MIDTERM – With Solutions
Fundamental Algorithms, Fall 2005, Professor Yap
Oct 20, 2005

INSTRUCTIONS:

0. This is a closed book exam, with one 8"x11" (2-sided) cheat sheet.
1. Please answer ALL questions
2. Please write clearly. Use complete sentences. This CAN AFFECT your grade. If necessary, consider printing.
3. Write answers ONLY on the front side of each booklet page. Use the reverse side for scratch and to show your working.
4. Read carefully, think deeply, write sparsely.
5. Begin each question in its own page.

SHORT QUESTIONS (6 Points Each)

No proofs required for this part. Just state tight upper and lower bounds for the following sums or recurrence functions.
(a) $\sum_{i=1}^{n}(i!)$
(b) $\sum_{i=1}^{n} \frac{i^2}{2^i}$
(c) $T(n) = 3T(n/10) + \sqrt{n}/\log n$.
(d) $T(n) = 10T(n/3) + \sqrt{n}\log n$.
(e) Order these 6 functions in increasing Θ-order:

$$\frac{n}{\lg n}, \quad n\lg\lg n, \quad 2^{\lg n}, \quad n!, \quad n^{\lg n}, \quad 2^n$$

SOLUTION:
(a) $\Theta(n!)$.
REMARK: Justification for is by our summation rule for exponentially large sums. HOWEVER, YOU DID NOT HAVE TO SAY THIS. Our instructions said to JUST STATE the bound. Certainly no proofs. Put any derivation on the work sheet if you want, but NOT on the answer part (but some students write several pages for one subpart!).
Students somehow forgot the summation rules completely, and some actually think this is a polynomial sum!
(b) $\Theta(1)$.
REMARK: Justification is by our summation rule for exponentially small sums. Some students say that this is $O(n^3/2^n)$. Why is this wrong?
(c) $\Theta(\sqrt{n}/\log n)$.
REMARK: Justification is by Master Theorem. You need to see that $\log_{10} 3 < \log_9 3 = 1/2$.
(d) $\Theta(n^c)$ where $c = \log_3(10) > 2$.
REMARK: Again by Master Theorem. You need to see that $\log_3 10 > \log_3 9 > 2$.
(e)

$$\frac{n}{\lg n}, \quad 2^{\lg n}, \quad n\lg\lg n, \quad n^{\lg n}, \quad 2^n, \quad n!$$

REMARK: Many people did not recogize $2^{\lg n} = n$. For some reason, they place $2^{\lg n}$ right after $n\lg\lg n$. Some even think that $n\lg\lg n \prec n/\lg n$ (where could this have come from?).

QUESTION 1, AVL trees (20 Points)

Consider the AVL tree in Figure 1.
(a) Find one key that we can delete so that the rebalancing phase requires two separate rebalancing acts (either a single- or double-rotation)? Note that a double-rotation counts as one, not two, rebalancing act. [EXTRA CREDIT IF YOU CAN TELL US ALL SUCH KEYS]

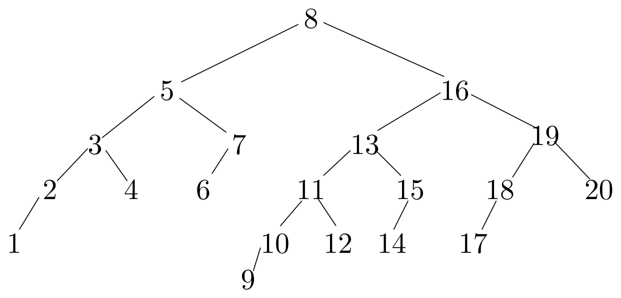

Figure 1: An AVL Tree for deletion

(b) Please delete this key, and draw the AVL tree after each of the rebalancing act. [We want to see two AVL trees]
(c) Recall the funny 'AVL' tree in the homework. Show that there are no 'AVL' trees with an even number of nodes. HINT: a node is **full** if it has exactly two children.

SOLUTION:

(a) We can delete any one of the keys 4, 5, 6, 7 or 20. Even keys 3 and 9 might work, but that depends on the convention for replacing a deleted key with 2 children by its predecessor, not predecessor.

