CSE 373 -- Autumn 1999 -- Assignment #4

(*due Monday, November 8*)

1. **Exercise 4.4** (Use an inductive proof)

2. **Exercise 5.1** (assume that we rehash when lambda is at least 0.8 for parts a-d)
   (e) Repeat part c, rehashing when lambda is at least 0.5

3. **Assume that we want to use a hash table to store chess game configurations so that we can store good moves from certain configurations. For each hash table entry, the "key" will be the pieces' locations on the board; the "data" associated with each key will be the recommended move from that configuration.**
   (Note that you don't need to know anything deep about chess for this problem: simply that the board is 8 x 8 squares, and that each square can hold a single piece. There are two players: white and black. Each player starts the game with 16 pieces: 8 pawns, 2 rooks, 2 knights, 2 bishops, 1 queen, and 1 king. Any two pieces of the same type are indistinguishable (e.g., all pawns are equal). As the game progresses, players lose their pieces which are removed from the board.)

   (a) Suppose that we number the rows of the board 1-8 and the columns of the board 1-8. One possible hash function is to compute the sum of the row and column of every piece on the board and mod the total by the tablesize. What is the disadvantage of this hash function?

   (b) What if we were to modify the above hash function so that we also add in a value based on the type of the piece (pawn = 1, rook = 2, ..., king = 8) and multiply the white pieces by 384 (16 pieces * (8 rows + 8 cols + 8 types of pieces)) to distinguish them from the black pieces? (As always, we mod this result by the tablesize)

   (c) Describe the best hash function you can think of for a chess board configuration.

4. **Henry Hacker decides to implement his own hash table that will support the normal hash table operations, but will also support a `FindMax()` operation in O(1) time. He does this by adding an extra piece of data to the HashTable class:**

```
class HashTable {
  private:
    ... normal stuff here ...
    HashedObj maxval;

  public:
    ... normal stuff here ...
    HashedObj& FindMax();
};
```

   His approach is to store the first piece of data inserted into the hash table in `maxval`. From that point on, every piece of data that is inserted is compared against `maxval` and, if it is bigger, replaces `maxval`. The implementation of `FindMax()` is simply to return `maxval`.

   This implementation has a fundamental flaw that will prevent `FindMax()` from always running in O(1) time. What is it? (you should assume that `HashedObj` supports the "<" and ">" operators)