Homework 2
Fundamental Algorithms, Spring 2003, Professor Yap

Due: Feb 24, in class

INSTRUCTIONS:

- Please read questions carefully. When in doubt, please ask.

---

1. (20 Points)
   Solve using the Master Theorem:
   (a) $T(n) = 3T(n/8) + 1$.
   (b) $T(n) = 3T(n/8) + \sqrt{n}\log n$.
   Be careful to justify all conditions required by the Master Theorem.

   SOLUTION:

   The watershed function in each case is $w(n) = n^{\log_8 3}$.

   (a) Since $f(n)$ in this case is 1 and 1 is polynomially smaller than $w(n)$ we can apply the first case of the Master Theorem, so $T(n) = \Theta(w(n))$.

   (b) In this case since $\log_8 3$ is more than $1/2$ and since $\sqrt{n}\log(n)$ is polynomially smaller than $w(n)$, we again apply case 1 of the Master Theorem and conclude that $T(n) = \Theta(w(n))$.

2. (10 Points)
   Give tight upper and lower bounds for the following:
   (a) $T(n) = 4T(n/2) + n^2/\log n$.
   (b) $T(n) = 4T(n/2) + n^2\log n$.
   NOTE: we DO NOT want you to transform these recurrences to solve them, nor to use real induction. Just use the Master theorem, or from remarks like $T(n) = \Omega(d(n))$ where $d(n)$ is the driving function.

   SOLUTION: Make sure you understand this type of exercise, because it captures a series of useful tricks!

   (a) We have:
   Upper Bound: We claim that the upper bound is $T(n) = O(n^2\log n)$. To see this, use the Master theorem. The watershed function in this case is $w(n) = n^{\log_2 4} = n^2$ and $f(n) = n^2/\log n$. So we can't apply the Master Theorem directly, but we can do the following trick: $T(n) = 4T(n/2) + n^2/\log n \le 4T(n/2)+n^2$. And now we can apply the Master Theorem for the new recurrence $T_1(n) = 4T_1(n/2)+n^2$. And this is a very nice recurrence, we can solve it quickly by applying the Master theorem. So $w(n) = n^2$ and also $f(n) = n^2$ so we can apply case two of the MT and get $T_1(n) = \Theta(n^2\log n)$. And since $T(n) < T_1(n)$, then we conclude $T(n) = O(n^2\log n)$.

   Lower Bound: We claim that $T(n) = \Omega(n^2)$. First, it is usefule to note a trivial lower bound, namely, $T(n) = \Omega(n^2/\log n)$, since this is just the driving function. But we can do a little better by invoking the Master Theorem: we have $T(n) = 4T(n/2) + n^2/\log n \ge 4T(n/2) + 1$. Hence if we let $T_2(n) = 4T_2(n/2) + 1$, then $T(n) \ge T_2(n)$. But the MT tells us that $T_2(n) = \Theta(n^2)$. Hence $T(n) = \Omega(n^2)$.

   (b) In this case, we have the driving function $f(n) = n^2\log n$ and $w(n) = n^2$. Clearly, for any $\varepsilon > 0$, $T(n) \le 4T(n/2) + n^{2+\varepsilon} = T_3(n)$. But the latter recurrence has a solution (using Case (+1) of MT) of $T_3(n) = \Theta(n^{2+\varepsilon})$. Hence $T(n) = O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.

   Actually, for both (a) and (b) we know the Θ-order of the solution. They are $T(n) = \Theta(n^2\log\log n)$ and $T(n) = \Theta(n^2\log^2 n)$, respectively. These can be proved using our transformation methods. However, we do not expect you to know this or to use it for this problem.

3. (15 Points)
   Let $T(n) = 2T(\frac{n}{2} + c) + n$ for some $c > 0$. You are told that this recurrence holds for $n \ge n_0$, and that for all $n < n_0$, we have $T(n) \le C$ for some constant C. Prove that $T(n) \le D(n-2c)\lg(n-2c)$ for D large enough. As usual, lg means $\log_2$. Describe carefully how you would choose D. NOTE: Thus we conclude that $T(n) = O(n\log n)$, which is the point of this exercise.

