University of Waterloo
CS240 - Spring 2014
Assignment 4

Due Date: Wednesday July 9 at 09:15am

Please read `http://www.student.cs.uwaterloo.ca/~cs240/s14/guidelines.pdf` for guidelines on submission.

Problem 1 [4+4+4+4 marks]

(a) Give a recursive algorithm for finding the height of an arbitrary binary tree. Determine and justify your algorithm's running time, using O-notation. Let this running time be $O(f(n))$.

```
int height(root)
{
   if (! root)   // If root is not a null pointer ...
                 // Indeed (! root) is equivalent to (root != NULL) in C++
     return -1   // In this course, the height of a node is by definition 0
                 // Some people defines this as 1. We accept both answers
   else
     return 1+max(height(root->left), height(root->right))
}
```

For any tree with n nodes, our algorithm takes time 1 plus a recursive call to the left subtree and a recursive call to the right subtree. Therefore the worst-case time satisfies $f(n) \leq \max_k\{1 + f(k) + f(n-k-1)\}$, where k denotes the number of nodes in the left subtree. By induction, we can easily show that $f(n) \leq n$ and so $f(n) = O(n)$.

Two marks for code, 1 mark for recurrence, 1 mark for solution.

(b) We want your bound above to be tight so give an example where your algorithm of part (a) takes time $\Omega(f(n))$. Deduce then that the worst case running time is $\Theta(f(n))$.

Consider a simple path, *e.g.* a tree with only right children. In this case the algorithm recurses down the path and takes time $\Omega(n)$. Since the algorithm is known to be $O(n)$ then it is $\Theta(n)$.

2 marks for giving a proper example, 1 mark for noting is $\Omega(n)$ and 1 mark for deducing is $\Theta(n)$.

(c) Give a recursive algorithm for finding the height of an AVL tree. Determine the running time $O(g(n))$ of your algorithm, and show that $g(n) = o(f(n))$.

```
int height(root)
{
   if (! root)
      return -1
   else
   if root->balance > 0
     return 1+height(root->right)
   else
     return 1+height(root->left)
}
```

The worst-case time $T(h)$ as a function of the height satisfies $T(h) \leq 1+T(h-1)$, where h is the height of the AVL tree. Therefore $T(h) = O(h)$.

Since we know $h = O(\log n)$ we have that the time $g(n)$ to compute the height of an AVL tree with n nodes is $g(n) = O(\log n)$ which implies $g(n) \in o(f(n))$.

One mark for code, 1 mark for recurrence, 1 mark for solution, 1 mark for $g(n) = o(f(n))$.

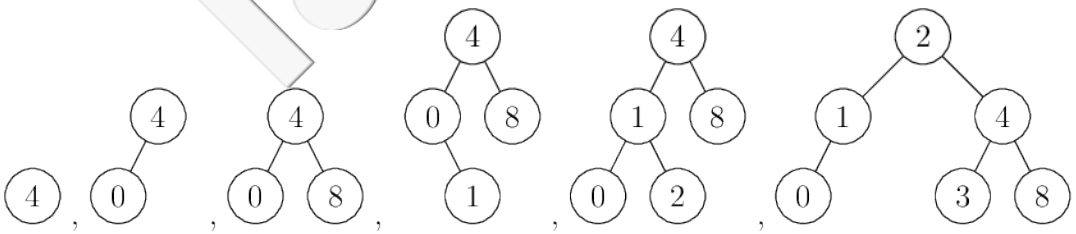
(d) Give an example where your algorithm of part (c) takes time $\Omega(g(n))$. Deduce then that the worst case running time is $\Theta(g(n))$.

If $R(h)$ is the runtime on a perfectly balanced tree of height h, then $R(h) = 1+R(h-1)$ and so $R(h) = h$ and $T(h) \geq R(h) = h$. Since $h \in \Omega(\log n)$, we have $g(n) \in \Omega(\log n)$ and therefore $g(n) \in \Theta(\log n)$.

2 marks for giving a proper example, 1 mark for noting is $\Omega(\log n)$ and 1 mark for deducing is $\Theta(\log n)$.

