Test 2: CPS 100

Owen Astrachan

November 16, 1994

Name: ____________________________________________

Honor code acknowledgement (signature) ____________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>value</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>24 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>18 pts.</td>
<td></td>
</tr>
<tr>
<td>(extra)</td>
<td>6 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>54 pts.</td>
<td></td>
</tr>
</tbody>
</table>
PROBLEM 1:  (Tree Basics (24 points))

part A (6 points)
Give the pre-order, in-order, and post-order traversals of the binary tree diagrammed below:

```
   A
  / \  
 B   C
  /   /  
D   E   F
 /   /  
G   H
```

1. pre-order:

2. in-order:

3. post-order:

part B (6 points)
Assume the definition below is provided for implementing trees.

```
struct Tree{
    int info;
    Tree * left;
    Tree * right;
};
```

Write a function that returns the sum of all nodes in a tree that are “left-children”, i.e., that are pointed to by a left field rather than a right field.

```
int LeftSum(Tree * t)
// postcondition: returns sum of all nodes in t that are left-children
{
}
```
part C (6 points)
The root of a tree is at depth 0, its children at depth 1, and the children of a an n-depth node are at depth
\( n + 1 \). Write the function \texttt{PrintDepth} below that prints all nodes at depth \( d \). For example, for the tree on
the previous page, \texttt{PrintDepth(t, 2)} should print \texttt{D E F} (in some order).

\begin{verbatim}
void PrintDepth(Tree * t, int d)
    // postcondition: all nodes of t at depth d printed
{
}
\end{verbatim}

part D (6 points)
A \textit{level-order} traversal of a tree visits nodes by level, from left to right within a level. The level order traversal
of the tree diagrammed on the previous page is: \texttt{ABCDEF GH}.
Assume that there is a templated queue class \texttt{Queue}. Complete the code below that prints a tree in level
order. The class \texttt{Queue} supports operations \texttt{Enqueue(Type item)} and \texttt{Dequeue(Type & item)}.

\begin{verbatim}
void LevelOrder(Tree * t)
{
    if (t != 0){
        Queue<Tree *> q; // queue of Tree pointers

        q.Enqueue(t); // root is on queue to start
        while (! q.IsEmpty()){

        }
    }
}
\end{verbatim}
PROBLEM 2:  (Conversion (12 points))

Using the definitions below, write the function `DoTree` that converts a sorted array to a balanced binary search tree. The diagram below shows a tree constructed and returned by the call `Array2Tree(a, 10).

```
struct Tree{
  int info;
  Tree * left;
  Tree * right;
  Tree(int x, Tree * branchL = 0, Tree * branchR = 0);
};

Tree::Tree(int x, Tree * branchL, Tree * branchR)
{
  info = x;
  left = branchL;
  right = branchR;
}

Tree * DoTree(int a[], int first, int last)
// precondition: a[first] <= a[first+1] <= ... <= a[last]
// postcondition: returns balanced search tree containing
// a[first]...a[last]
{
  // you'll write this function
}

Tree * Array2Tree(int a[], int size)
// precondition: a is sorted
// postcondition: returns balanced search tree containing elements of a
{
  return DoTree(a, 0, size-1);
}
```
part A (9 points)
Write the function DoTree below.

Tree * DoTree(int a[], int first, int last)
// precondition: a[first] <= a[first+1] <= ... <= a[last]
// postcondition: returns balanced search tree containing
// a[first]...a[last]
{
}

}  

part B (3 points)
What is the complexity of the code you wrote above? Express your answer using $O$-notation for an $n$-element array being converted to a tree and briefly justify your answer. Would the complexity change if your code was modified to turn a sorted linked list into a tree? Why?
PROBLEM 3: (Heaps (18 points))

part A (6 points)
In the heap implementation of a priority queue, the elements in the heap are stored in an array that represents a binary tree. Draw a heap (binary tree) containing the elements below and draw the array with the elements stored in the appropriate positions for the heap you draw.

5 10 15 20 25 30 35 40

part b (6 points)
Draw the heaps (trees) that result from (a) inserting the element 17 into your heap above and (b) deleting the minimal element from the heap you drew above [the two heaps you draw should be related to the heap you drew above and not to each other].

part c (6 points)
You have been asked to write code to monitor voltage readings coming in from a source at the rate of 50 readings per second. At the end of every second your program must print the smallest voltage of ALL voltages read so far. Explain how a heap-based implementation of a priority queue can be used to solve this problem quickly with minimal storage — i.e., never storing more than roughly 50 voltage readings at a time.
PROBLEM 4: (Extra Credit)

What is the output of the program below and why?