SOLUTION: This exercise requires a little bit of analysis to get the right values for D and also for the starting interval of the real analysis. Note that the inequality does not hold for every n. For instance we can not have $n \le 2c$, because otherwise, $\log(n-2c)$ is undefined. Therefore we will take as the starting point of the induction $n_1 = \max\{n_0+1, 2c+2\}$. The reason we want the "+2" is because we want $\lg(n_1-2c)$ to be positive (recall that $\log x > 0$ iff $x > 1$). The reason we want the "+1" is because we want $n_1 > n_0$.

So we will take the interval $[n_0, n_1]$ to be the base for our real induction. Note that n_1 is greater than $(n_1/2)+c$ by at least 1. Hence we can choose $\delta = 1$.

Real Basis step We claim that there exists a D_0 such that on the interval $[n_0, n_1]$, the hypothesis holds with $D = D_0$:

$$T(n) \le D_0(n-2c)\lg(n-2c) \tag{1}$$

This is clear because $\lg(n-2c)$ is positive and so we just make D_0 large enough. Then we choose $D = \max\{D_0, D_1\}$ where D_1 is specified below.

Real Induction step Assume that (1) holds for $n \in [n_0, N]$ with $N \ge n_1$. Now we show that it holds on the interval $[N, N+\delta]$. Let n be in $[N, N+\delta]$. Then $\frac{n}{2}+c$ is in the interval $[n_0, N]$ (see above). So we can apply the induction hypothesis.

$$T(\frac{n}{2}+c) \le D((\frac{n}{2}+c)-2c)\lg((\frac{n}{2}+c)-2c) = D(\frac{n}{2}-c)\lg(\frac{n}{2}-c)$$

Now we estimate $T(n)$.

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}+c)+n \\ &\le 2D(\frac{n}{2}-c)\lg(\frac{n}{2}-c)+n \\ &= D(n-2c)\lg(n-2c) - (D(n-2c)-n) \\ &\le D(n-2c)\lg(n-2c) \end{aligned}$$

where the last inequality is true provided $D(n-2c) \ge n$ for all $n \ge n_0$. So we need $D \ge n/(n-2c)$. But the righthand side is decreasing, and so it suffices that $D \ge n_1/(n_1-2c)$. Choose $D_1 = n_1/(n_1-2c)$.

4. (20 Points)
(a) Prove that $S(n) = \sum_{i=1}^{n} \frac{\log i}{i} = \Theta(\log^2 n)$.
(b) Let $T(n) = aT(n/b) + w(n)\frac{\log\log n}{\log n}$ where $w(n) = n^{\log_b a}$ is the watershed function. Prove that $T(n) = O((\log\log n)^2)$. Prove that $T(n) = \Theta(w(n)(\log\log n)^2)$. You must use range and domain transformation methods to solve this. You can combine the two transformations into one, by defining, for all $k \ge 0$,

$$s(k) = \frac{T(b^k)}{a^k}.$$

SOLUTION:
(a) We have to prove two parts : $S(n) \in O(\log^2 n)$ and $S(n) \in \Omega(\log^2 n)$.
The first part is easy : just use $\log i \le \log n$ for all $i \le n$. So

$$S(n) \le \log n(\sum_{i=1}^{n}\frac{1}{i}) = \log^2 n + c\log n$$

where c is a constant.
For the second part, we have to find an n_0 such that $S(n) \ge \sum_{i=n_0}^{n} \frac{\log i}{i}$ and the RHS of this sum is $\Theta(\log^2 n)$. What candidates do we have for n_0? Well, we can think of $\frac{n}{2}$, but this won't give us what

we want, so we try the second candidate $\sqrt{n}$ and this is just the right value. So

$$S(n) \geq \sum_{i=\sqrt{n}}^{n} \frac{\log i}{i} \geq \log\sqrt{n}(\sum_{i=\sqrt{n}}^{n} \frac{1}{i}) = \log\sqrt{n}(H_n - H_{\sqrt{n}-1}) \geq$$
$$\geq \log\sqrt{n}(\log n - \log\sqrt{n} + const) = \frac{1}{2}\log n(\frac{1}{2}\log n + const) = \Theta(\log^2 n)$$