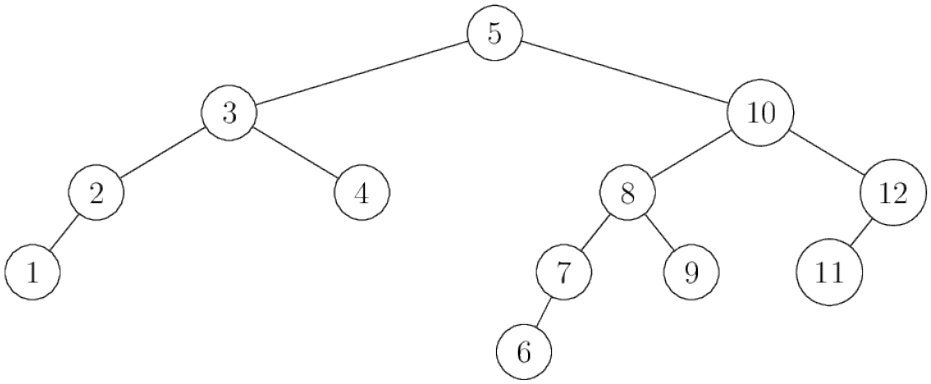
Problem 2 [8+4 marks]

(a) Beginning with an empty AVL tree, insert the keys 4, 0, 8, 1, 2, 3. Show the tree that results after inserting each key.

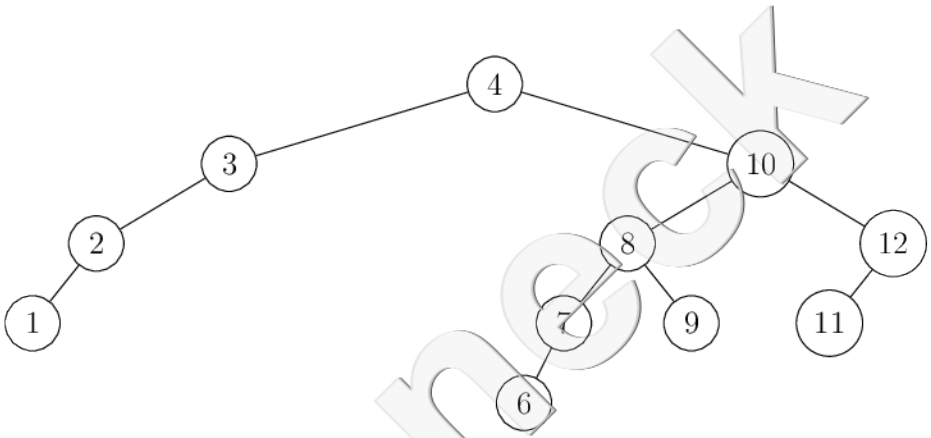


Five marks for correct trees, one mark for each correct rotation (double rotation counts as two)

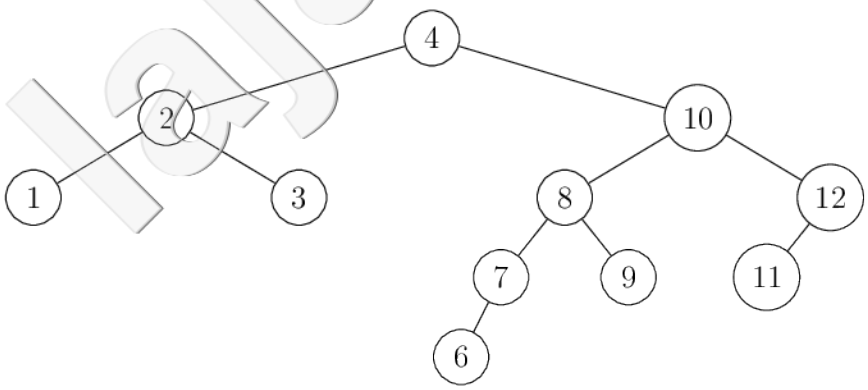
(b) Delete the key 5 from the following tree using in-order predecessor for swaps. Show the 2-3 tree that results at each step (after or before each rotation).