(b) Suppose we delete 6. To rebalance, we need to do rotate 3 and double rotate 13, as in Figure 2.

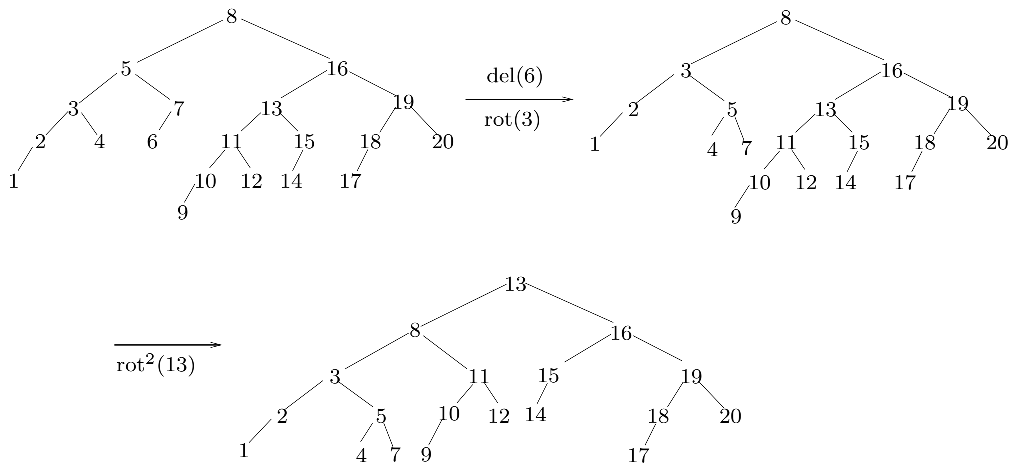

Figure 2: Rebalancing AVL Tree after deleting key 6

(c) A **full binary tree** is one where all internal nodes are full. All 'AVL' trees are full binary trees. binary tree have an odd number of nodes. Other than the root level, every other level must have an even number of nodes in a full binary tree. To see this, note that every nodes in a non-root level can be paired up into siblings. Thus there is only one level with odd number of nodes, and thus the overall number is odd.

REMARK: For some reason, several of you start to invoke the function $\mu'(h)$ of homework. This is the minimum size of an 'AVL' tree of height h, but it has nothing to do with our problem.

QUESTION 2, Program correctness (30 Points)

Write a detailed, complete and correct pseudo-code for checking whether a binary tree with keys at each node is a binary search tree (BST). Assume each binary node u has three fields ($u.Key, u.\mathtt{left}, u.\mathtt{right}$) that satisfy the inequality

$$u.\mathtt{left}.Key \le u.Key < u.\mathtt{right}.Key$$

THIS is the promised problem in which I will grade for your ability to write clean (beautiful) code. There should be enough details to trivially turn your pseudo-code into a compilable program in, say Java. Use indentation to indicate block structure. It should be a pleasure to read your solution, not a pain. Checklist:
(a) Before giving the code, give a brief description of your approach (2 or 3 lines suffice).
(b) Specify the input and output of your program.
(c) Explain any variables and data structures.
(d) Initialize your variables and cover the base cases.
(e) Add comments as appropriate.

SOLUTION:

Write T_u for the binary tree rooted at u, and $\max\{T_u\}$ for the maximum key in T_u (similarly for $\min\{T_u\}$. Then T_u is a BST iff $u = \mathsf{nil}$ or:
(1) $T_{u.\mathtt{left}}$ is a BST
(2) $T_{u.\mathtt{right}}$ is a BST
(3) $\max\{T_u.\mathtt{left}\} \le u.Key < \min\{T_u.\mathtt{right}\}$.

The problem with (3) is that we need to make the left inequality true in case $u.\mathtt{left} = \mathsf{nil}$, and similarly for the right inequality. To achieve this, we define $\max\{T_u.\mathtt{left}\} = -\infty$ when $u.\mathtt{left} = \mathsf{nil}$ and define $\min\{T_u.\mathtt{right}\} = -\infty$ when $u.\mathtt{right} = \mathsf{nil}$.