So we proved both parts and we are done!
(b) Using the hint we take:

$$s(k) = \frac{T(b^k)}{a^k}.$$

With this transformation we get:

$$s(k) = s(k-1) + \frac{1}{a^k} w(b^k) \frac{\log\log b^k}{\log b^k}$$

And this recurrence is easy to solve, just write it in the form:

$$\begin{aligned}
s(k) - s(k-1) &= \frac{1}{a^k} w(b^k) \frac{\log\log b^k}{\log b^k} \\
s(k-1) - s(k-2) &= \frac{1}{a^{k-1}} w(b^{k-1}) \frac{\log\log b^{k-1}}{\log b^{k-1}} \\
\cdots & \\
s(1) - s(0) &= \frac{1}{a} w(b) \frac{\log\log b}{\log b}
\end{aligned}$$

Now add all this things up and get:

$$s(k) = \sum_{i=1}^{k} \frac{1}{a^i} w(b^i) \frac{\log\log b^i}{\log b^i} + s(0)$$

Observe that $w(b^i) = a^i$. So we get:

$$s(k) = \sum_{i=1}^{k} \frac{\log i + \log\log b}{i \log b} = \frac{1}{\log b}\sum_{i=1}^{k} \frac{\log i}{i} + \frac{\log\log b}{\log b}\sum_{i=1}^{k} \frac{1}{i} = \Theta(\log^2 k)$$

using part (a). Now we know that $T(b^k) = a^k s(k)$, so we deduce that $T(n) = \Theta(w(n)(\log\log n)^2)$.

5. (15 points) (a) Implement the rotation operation in pseudo-code. Make the following assumption: each node u has three pointers, $u.Parent$, $u, Left$ and $u.Right$, with the obvious meaning. $u.Parent = null$ iff u is the root, and $u.Left = u.Right = null$ iff u is a leaf.
(b) How may assignments did you perform in (a)? Can you argue that you have used the minimum number of assignments possible?

SOLUTION:

(a) Consider the description of rotation in the Lecture notes, where we want to rotate u and the parent of u is v.

Let a pair of pointers, from any node u to any v and a pointer from v to u, be called a "link-pair". You need to change 3 such link-pairs: between u and v, between v and its parent, and between u and its right child (assuming u is the left child). Thus we need to update six pointers.

We do it in the following way.

u.parent.left:=u.right – v.left:=B u.right.parent:=u.parent – B.parent:=v u.right:=u.parent – u.right:=v u.parent:=u.right.parent – u.parent:=v.parent u.right.parent:=u – v.parent:=u u.parent.child=u – w.child=u

(b)

We did 6 assignments and since 6 pointers need to be updated, this is the minimum number of assignments. NOTE: if you used 7 assignments, it is not bad.

6. (15 points)
Exercise 3.3 (a) and (b) in Lecture III. Rotations to transform any binary search tree into any other equivalent one. Recall that two BST's are equivalent if they store exactly the same set of keys.

SOLUTION:

(a) We apply induction on the number of nodes of the tree. **NOTE**: When we say induction on trees, usually we mean induction on the number of nodes or on the height (depending on the problem one might work, or the other.)

The basis step is when both trees have just a node (the root) and in this case they are equal so nothing has to be done.

The induction step: assume the property is true for trees with less than n nodes and prove it for n nodes. For this problem the trick was to realize that once you make the trees have the same key at root, you can apply induction on the left and right subtree. So suppose we have 2 BSTs, which we denote by T_1,T_2.

Suppose we fix T_1, and transform T_2 into T_1. The first thing we do is make them have the same root. So let's try to see how we can do it. Suppose r_1 is the root of T_1. Since the trees have the same keys, then r_1 must be somewhere in T_2. Now, by a series of rotations we can bring it up to the root of T_2. Since our trees are binary search trees, i.e. keys in left subtree $<$ key at root $\leq$ keys in the right subtree, and since our trees have the same keys and the same key at root, it is clear that the left subtree in T_1 and the left subtree in T_2 have the same keys and same thing holds for the right subtrees.

Since the left and right subtrees in both trees have less than n nodes, then we can apply the induction hypothesis. So the left subtree in T_2 can be transformed into the left subtree of T_1 and the right subtree into T_2 can be transformed in the right subtree of T_1, so we showed that T_2 can be transformed in T_1. We're done!

