Name:	
(as it would appear on official course roster)	
Umail address:	**@umail.ucsb.edu**
Optional: name you wish to be called if different from name above.	
Optional: name of "homework buddy" (leaving this blank signifies "I worked alone"	

section **9am, 10am, 11am, 12pm**

1

h08

CS32 F19

h08: Hashing

ready?	assigned	due	points
true	Tue 10/22 02:00PM	Tue 10/29 02:00PM	50

You may collaborate on this homework with AT MOST one person, an optional "homework buddy".

MAY ONLY BE TURNED IN IN THE LECTURE LISTED ABOVE AS THE DUE DATE,
OR IF APPLICABLE, SUBMITTED ON GRADESCOPE. There is NO MAKEUP for missed assignments;
in place of that, we drop the lowest scores (if you have zeros, those are the lowest scores.)

Reading: Hashing, DS 12.2

1. (10 pts) Fill in the information in the header. The following are required to get the 10 "participation" points.
   - Filling in your name and umail address.

   Also: For paper submission PLEASE submit on ONE SHEET OF PAPER, double-sided if at all possible. If you must submit on two printed sheets write name on BOTH sheets and no staples, paperclips, or folded corners.

2. From DS4 (12.2): There is a technique known as "double hashing".
   a. (5 pts) What is the problem that double hashing is designed to solve? **Briefly explain.**

   b. (5 pts) How does double hashing solve that problem? **Briefly explain**

2

h08

CS32 F19

3. From DS 12.2: The abstract data type (ADT) known as a "dictionary" provides a way to look up keys and find values. We define these abstract operations for the ADT:
   - lookup(key) which returns either value, or an indication that the value is not present
   - remove(key) which removes the item with that key (if it exists)
   - insert(key,value) which inserts the new key, and the value (or replaces the value if it is already present)

   We could implement a dictionary by having an array of (key,value) pairs, sorted by key. We could use binary search for lookup. We'd have to figure out a way to put new values into the sorted list, and remove values. Or we could use hashing, as described in DS 12.2:

   a. (4 pts) If we use binary search on a sorted array, what is the worst case time for lookup(key) in terms of big-O? (No explanation needed; just state the answer)

   b. (8 pts) If we use binary search on a sorted array, what is the worst case big-O time for remove(key)? This time, **briefly** explain your answer.

   c. (5 pts) If we use hash tables instead, what is the worst case big-O time for lookup(key)? Just state it.

   d. (4 pts) Is the worst case for lookup(key) for hash tables better or worse than the binary search approach?

   e. (5 pts) If we use hash tables instead, what is the average (expected) case big-O time for lookup(key) assuming no collisions? Just state it.

   f. (4 pts) Is the average (expected) case for lookup(key) for hash tables better or worse than the binary search approach?