Programming Assignment: Ternary Trees

1. The problem

Generalize the idea of binary trees. First, consider **unary** trees: Each node has at most one child. Unfortunately, unary trees are just linked lists. Next, consider **ternary** trees. After all, if binary trees are enormously better than unary, certainly ternary will be better still!

Each node in a non-empty ternary tree is allowed to have either 1 or 2 values. If a node has 2 values, say $a < b$, then it is easy to use them to choose among three directions to descend below that node given a probe value p. There are three possibilities: $p \le a$, $a < p \le b$, $b < p$. But what if a node only has 1 value, say a? If you are inserting p, make p the other value, making sure that you sort the two values with the lower one first. In other words, only two-valued nodes can have children.

To test this idea, write a program called `ternary` that takes n as a parameter, reads in n integers (one per line; there may be duplicates), inserts them into an initially empty ternary tree, and then prints the tree in symmetric order (inorder), with parentheses showing the children of a node. That is, if a node contains values a and b, print it as

```
(left child) a (middle child) b (right child)
```

If any child is empty, just leave out the parentheses. For instance, if I insert the following 20 integers into an initially empty ternary tree:

```
4 6 4 18 8 2 14 7 15 5 19 12 15 5 9 0 17 2 2 19
```

the output is

```
((0 (2) 2) 2 4) 4 (5 5) 6 ((7) 8 ((9 12) 14 (15) 15 (17)) 18 (19 19))
```

2. What to hand in

Hand in a copy of your program and its output when run as follows:

```
randGen.pl 49 10000 | ternary 1000
```

3. Useful tools

You have access to some useful tools. First, there is a sample `Makefile` at http://www.cs.uky.edu/~raphael/courses/CS315/prog2/Makefile. It has a *run* target that compiles the program (either `ternary.c` or `ternary.cpp`) and runs it. Feel free to modify `Makefile`.

Second, you have the `randGen.pl` program from before. It takes two parameters; a randomizing seed and a value that limits the size of its outputs.

Third, there is a working version of the program at http://www.cs.uky.edu/~raphael/courses/CS315/prog2/workingTernary.

4. Extra credit ideas

1. Compute statistics for large randomly built ternary trees to estimate the function relating number of elements in the tree to the average depth of a node in the tree. The function ought to be no worse than logarithmic, but is it better than the equivalent function for binary trees?

2. Experimentally compare the time and space required to build a ternary versus a binary tree on a large list of randomly generated values.
3. Build a sorting routine that uses your ternary-tree insertion code as a subroutine.
4. Build a search routine that takes a probe value *p* and looks for it in a ternary tree.
5. (Hard) Derive a formula for the number of ternary trees with *n* values.
6. Generalize the program so it takes another parameter indicating the *arity* of the tree; 2 means binary, 3 means ternary, 4 means quaternary, and so forth.