(b) We can write an approximate recurrence for the number of rotations. We need cn rotations for making both trees have the same root. And so we can write: $T(n) \leq max(T(n_1)+T(n-n_1-1))+cn$ (we need $T(n_1)$,$T(n-n_1-1)$ rotations for the left and right subtree). We prove by induction that $T(n) \leq an^2$. We have

$$T(n) \leq a(n_1^2 + (n-n_1-1)^2) + cn \leq a(n-1)^2 + cn < an^2$$

Observe that we have to choose a large such that $T(1) < a$ for the basis case of induction and a much larger than c for the above inequality to hold!

7. (30 Points)

(a) Insert the keys $1, 2, 3, \ldots, 14, 15$ into an AVL Tree, in this order. Please draw your tree at the end of each insertion.
(b) Suppose you continue inserting in this manner indefinitely. Prove that when you insert key $2^n - 1$, you will have a complete binary tree (i.e., every leaf is at the same level).
(c) Consider the AVL tree T_{12} after you have inserted keys $1, 2, \ldots, 11, 12$ as in (a). What is the

minimum number of keys you need to delete from T_{12} before you cause a double rotation to happen? Please specify the order of keys you need to delete, and show the tree after each deletion. The last deletion must cause a double-rotation.

SOLUTION:

(a) See Figure 1, Figure 2 and Figure 3.

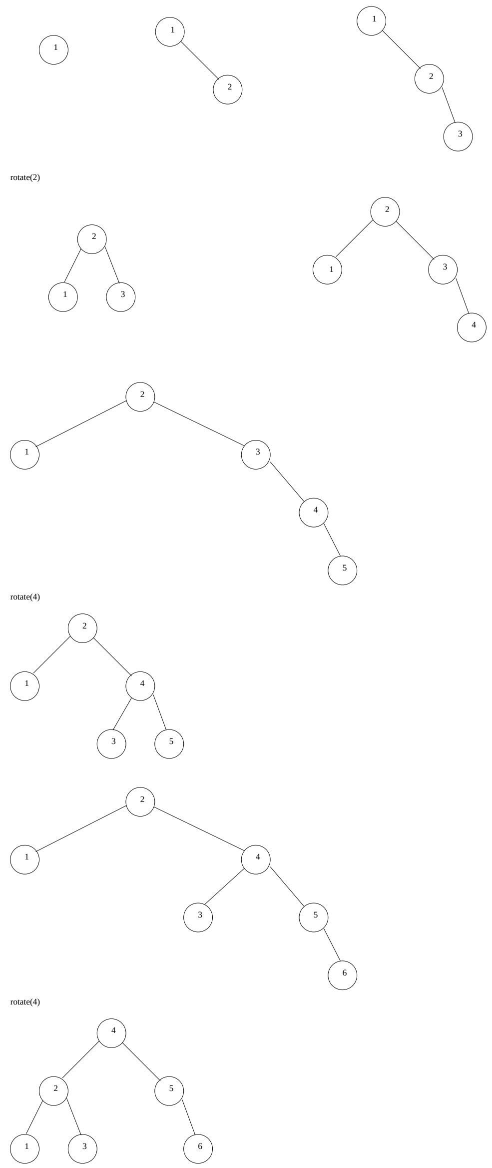

Figure 1: First six figures

(b) The proof is by induction on n, assuming that we have inserted $2^n - 1$ keys into the tree. We claim that the result is a complete binary tree of height $n - 1$.

The basis case is when $n = 1$ and $n = 2$. This is clear.

Inductively: suppose that the result is true for $2^n - 1$ ($n \geq 2$). We must prove the result for $2^{n+1} - 1$. Consider what happens when we insert the next 2^n keys into the complete binary tree with $2^n - 1$ keys: view this tree as a root v with two subtrees, A and B, where A and B are complete binary trees of heights $n - 2$.

[FIG A – fill in your own – the figures in this part are the ones I used in class, Monday Mar 10th, during midterm review]

- After the next 2^{n-1} keys are inserted, by induction, the tree B will be transformed into a complete binary tree of height $n-1$. Let this subtree be denote B'. Now, view B' as a root u with two subtrees C and D, each complete binary tree with height $n-2$. Assume $A < C < D$ (i.e., the keys in A are less than the keys in C, etc). At this point, the tree size is $2^n - 1 + 2^{n-1}$.

