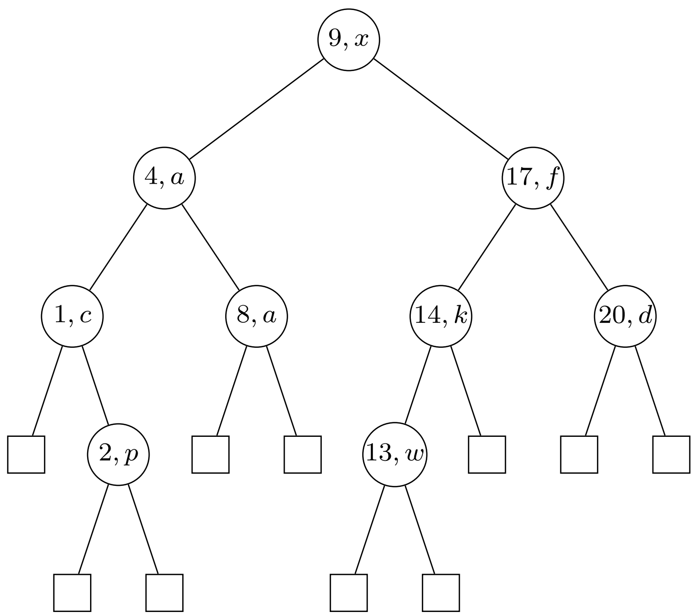
University of Houston

COSC 3320: Algorithms and Data Structures
Spring 2016

Solutions for Homework 6

1. Insert, in this order, the following entries in an initially empty binary search tree: $(9, x), (4, a), (17, f), (1, c), (8, a), (14, k), (20, d), (2, p), (13, w)$. You are to draw the final binary search tree.

   Solution:



2. Let T be a binary search tree which implements a dictionary. Let v be a node of T, and T_v be the subtree rooted at v. Design a recursive algorithm `CountLE`(v, k) which, given an input node v and a key k, returns the number of entries in T_v with key at most k.

   Solution:

```
CountLE(v,k)
input: node v of a BST T, key k
output: number of entries with key <= k in T_v
if (T.isExternal(v)) then return 0
q <- v.element().getKey()
```

```
if (q > k) then return CountLE(T.left(v),k)
else return 1 + CountLE(T.left(v),k) + CountLE(T.right(v),k)
```

3. Design and analyze a simple and efficient non-recursive algorithm to determine the height of a $(2,4)$-tree.

   Solution:

```
24height(T)
input: (2,4)-tree T
output: height of T
h <- 0
v <- T.root()
while (T.isInternal(v)) do
  v <- any child of v
  h <- h+1
return h
```

   Since outside the while and in each of its iterations the algorithm performs $O(1)$ operations, the complexity of the algorithm is proportional to the number of iterations of the while. At each of such iterations the algorithm goes down by one level in the tree T, and the while stops when v reaches a leaf. Hence the complexity is proportional to the height of the tree T, that is, $\Theta(\log n)$.

4. Let T be a $(2,4)$-tree containing n entries with distinct, integer keys. Suppose every node $v \in T$ maintains a variable $v.size$ that stores the number of entries contained in the subtree rooted at v (denoted T_v), included the entries in v. Design a recursive algorithm `Count` which, given an integer k, returns in $O(\log n)$ time the number of entries in T with key less than k.

   Solution:

```
Count(T,v,k)
if (T.isExternal(v)) then return 0
else
  let (k_i,x_i), 1 <= i < d be the entries in v
  let k_0 = -infinity and k_d = +infinity
  let v_i, 1 <= i <= d, the children of v
  find i such that k_{i-1} < k <= k_i
  if k = k_i then return (i-1) + \sum_{j=1}^{i} v_j.size
  else return (i-1) + \sum_{j=1}^{i-1} v_j.size + Count(T,v_i,k)
```

   The algorithm is invoked with `Count(T,T.root(),k)`.