CS311 Homework 4

160 points, 20 points per problem

Problem 1

Consider the following (partial) implementation of a binary search tree:

```
BST
{
    integer data
    BST left
    BST right

    BST(integer d)
        data = d
        left = null
        right = null

    func add(int toAdd)
        if toAdd < data
            if left is empty
                left = new BST(toAdd)
            else
                left.add(toAdd)
        else
            if right is empty
                right = new BST(toAdd)
            else
                right.add(toAdd)
}
```

and the following algorithm:

BUILD-BST(list)

1: $r =$ new $BST(list[0])$
2: **for** $i = 1$ to $list.length - 1$ **do**
3: $\quad r.add(list[i])$
4: **end for**
5: **return** r

Assume that *list* is a nonempty list of n elements.

(a) Prove that the best case running time of BUILD-BST is $\Omega(nlog(n))$.

(b) Prove that the worst case running time of BUILD-BST is $O(n^2)$.

Problem 2

Recall that the *balance factor* of a node in an AVL tree is the height of the node's left subtree minus the height of the node's right subtree. For the purposes of this problem, assume that a tree with no nodes has a height of 0.

Consider an AVL tree where every node in the tree has a balance factor of 0. Prove using induction that any such AVL tree is also full, i.e., the d^{th} level of the tree (where the root is in level 0) has either 2^d nodes or 0 nodes.

Problem 3

Problem: BST initialization
Input: A list of elements, sorted in ascending order
Output: A complete BST (i.e. a BST where every level except the last is full, and the nodes in the last level are as far left as possible) that contains precisely the elements in the input list.

Provide pseudocode that solves the BST initialization problem in linear time.

Problem 4

As described in lecture, *open addressing* is a method for resolving collisions in hash tables. This method resolves hash collisions by searching through alternate locations in a specific order (defined by some *probing sequence*) until the target record is found or an empty bin is found. For this problem we adopt the convention that a *probing function* $P(x, i)$ induces the probing sequence $P(x, 0), P(x, 1), P(x, 2), \ldots$ for any given x. For example, $P(x, i) = h(x) + i$ (where h is the hashing function for the hash table in question) defines a linear probing strategy.

Let H be an empty hash table with $n = 6$ bins. Assume that H stores integers according to the hash function $h(x) = x$ and the probing function

$$P(x,i) = \begin{cases} (P(x,i-1)+i)\%n & i > 0 \\ h(x)\%n & i = 0 \end{cases}$$

So if an element x is added to H, $P(x, 0) = h(x)\%n = x\%n$ would be the first location in the hash table that is checked. If that location is occupied, $P(x, 1) = (P(x, 0) + 1)\%n = (x + 1)\%n$ would be checked next, followed by $P(x, 2) = (P(x, 1) + 2)\%n = (x + 3)\%n$, and so on until an empty location is found.

(a) Show the configuration of H after the integers 6, 1, 3, 10, and 4 are added (in that order).

(b) What happens when the table is checked for the inclusion of the integer 0? Explain your answer.

Problem 5

Consider the following algorithm:

```
TRAVERSE(t, visit)
1: if t = null then
2:     return
3: end if
4: TRAVERSE(t.left, visit)
5: visit(t)
6: TRAVERSE(t.right, visit)
```

where t is a rooted binary tree and *visit* is a constant-time function.

Using the master theorem, give a tight bound on the running time of TRAVERSE as a function of the number of nodes in t.

Problem 6

Consider the following recurrence relation:

$$T(n) = \begin{cases} 1 & n = 1 \\ 7T(\frac{n}{2}) + 1 & n > 1 \end{cases}$$

(a) Use the master theorem to show that $T(n) \in \Theta(n^{log_2(7)})$.

(b) Use induction to show that $T(n) = \frac{1}{6}(7n^{log_2(7)} - 1)$. You may assume that n is a power of two.

Problem 7

A *priority queue* is a data type that stores a collection of elements, each of which has an associated priority. It supports the following operations:

- enqueue(e, p), which stores e in the queue with priority p
- dequeue(), which removes the highest-priority element from the queue and returns it. In the case of a tie in highest priority, the oldest element is removed and returned.

(a) Provide pseudocode implementations of the enqueue(e) and dequeue() operations of a queue that uses a priority queue to store its data.

(b) Provide pseudocode implementations of the push(e) and pop() operations of a stack that uses a priority queue to store its data.

Problem 8

Consider the following algorithm:

```
QUICKSORT(list, start, end)
1: if end ≤ start then
2:     return
3: end if
4: pivot = SELECT-PIVOT(list, start, end)
5: mid = PARTITION(list, start, end, pivot)
6: QUICKSORT(list, start, mid - 1)
7: QUICKSORT(list, mid, end)
```

The running time of QUICKSORT depends on the behavior of SELECT-PIVOT. For this problem, assume that SELECT-PIVOT executes in constant time and PARTITION in time $\Theta(end - start)$.

(a) What SELECT-PIVOT behavior will lead to the worst case QUICKSORT behavior? Use the master theorem to derive a tight bound on QUICKSORT's worst case running time.

(b) What SELECT-PIVOT behavior will lead to the best case QUICKSORT behavior? Use the master theorem to derive a tight bound on QUICKSORT's best case running time.

Problem 9

A 10-foot rope is stretched horizontally across a gap. n ants are distributed on top of the rope, each facing along the rope in one of the two possible directions. At the same instant, the ants begin to walk, moving forward at a rate of 10 feet per minute. If two ants collide, they immediately reverse direction. When an ant reaches the end of the rope, it steps off the rope and can no longer block the other ants.

(a) What is the largest possible amount of time that can elapse before the last ant steps off the rope?

(b) In what situations does this worst-case time arise?