We use postorder traversal. Let $POST(u)$ take a node u as argument, and returns a pair $[minkey, maxkey]$ of numbers. When T_u is a BST, $minkey$ is the smallest key in the subtree T_u and $maxkey$ is the largest key in T_u. There are two special cases: in case $u = \mathsf{nil}$, we return $[minkey, maxkey] = [+\infty, -\infty]$. In case T_u is not a BST, we return $[minkey, maxkey] = [-\infty, +\infty]$. Then our algorithm is:

```
POST(u)
▷ Base Case:
    if ($u = \mathsf{nil}$)
        return($[+\infty, -\infty]$)
▷ Recursive Case:
    $[Lmin, Lmax] \leftarrow POST(u.\mathtt{left})$
    $[Rmin, Rmax] \leftarrow POST(u.\mathtt{right})$
▷ If not BST:
    if (($Lmax > u.Key$) or ($Rmin \le u.Key$))
        else return($[-\infty, +\infty]$)
    else
▷ If BST:
        return($[\min\{Lmin, u.Key\}, \max\{Rmax, u.Key\}]$
                ◁ This takes care of the cases $Lmin = +\infty$ and $Rmax = -\infty$
```

REMARK:
(1) Many of you committed the standard mistake of thinking that a BST only has to satisfy the relationship $u.\mathtt{left}.Key \le u.Key < u.\mathtt{right}.Key$ for all node u.
(2) Second, I said in class that anything about binary trees ought to be done recursively by induction. So a post order traversal is the best solution. HOWEVER, for this problem the post order must return some number (not just a Boolean value) in order for this to work.
(3) As for the base case, it seems always simplest to use $u = \mathsf{nil}$ as the base case.

QUESTION 3, Recurrences (20 Points)

Suppose $T(n) = n + 3T(n/2) + 2T(n/3)$. Joe claims that $T(n) = O(n)$, Jane claims that $T(n) = O(n^2)$, John claims that $T(n) = O(n^3)$. Who is closest to the truth? Prove it.

SOLUTION:

Answer: Jane.

We can see at once that Joe is wrong: the related function $T'(n) = n + 3T'(n/2)$ is already nonlinear (by Master Theorem). Clearly, $T(n) \geq T'(n)$.

We can check by real induction that Jane is correct. To show that $T(n) \leq Kn^2$ (ev.), we can check the critical constant $k_0 = \frac{3}{2^2} + \frac{2}{3^2}$. Since $k_0 < 1$, our induction will succeed.

Now, if Jane is right, then John is also correct, but of course. But he is far from the truth.

REMARKS: Some tried to use "Master theorem" on the recurrence $T(n) = n + 3T(n/2) + 2T(n/3)$. There is no such theorem! Certainly, it is impossible to use rote method (try it if you don't see why). As for upper bound, you can use the fact that the function

$$T''(n) = n + 5T''(n/2)$$

is an upper bound on $T(n)$. The Master theorem tells us that $T''(n) = \Theta(n^{\lg 5})$ where $\lg 5 < 3$. But is this closer to Jane or to John? Well, a pocket calculator would tell you that $\lg 5 < 2.322$, so you might say that this shows Jane is closer to the truth. But my questions normally do not need a pocket calculator unless I tell you so.

QUESTION 4, (a,b)-Trees (5 Points each)

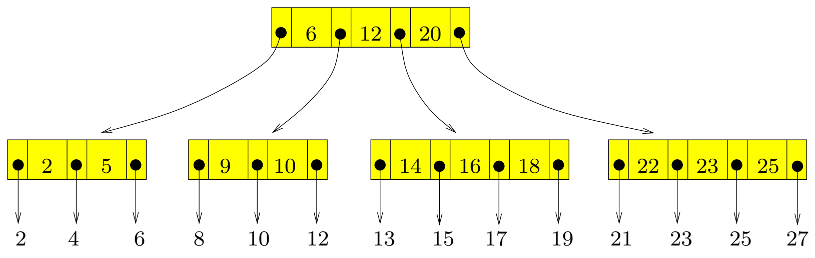

Figure 3: A (2,4)-tree or a (3,4)-tree.

Consider the (a, b, c)-tree in Figure 3. We will view it as a $(2, 4, 1)$-tree or a $(3, 4, 2)$-tree. Note that $a' = b' = 1$. Make these assumptions below:

• When splitting a node with 5 children into two, the left node is to have 2 children, and right node have 3 children.