  [FIG B – fill in your own]

- When we insert the next item, this will transform D into D' with height $n-1$. That means the root v of the entire tree (which now has $2^n + 2^{n-1}$ keys) is unbalanced. This will force a rotation at u. Now, u is the root, and v the left child.

  [FIG C – fill in your own]

- Now, if we continue with inserting the next $2^{n-1} - 1$ keys, by induction, the tree D' will become a complete balanced tree of height $n-1$. Call this tree E.
- At this point, the tree is rooted at u, leftchild is v. The left subtree of v is A and right subtree of v is C. The right subtree of u is E. This is a complete binary tree of height n, with $2^{n+1} - 1$ keys.

(c) The solution is illustrated in Figure 4.

POSTSCRIPT: the solution in the figure is wrong. It deleted the keys 1, 3, 12, 9 and 11 to get a double rotation. But it suffices to delete the four keys: 1, 3, 7, 2. Thanks to Yanjun for pointing this out.

8. (30 points)
   Exercise 4.9 (Lect.III), parts (a) and (b) only. For the algorithm in (b), describe it in the style that we use to explain the insertion algorithm. Draw pictures to help explain your ideas.

   SOLUTION:

   a. To get the formula we do a similar reasoning as in the normal AVL trees. The smallest AVL tree of height h is one that has the left subtree of height $h-1$ and the right subtree of height $h-3$ because now we allow the balance factor to be 2. So the formula we get is:

   $$\mu(h) = \mu(h-1) + \mu(h-3) + 1$$

   Now to prove that such trees are balanced, all we need to do is to prove that height$(h) = O(\log n)$. If you look well at the recurrence, you see that you can deduce:

   $$\mu(h) = \mu(h-1) + \mu(h-3) + 1 \geq 2\mu(h-3) \geq 4\mu(h-6) \geq \ldots \geq 2^{\frac{h-3}{3}}\mu(1) \geq c2^{\frac{h}{3}}$$

   So for an AVL tree of height h we have $n \geq \mu(h) \geq c2^{\frac{h}{3}}$. So it follows that $h = O(\log(n))$.

   b. We need to do two things: first insert the node as in a BST and then we need to rebalance the tree. We'll explain in the following how to do the rebalance step. The possible unbalance is 3 or -2. The first happens when before inserting the balance was 2 and we inserted in the left subtree. The second case happens when we insert in the right subtree. The unbalance can occur only on the path from the inserted node to the root, so we start from the inserted node and balance when necessary on the way up to the root.

   Case 1:
   Unbalance 3 at node u. We have inserted a node on the left, making the height of the left subtree to increase by 1. Call the root of the left subtree v and its height h. Note that the right subtree has height $h-3$. Also call h_R and h_L the heights of the left and right subtrees of v. We have $\max(h_L, h_R) = h-1$ and we also know that v is balanced, so $h_L - h_R \in \{-1, 0, 1, 2\}$. So there are 4 cases:

1.
$h_L = h-1$, $h_R = h-3$. By rotate(v) we rebalance the tree. The left child of v has height $h_L = h-1$, the right child has height $1+\max(h_R, h-3) = h-2$. The left child of u has height $h_L = h-3$ and the right child has height $h-3$. So we see that the new tree is balanced. 2.
$h_L = h-1$,$h_R = h-2$ Exactly the same argument as above works. (rotate(v)) 3.
$h_L = h-1$,$h_R = h-1$ The same rotation as in previous cases. 4.
$h_L = h-2$ and $h_R = h-1$. We need to do a double rotation in this case. We expand the right subtree of v. Call its root w and D_L and D_R the height of the left and right subtrees of w. We have the relations:
$\max(D_L, D_R) = h-2$ and $D_L - D_R \in \{-1, 0, 1, 2\}$. The double rotation of w will make the tree balanced: at w, the left height is $1+\max(h-2, D_L) = h-1$ and the right height $(1+\max(D_R, h-3)) \in \{h-2, h-1\}$. At v left is $h-2$, right is $D_L \in \{h-3, h-2\}$, so v is balanced. At u left is $D_R \in \{h-4, h-3, h-2\}$ and right is $h-3$. So again balanced. Case 2:
Analogous.

