15-122 : Principles of Imperative Computation, Spring 2016

Written Homework 9

Due: Monday 21st March, 2016

Name: _____

Andrew ID: _____

Section: _____

This written homework covers binary search trees and AVL trees.

The assignment is due by 1:30pm on Monday 21st March, 2016.

This assignment is to be completed by editing the PDF file,
printing it to another PDF file,
and then submitting through Gradescope.

1. **Binary Search Trees**

1pt (a) Describe the final binary search tree that results from inserting the following keys in the order given.

```
93, 86, 71, 115, 88, 99, 94, 77, 95, 109
```

You can describe it either by recursively putting parentheses around subtrees and writing _ for empty subtrees, so that for example

```
      5
     / \
    2   7
   / \   \
  1   4   9
     /
    3
```

is described as

```
((_ 1 _) 2 ((_ 3 _) 4 _)) 5 (_ 7 (_ 9 _)).
```

If you have access to a PDF editing appliction that allows you to draw, feel free to use that instead. In that case, be sure all branches in your tree are drawn *clearly* so we can distinguish left branches from right branches.

2pts

(b) How many different binary search trees can be constructed using the following five keys if they can be inserted in any order?

```
35, 17, 42, -7, 90
```

Show how your answer is derived. We've begun the derivation below; we've used $t(n)$ to stand for the number of binary search trees with n elements.

Think recursively: How many trees with 0 elements can possibly exist? How many different trees with 1 element can possibly exist? 2 elements? 3 elements? 4 elements? Think about how to build up your answer from answers to simpler questions. (It might help to come back to this question after doing the last question on AVL tree height.)

n	$t(n)$
0	$t(0) = 1$
1	$t(1) = 1$
2	$t(2) = t(0) \times t(1) + t(1) \times t(0) = 2$
3	
4	
5	

For the following questions, you should refer to the implementation of binary search trees discussed in class. The code is available on the course website. We'll use the version of binary search trees where the client defines a single `elem_compare(x,y)` function that returns `-1` if `x` is "less than" `y`, `0` if `x` is "equal to" `y`, and `1` if `x` is "greater than" `y`. These relations are defined based on the type of the keys in the elements.

2pts

(c) Assume that the client also provides a function `elem_print(e)` that prints the given element `e` in a readable format on one line. Complete the function `bst_inorder` which prints the elements of the given BST on one line in order from smallest key to largest key. If the BST is empty, nothing is printed. You will need a **recursive** helper function `tree_inorder` to complete the task.

Think recursively: To print the elements rooted at some tree node T in order, first print all of the elements of T's left subtree in order, then print the element of the node T, and finally print all of the elements of T's right subtree in order. You should not need to examine the keys since the contract guarantees the argument is a BST.

```
void tree_inorder(tree* T)
//@requires is_ordered(T, NULL, NULL);
{









}

void bst_inorder(bst_t B)
//@requires is_bst(B);
{
  tree_inorder(_____________________________________________);
  print("\n");
}
```

3pts

(d) The `tree_insert` function in the lecture notes is recursive, but it is also possible to implement it iteratively. Fill in the missing code.

```
tree* tree_insert(tree* T, elem x)
//@requires is_tree(T) && x != NULL;
//@ensures is_tree(\result);
{
  tree* parent = NULL;
  tree* current = ____________________________________;
  while (current != NULL)
    /*@loop_invariant current == NULL || parent == NULL

      || current == ____________________________________

      || current == ____________________________________;@*/
    {
      parent = current;
      int cmp = elem_compare(x, current->data);
      if (cmp == 0) {
        current->data = x;
        return T;
      } else if (cmp < 0) {

        current = ____________________________________;
      } else {//@assert cmp > 0;

        current = ____________________________________;
      }
    }
  tree* R = alloc(tree);
  R->data = x;
  if (parent != NULL) {
    int cmp = elem_compare(x, parent->data);
    if (cmp < 0)
      ____________________________________________________;
    else
      ____________________________________________________;
  }
  else {
    ____________________________________________________;
  }
  return T;
}
```

2. **AVL Trees.**

3pts (a) Describe the AVL trees that result after successively inserting the following keys into an initially empty tree, in the order shown:

```
39, 77, 91, 88, 106, 80, 97
```

Show the tree after each insertion and subsequent re-balancing (if any) is completed: the tree after the first element, `38`, is inserted into an empty tree, then the tree after `76` is inserted into the first tree, and so on for a total of seven trees. Make it clear what order the trees are in.

Be sure to maintain and restore the BST invariants and the additional balance invariant required for an AVL tree after each insert.

(b) Recall our definition for the height h of a tree:

> *The height of a tree is the maximum number of nodes on a path from the root to a leaf. So the empty tree has height 0, the tree with one node has height 1, and a balanced tree with three nodes has height 2.*

The minimum and maximum number of nodes m in a valid AVL tree is related to its height. The goal of this question is to quantify this relationship.

2pts

i. Let $m(h)$ be the minimum number of nodes in an AVL tree of height h. Fill in the table below relating h and $m(h)$:

h	$m(h)$
0	0
1	1
2	2
3	____
4	____
5	____
6	____

1pt

ii. Guided by the table in part (i), give an expression for $m(h)$.
Here's a hint: recall that the nth Fibonacci number $F(n)$ is defined by:

$$F(0) = 0$$
$$F(1) = 1$$
$$F(n) = F(n-1) + F(n-2), \qquad n > 1$$

You may find it useful to use the Fibonacci function $F(n)$ in your answer. Your answer does not need to be a closed form expression; it could be a recursive definition like the one for $F(n)$.

1pt

iii. Give a closed form expression (non-recursive) for $M(h)$, the *maximum* number of nodes in a valid AVL tree of height h.

Solution: $M(h) =$ _____