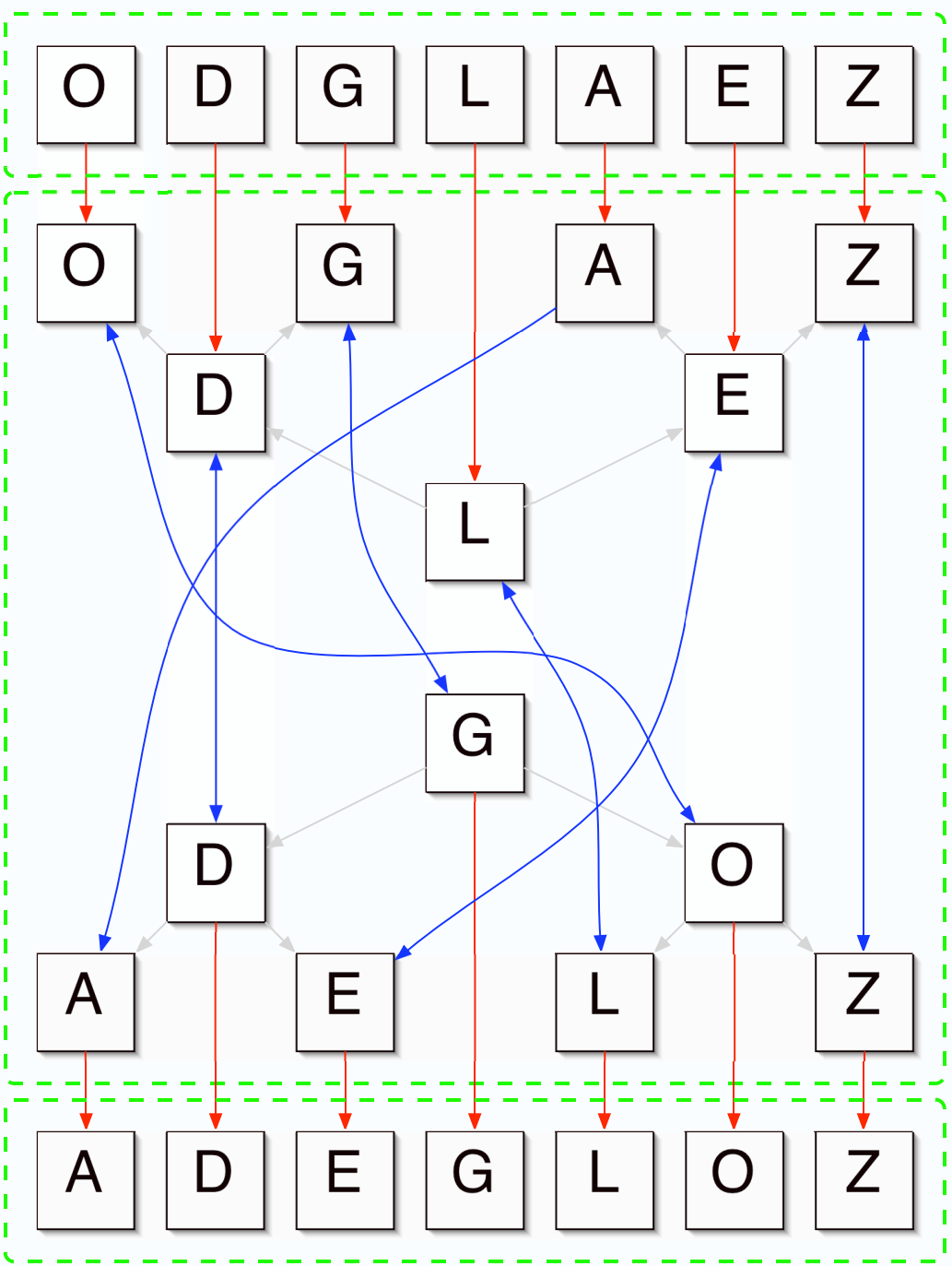
Glazed Lists: Sorting Dynamic Data Efficiently

November 2004
http://www.publicobject.com/glazedlists/

Jesse Wilson
Software Developer
O'Dell Engineering Ltd.
jesse@odell.ca



Sorting is a well understood problem in computer science. There exist many algorithms to efficiently sort a list of objects. All of these algorithms work on the premise that the data to be sorted is static. Therefore when only a single element is changed it is necessary to sort the entire data set again.

Because Glazed Lists manages a dynamic list of data, sorting the entire data set on every change is not satisfactory. This requirement has prompted the development of a new strategy that can efficiently sort live data.

To implement a live sorting strategy it is necessary to create a map between the sorted and unsorted data.

The simplest solution is to implement this map using an array of integers. To get the sorted index from the unsorted index, simply fetch the value in the map at the unsorted index. Unfortunately this solution does not scale well for lists with a large number of elements. When a single element is inserted into the unsorted list, it will be given a new index into the sorted list. The insert requires all indices in the map to be shifted. For a list of N elements, a single insert costs O(N) operations.

To overcome this limitation, Glazed Lists' SortedList uses a pair of balanced trees to implement the map. The first tree mirrors the source list's unsorted ordering. Each node in this tree points to a corresponding element in the second tree. The second tree tracks the source list's elements in sorted order. Its nodes include pointers back to the corresponding nodes in the first, unsorted tree. Both trees support operations to get the node at a specified index, and to the get index for a specified node. All tree operations cost O(log N). To use these trees to get the sorted index from the unsorted index, find the node in the unsorted tree with the specified index. Use this to get the corresponding node in the sorted tree. The sorted index is then found by getting the index of the sorted node.

The trees support insert, remove and set operations all with O(log N) cost for a tree of size N. In the diagram, the top box shows a unsorted list and the bottom box shows the corresponding sorted list. The middle box shows the two trees and the pointers between their nodes.

This mechanism gives Glazed Lists high-performance sorting when the source data is dynamic.