frac{1}{iC}$  where  $C = 1 + 1/2 + \dots + 1/n$  ( $C$  is chosen so the  $p_i$ 's add to 1; note that  $C$  is about  $\ln(n)$ ).

Suppose that we insert items as suggested in part a). Analyze the expected number of items in a chain that are examined for successful lookup for these probabilities. Note that we are looking for a precise analysis not a big  $O$  or  $\Theta$  answer (e.g. for an item at the front of a chain, its lookup cost is 1, not just  $O(1)$ ).

**d)** The prior discussion assumed that the lookup probabilities are known (e.g. we know the  $p_i$  values) and don't change. In real life, neither of those are always (are even often) true. Suppose we initially insert the items in an arbitrary order (since we don't know the  $p_i$  value). Consider the following strategy whenever we lookup a key, we move that key to the front of the list.

Would this be a sensible strategy? Discuss when it might or might not be good (as in a), only looking for a qualitative answer here; too hard to do a precise analysis).

**e)** Up to now we have been talking about hashing with chaining. Now consider open addressing (where all items are stored in the table) and collisions are resolved using double hashing.

i) As in a) suppose we know the  $p_i$ 's in advance. Would it still be best to insert the items in decreasing  $p_i$  order? (i.e. first insert  $k_n$ , the key with the smallest  $p_i$ , and insert  $k_1$  last). Justify your answer.

ii) Now suppose we want to mimic the strategy of part d) where when we look up an item we "move it to the front" which now means that after looking up key  $k_i$ , we want to store it in location  $j = h_1(k_i)$  (of course  $k_i$  might already be there, in which case we do nothing). How would you implement this "move it to the front" operation, and describe any complications with implementing the "move it to the front" strategy for this setting?

**(30) Problem 3.** We consider some variants of the subset sum problem discussed in class: given a set of positive integers  $L = \{x_1, x_2, \dots, x_n\}$  and a target  $b$ , the *Subset Sum* problem asks you to find  $S \subseteq L$  such that the sum of the elements in  $S$  is no greater than  $b$  but is also as close to  $b$  as possible.

**a)** We assumed the  $x_i$  were integers. Why was this important?

Would it cause a problem if  $b$  was not an integer (assuming the  $x_i$  are still integers)?

**b)** Describe how to modify the dynamic programming formulation so that it only uses a single array of size  $b + 1$  to find the target value closest to  $b$  (so you need not find the actual set of items). Be careful that you don't add the value of the item  $x_i$  to a total which already used  $x_i$ .

**c)** For the setting of part b) where we only use  $O(b)$  space, describe how to find the actual items in the best solution while still using only  $O(b)$  space (hint keep track of when a table entry becomes 1).

**(15) Problem 4.** Problem 15.1-2, page 331.