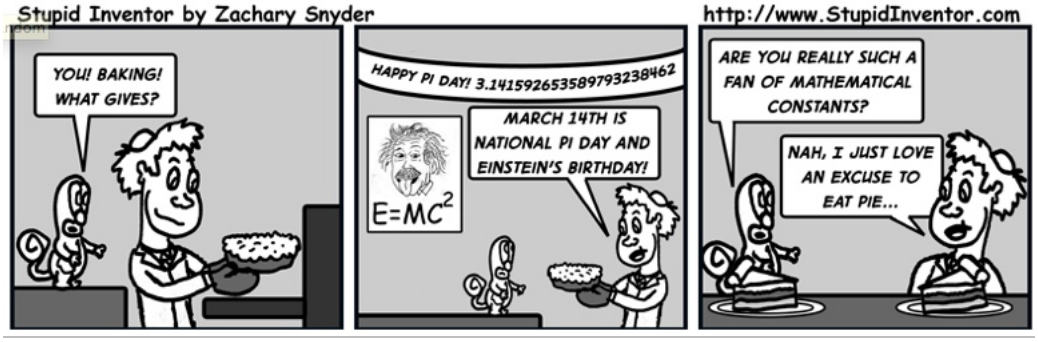
CS302 - Assignment 10

Due: Tuesday, March 20 at the beginning of class
Hand-in method: paper



http://www.stupidinventor.com/2010/03/14/comic-55-happy-pi-day/

1. **[8 points]** We saw HeapSort which takes the data points, inserts them into a heap and then repeatedly calls ExractMax to sort the data (recall this was $O(n \log n)$). A similar sorting algorithm exists for binary search trees called TreeSort: call Insert on each data item to build a binary search tree, then do an inorder traversal of the data to get the sorted list.

   (a) **[3 points]** Does it work, i.e. is it guaranteed to always give you data in sorted order? Give a counterexample or a *brief* justification of why it's correct.

   (b) **[3 points]** What are the best, average and worst case running times for this algorithm assuming you randomize your data before inserting.

   (c) **[2 points]** How do these run-times change if we are using a balanced binary search tree (e.g. a red-black tree)?

2. **[5 points]** Suppose we decide to speed up our B-Tree-Search algorithm and use binary search within a node rather than a linear search. Show that this make the CPU time required $O(logn)$ independently of how t is chosen.

3. **[8 points]** We'd like to add some additional functionality to our B-tree. For each of the functions below, describe succinctly (or write pseudo-code) how to accomplish this and state the worse-case running time.

   (a) **[3 points]** Find the minimum key in the B-Tree

   (b) **[5 points]** Find the successor of a given key. You can assume that you already have a reference to the key (i.e. you don't have to search for it).

4. **[7 points]** Write pseudo-code for a recursive function NumTrees that, given a number n, computes the number of unique binary search trees that could store the numbers 1 through n (note, this function would actually give you the number of trees possible for storing any n unique numbers). For example, NumTrees(4) should return 14, since there are 14 unique binary search trees that store 1, 2, 3, and 4 (try and list them all out).

Just for fun

Try implementing your function from Problem 4 and see what values you get for larger n.