• When looking for one sibling, we first try the left-sibling. If there is no left-sibling, we try the right-sibling.

• When looking for two siblings, we first try the left-sibling then the right-sibling. Of course, sometimes we have to look further.

(a) Suppose we insert the key 14 into Figure 3, viewed as a $(2, 4, 1)$-tree. Draw the resulting tree. When drawing trees, you need not draw the contents of nodes that are not modified.

(b) Same as part(a), but viewed as a $(3, 4, 2)$-tree.

(c) Suppose we delete the key 4 from Figure 3, viewed as a $(2, 4, 1)$-tree. Draw the resulting tree.

(d) Same as part(c), but viewed as a $(3, 4, 2)$-tree.

(e) Find the smallest choices of the parameters a, b such that we could have an $(a, b, 3)$-tree. Explain.

SOLUTION:

(a) See Figure 4. After insert,ion we split the overfull node. Now parent (root) is overfull, and is split.

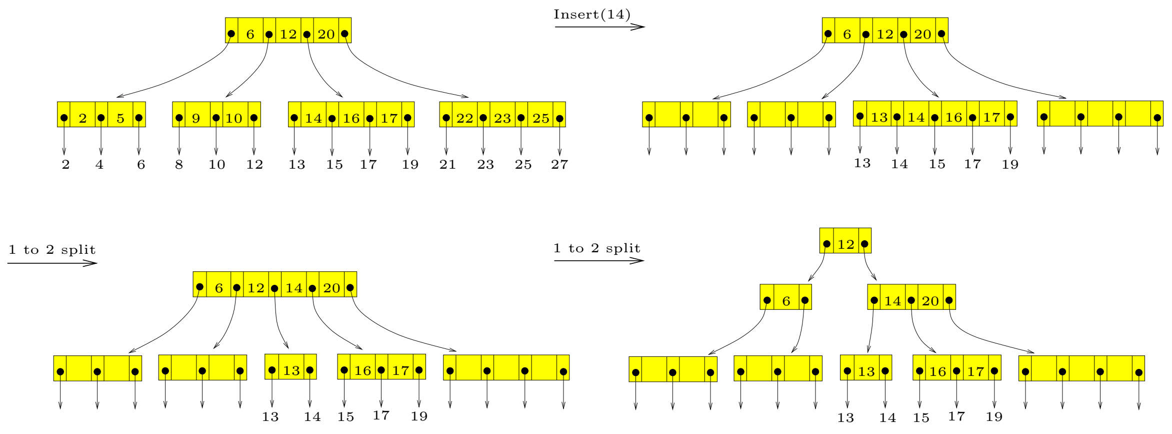

Figure 4: Inserting 14 into $(2, 4, 1)$-tree.

REMARK: In (a, b, c)-trees, the c parameters tells you that we want to do a $c \to c + 1$ generalized split, and do a $c + 1 \to c$ generalized merge. Since $c = 1$, you do a $1 \to 2$ split. DO NOT try to donate any child to your sibling! Some students do not remember that the keys in internal nodes are just used for searching, and do NOT represent items.

(b) See Figure 5. After insertion, the overfull node donates a child to its left sibling.

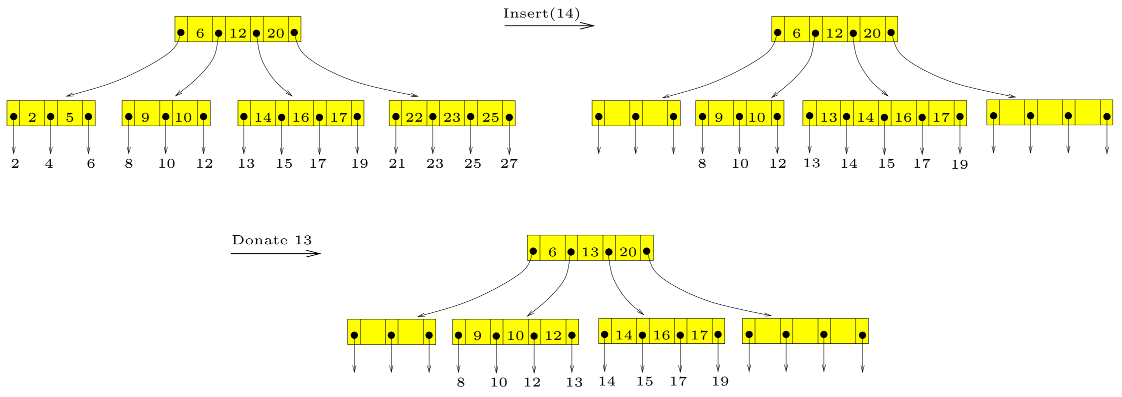

Figure 5: Inserting 14 into $(3, 4, 2)$-tree.

REMARK:Note that a key from the parent must be transferred to the left sibling, and we must also move a key to the parent: you must get these keys right!

(c) See Figure 6. Everything is fine after the deletion. But note that a separator key in the parent must be deleted (here we deleted key 5, but we could delete key 2 instead).

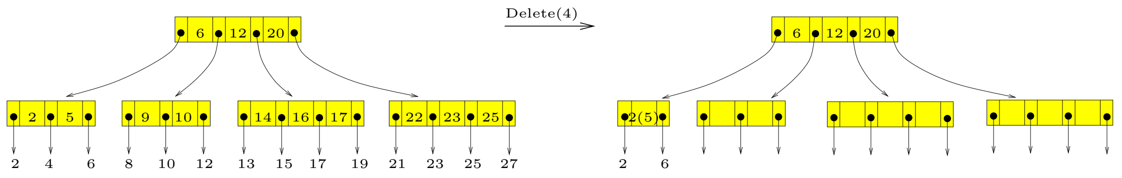

Figure 6: Deleting 4 from $(2, 4, 1)$-tree.

(d) See Figure 7. After deletion, the current node u is underfull. We try to borrow from the right sibling, but failed. But the right sibling of the right sibling could give up one child. So we first merge u with the 2 siblings to its right (call this a 3-to-1 merge). This requires bringing some keys from the parent of u into the supernode. The supernode has 9 children, which we can split into 3 nodes, each with 3 children (call this a 1-3 split).

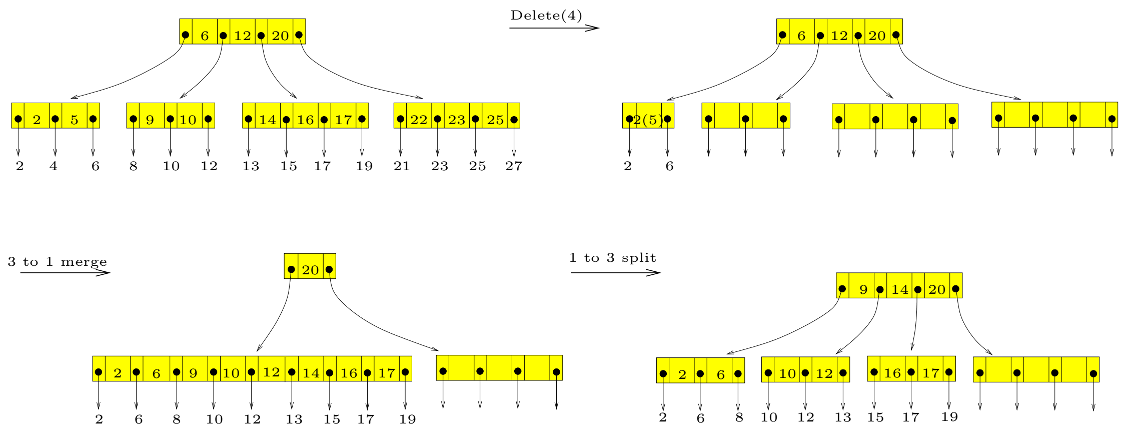

Figure 7: Deleting 4 from $(3, 4, 2)$-tree.

(e) We need to satisfy

$$c + 1 \leq a \leq \frac{cb + 1}{c + 1}.$$

Since $c = 3$, a is at least 4. Also, $b > a$ (by a, b-tree inequalities). So b is at least 5. We may verify that $(a, b) = (4, 5)$ is the smallest values that satisfy these inequalities.