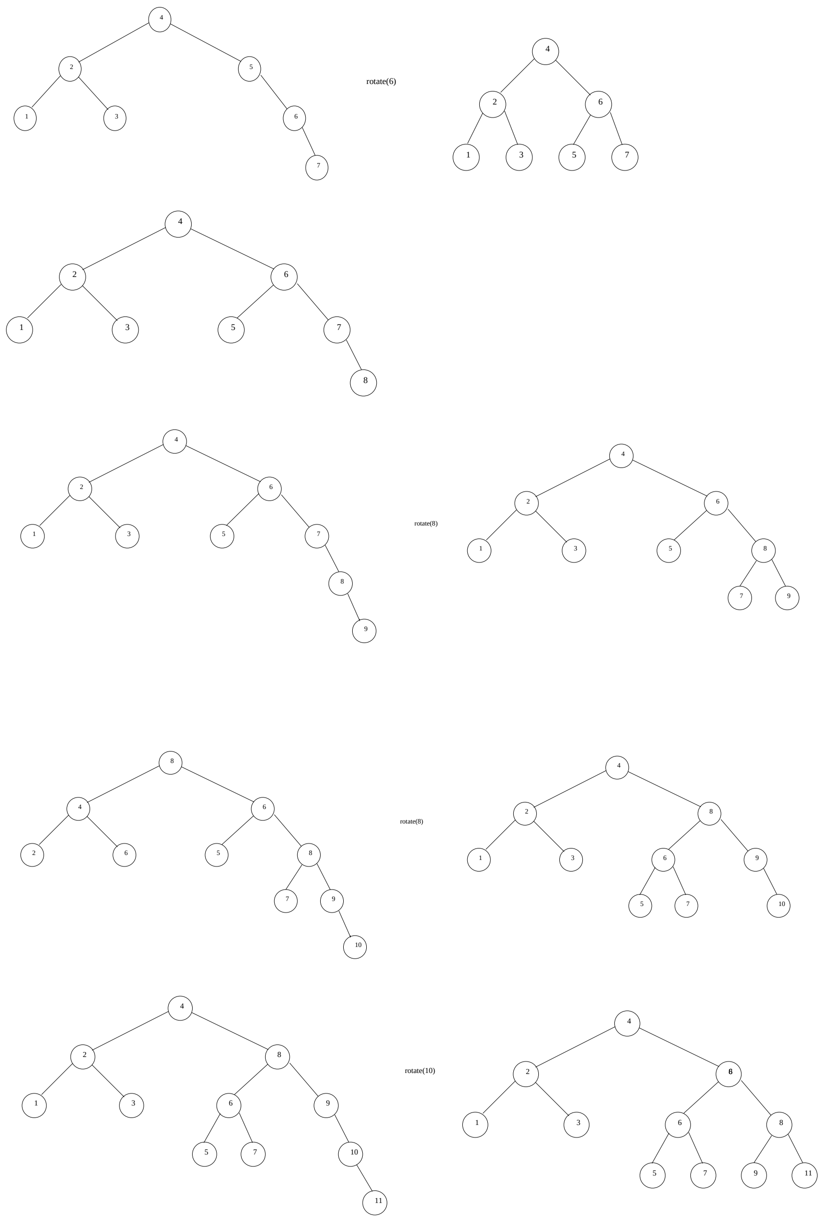

Figure 2: Next five figures

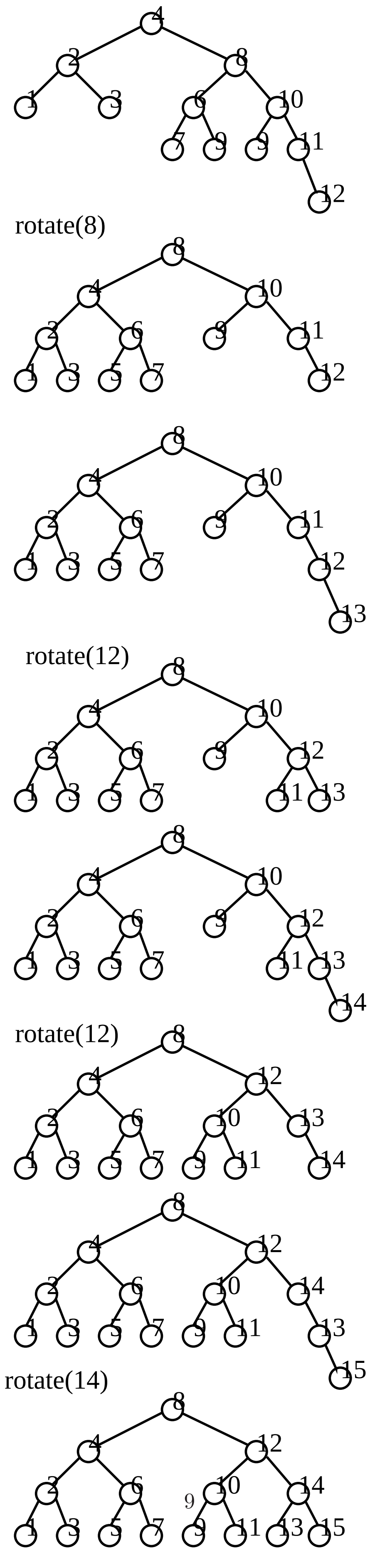

