COMP20007:

Design of Algorithms

Assignment 2

Name	Student ID Number
Callum Johnson	910519

3. Sorting Algorithm Problems:

a. A variation of Merge Sort – Since each box is already sorted, once all the boxes have been scanned, we partition the array corresponding to each box's relative books and run 'MERGE' on each partition. This method is ideal since it preserves ordering of duplicate books relative to each other, and since the boxes are already sorted, this algorithm will run faster than a normal divide and conquer style sorting method, making it very quick.

b. Selection Sort is recommended, because although it is not the fastest algorithm it guarantees at most n swaps. This is more efficient in minimizing swaps than other methods, since once an element is swapped into its correct position it is not swapped again.

c. Counting Sort is recommended, since we know that the luminosity values can only be one of a fixed set of values, we can use the extra memory space to create a frequency table and then send iterate the sorted array over the original array before sending. This is extremely fast, completing in theta-n time in every case.

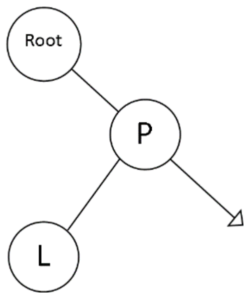
4.
```
function STICKIFY(parent, new):
    if new.value < parent.value then
        ROTATERIGHT(parent)
```

Stickify works by maintaining the search tree such that it is only possible for a node to have a right child. If a node is added to the right of an existing node, then STICKIFY doesn't need to make any changes, however if a left branch is added STICKIFY forces the tree to place the new node into the 'right stick'

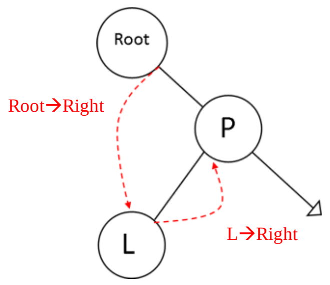
To show this in detail we will look at a node P and its left child, node L. We assume that stickify has been called as soon as L is added to the tree, resulting in the following function call:

→ STICKIFY(P, L):

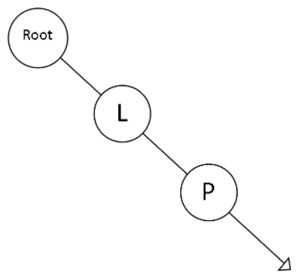
Since L was placed as the left child of P, we know that L.value < P.value so the 'if' condition of STICKIFY is satisfied. We perform a right rotation around P, so that now the node originally above P is pointing to L, and L's right pointer is pointing to P. Even if P already had some right children, these pointers are maintained and so the search tree becomes a 'right stick'. This situation is shown in the diagram below



1. L is added to an existing tree, STICKIFY is called



2. Since L.data < P.data, ROTATERIGHT(P) is called, with the updated pointers shown in red



3. Now the tree has been left as a 'right stick'