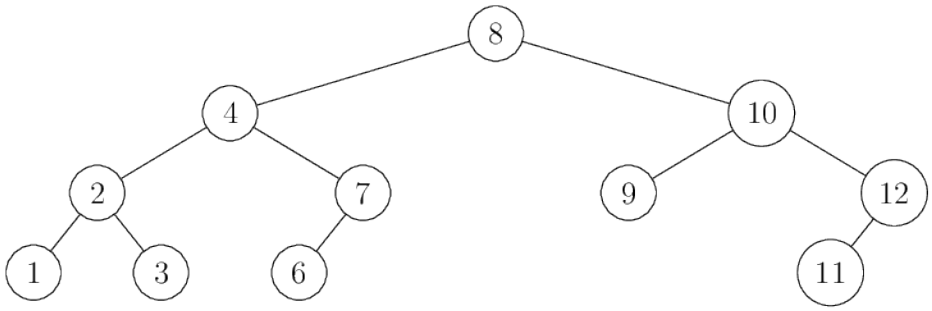
Swap



Simple right rotation



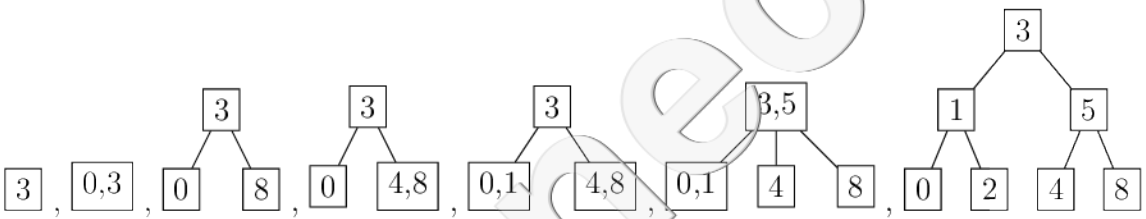
Double left rotation



One mark for swap, one for simple right rotation and two for double left rotation.

Problem 3 [7+4 marks]

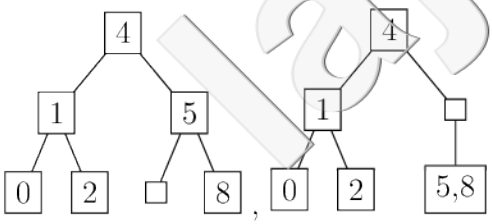
(a) Beginning with an empty 2-3 tree, insert the keys 3, 0, 8, 4, 1, 5, 2. Show the tree that results after inserting each key.



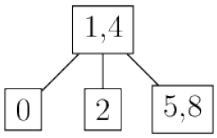
4 marks for the right trees, 1 mark for each correct split

(b) Delete the key 3 from your answer to part (a), replacing it with its in-order successor and show the 2-3 tree that results.

Intermediate steps (not required)



Final tree



1 mark for the right tree, 1 mark for moving the right key, 1 mark for each tree adjustment/merge

Problem 4 [8 marks]

Consider a variant of AVL trees that uses size instead of height as balancing parameter. Each node has a leftDescendants, a rightDescendants and a balance field containing the number of descendants in each of the subtrees as well as the balance defined as the ratio between the left and right subtree sizes, i.e.

$$balance = (rightDescendants + 1)/(leftDescendants + 1).$$

The idea now is that a tree is "balanced" if the left and right sides are within a multiplicative factor of two of each other, i.e $1/2 \leq balance \leq 2$.

(a) Prove that balanced trees have height $\Theta(\log n)$ where n is the number of keys in the tree.

Let ld and rd denote the number of left and right descendants. We know that $rd+ld+1 = n$. Using the balance equation $(ld+1) \geq 1/2\,(rd+1)$ we get, $n = rd+ld+1 \geq rd+1/2\,(rd+1)$ and therefore $rd \leq 2/3\,(n - 1/2) \leq 2n/3$ and the same for ld.

Let $D(n)$ denote the height of the tallest size-balanced tree on n nodes. Then it follows that $D(n) \leq 1 + \max_{ld,rd}\{D(ld), D(rd)\} \leq 1 + D(2n/3)$ since D is an increasing function.

Hence $D(n) = O(\log_{3/2} n) = O(\log n)$. Now every binary tree has height $\Omega(\log n)$ and finally

$$D(n) = \Theta(\log n)$$

.

3 marks for showing that the balance condition implies that sizes of trees are at worst 1/3rd-2/3rds imbalanced.

3 marks for setting up the recurrence

2 marks for arguing that the solution is $O(\log_{3/2} n)$

Problem 5 [9 marks]

Suppose the leaf pages in extendible hashing can hold at most 3 keys. Show the result after keys with the following hash values are inserted in this order:

00001, 10000, 10001, 00011, 01111, 00000, 11010, 11111, 10111.

You need only show the extendible has table after either a local or global depth value changes. Also show the final extendible hash table.

$[0, 1] \rightarrow (0)[00001, 00011](1)[10000, 10001]$
$[00, 01, 10, 11] \rightarrow (00)[00000, 00001, 00011](01)[01111](10, 11)[10000, 10001]$
$[00, 01, 10, 11] \rightarrow (00)[00000, 00001, 00011](01)[01111](10)[10000, 10001](11)[11010, 11111]$
$[00, 01, 10, 11] \rightarrow (00)[00000, 00001, 00011](01)[01111](10)[10000, 10001, 10111](11)[11010, 11111]$
1 mark per insertion plus one mark per proper split