

Homework 3

Due March 13

Problem 3.1

At which nodes of a heap can an entry with the largest key be stored? Explain.

Solution: The entry with the largest key must be stored at a leaf. Otherwise it would have a child with a smaller key, violating the heap-order property.

Problem 3.2

Suppose two binary trees, T_1 and T_2 , hold entries satisfying the heap-order property. Describe a method for combining T_1 and T_2 into a tree T whose internal nodes hold the union of the entries in T_1 and T_2 and also satisfy the heap-order property. Your algorithm should run in time $O(h_1 + h_2)$ where h_1 and h_2 are the respective heights of T_1 and T_2 . You may assume that all entries in the two trees are distinct. Note that the trees need only have the heap-order property; neither T_1 , T_2 , nor the result are necessarily left-full.

Solution:

Algorithm 3.2.1 $\text{combine}(t_1, t_2)$

```
let  $t_s$  be whichever of  $t_1$  and  $t_2$  has the smaller key at its root
let  $t_b$  be whichever of  $t_1$  and  $t_2$  has the bigger key at its root
if  $t_s$  has 0 or 1 children then
    make  $t_b$  be a child of  $t_s$ 
else
    let  $t_l$  be the left child of  $t_s$ 
     $t^* \leftarrow \text{combine}(t_b, t_l)$ 
    make  $t^*$  be the left child of  $t_s$ 
end if
return  $t_s$ 
```

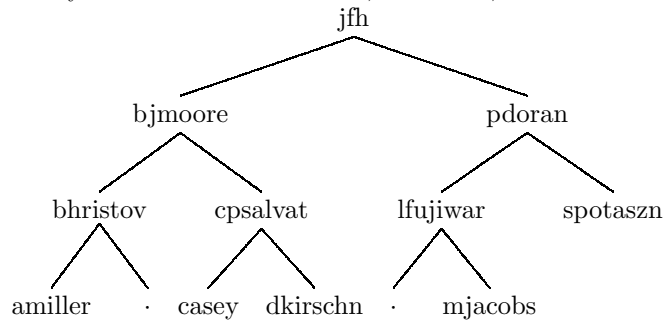
Problem 3.3

The number of steps that a sorting algorithm must execute depends not only on the length of its input, but also on the ordering of the input. Provide a list of five integers that maximizes the number of steps executed by insertion sort. That is, no other list of five integers would cause insertion sort to execute more steps. Similarly, provide a list of five integers that maximizes the number of steps executed by selection sort.

Solution: For insertion sort, any list of five integers in reverse order takes the maximal number of steps. For selection sort, any list of five integers takes the the same number of steps.

Problem 3.4

A binary search tree (BST) is one in which each node is greater than all its left descendants and less than all its right descendants, like the one shown below. In this case, the node contents are strings, and we're using dictionary order. We'll assume throughout this problem that all the entries are distinct: the same string never appears twice. (Note that we draw trees so that a non-leaf node with only one child has a null leaf, denoted \cdot , as its other child.)



- (a) Write the sequence of nodes encountered in an inorder traversal of this tree.

Solution: amiller, bhristov, bjmoore, casey, cpsalvat, dkirschn, lfujiwar, mjacobs, pdoran, spotasz

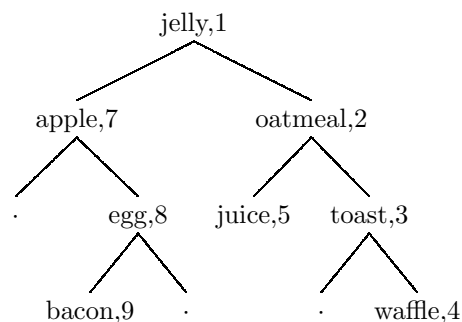
- (b) Explain why the inorder traversal of any BST is always in sorted order.

Solution: All left children of a node precede its root, which in turn precedes all of its right children, so the inorder traversal of every node obeys the ordering.

Consider the following data, where each datum consists of a key (a string) and a number:

(toast, 3) (apple, 7) (egg, 8) (waffle, 4) (jelly, 1) (juice, 5) (bacon, 9)
 (oatmeal, 2)

We can put these into a binary tree as shown below:

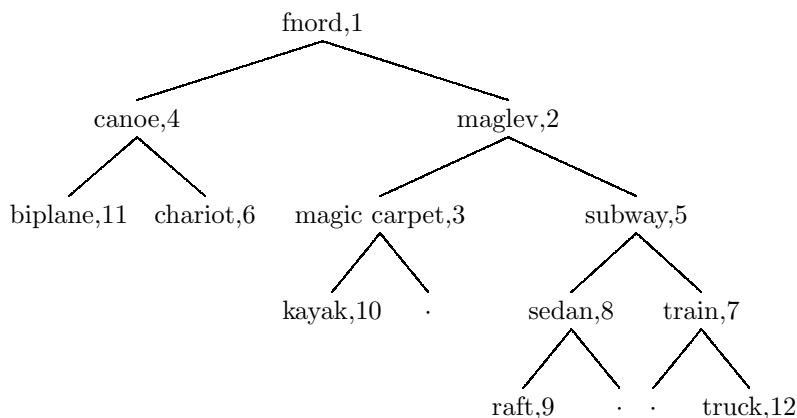


We've constructed the tree cleverly, so that it is a BST when we look at only the keys, and it's in heap-order when we look at only the numbers (i.e., each node's number is less than the numbers of its children).

- (c) Construct a tree with the same properties, but using the following data instead:

(train, 7) (truck, 12) (canoe, 4) (fnord, 1) (sedan, 8) (maglev, 2)
 (kayak, 10) (magic carpet, 3) (raft, 9) (subway, 5) (chariot, 6) (biplane, 11)

Solution:



- (d) Show that if you're given a list of key-number data with all the keys distinct, and all the numbers distinct, there's *always* a tree that's a BST with respect to the keys, and in heap order with respect to the numbers. Do this by writing pseudocode for constructing such a tree. Assume that the input is provided as a list of pairs, and that a pair has `getKey` and `getNum` methods. You may assume that you can compare strings or numbers with "`<`" or "`>`".

Solution: Your pseudocode could take this approach: First, sort the input by number (in ascending order). Then, starting with an empty tree, iterate through the sorted input, inserting each pair in the tree. Inserting a pair into an empty tree is just setting the root to be the pair. Inserting a pair into a non-empty tree involves comparing the pair's key to the root's key; if the pair's key precedes the root's key, insert the pair into the left child of the root; otherwise, insert the pair into the right child of the root.

- (e) Explain why the tree constructed in part (d) is unique.

Solution: Each subtree has the properties that

- its root is uniquely determined (so that the heap property is preserved), and
- no left-descendent could be a right-descendent or vice versa (so that the BST property is preserved).

Because these properties must hold for every subtree, no alteration of the constructed tree could satisfy them.