Figure 3: Last four figures

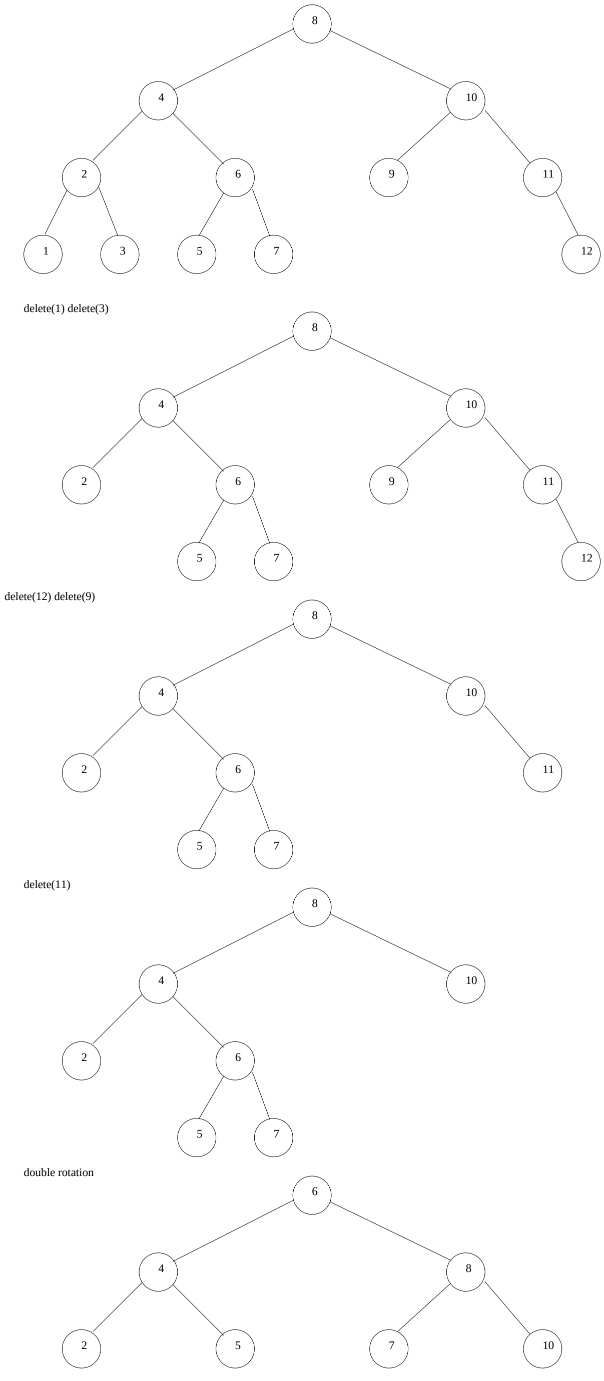

Figure 4: Double Rotation Question