Poorly Balanced Binary Search Trees

The same keys might be arranged to form a "perfectly unbalanced tree."
Rotation preserves the binary search tree property.
A single left rotation on the original root („3„) of the tree produces:

A Single Rotation Applied
Multiple Rotations Applied

Performing a left rotation on alternating nodes (3, 31, 41) or

the original tree produces:

69
41
33
22
31
16
3

April 16, 2001
public class Rootlet implements IVisitor

package binarytreevisistor;

A BinaryTree Visitor For Left Rotation
{ return right;

right = host.rightSubtree(right);

host = host.rightSubtree();

right = host.rightSubtree(right);

public Object objectNonEmptyCase(right host, Object input)
property on the priority. – and that we maintain the BST property on the key and the heap.

Suppose that each node has a distinct key and priority.

An Idea: The Treap
Following treap.

- For example, suppose we had inserted (key=3, priority=6) into the parent, lifting the node above its parent.
- If the node's priority is less than its parent's priority, rotate around the parent.
- Insert the node (key, priority, object) by key, just like a BST.

The insertion procedure is straightforward.

An Idea: The Treap (cont.)
What happens if we insert (key=49, priority=19) into the following?

An Idea: The Heap (cont.)
- What happens? Note: ignoring the priority, this produces a worst-case BST.

Suppose that we insert (key=3, priority=56), (16,33), (22,89), (31,17), (41,49), (53,22), and (69,36) in order of increasing key.
The Punchline (cont.)
The Punchline (cont.)

Suppose that you don't require the priority for your application. The insertion of the punchline: The same treap will result regardless of the order of A "hash" of the key.

- A randomly generated number
- A priority can be: