MIDTERM with SOLUTION

1. PLEASE READ INSTRUCTIONS CAREFULLY.
2. Answer all questions.
3. Use ONLY THE FRONT SIDE OF EACH PAGE for your answers.
4. Use the REVERSE SIDE of each page for scratch work.
5. This is a closed book exam, but you may refer to one $8'' \times 11''$ sheet of prepared notes.

GENERAL COMMENTS ABOUT YOUR ANSWERS:

- *Remember that grades for this course is curved (normalized) and not based on absolute points.*
- *Question 1 asked for "brief justifications" but many simply wrote down an answer and left it at that! Question 2 asked for intermediate trees after EACH insertion/deletion/rotation, but some simply skip many crucial intermediate steps. Please realize that you are not entitled to any points when you ignore such explicit instructions (but we have graded more generously than this).*
- *Each question asks for something specific: Question 1(iii) asked for a set, but most answers never even mention a set. Question 1(iv) asked for a solution a recurrence. You must give us what we asked for. Otherwise, you are not answering the question.*

1. **(10 points each part) Short Questions**
No credit UNLESS you show your computation or give a brief justification (less than five lines). I will ignore long answers!
(i) Give upper and lower bounds for $M(2, 10)$. Recall that $M(m, n)$ is the worst case number of comparisons for merging two sorted lists of sizes m and n, respectively.

SOLUTION: The best answer is $7 \leq M(2, 10) \leq 8$.

Lower Bound: The information-theoretic lower bound shows that $M(2, 10) \geq \lceil \lg \binom{12}{2} \rceil = \lceil \lg 66 \rceil = 7$. A weaker lower bound uses the argument we discussed in class and notes:

$$M(m, n) \geq 2 \min\{m, n\} - \delta(m, n)$$

where $\delta(m, n) = 1$ if $m = n$, and otherwise $\delta(m, n) = 0$. This gives $M(2, 10) \geq 4$.

Upper bound: recall that the standard merging algorithm gives $M(m, n) \leq m + n - 1$. Hence $M(2, 10) \leq 11$. But using binary search ideas, we can do better when m is much smaller than n (as in this case). We can first insert the first of the 2-elements list into the list of size n. This takes $\leq \lceil \lg(1 + n) \rceil$ comparisons. The second element can also be inserted in the same way, and again takes $\leq \lceil \lg(1 + n) \rceil$ comparisons. Hence $M(2, n) \leq 2 \lceil \lg n + 1 \rceil$ or $M(2, 10) \leq 2 \lceil \lg 11 \rceil = 8$.

REMARKS: You get most of the credit if you show $4 \leq M(2, 10) \leq 11$. Students are still confused about where upper and lower bounds come from. The arguments for these two cases are very different! Upper bounds are usually easier to understand – it comes from an algorithm.

(ii) Suppose

$$T_0(n) = 18T_0(n/6) + n^{1.5}$$

and

$$T_1(n) = 32T_1(n/8) + n^{1.5}.$$

Which is the correct relation: $T_0(n) = \Omega(T_1(n))$ or $T_0(n) = O(T_1(n))$? YOUR JUSTIFICATION MUST NOT USE CALCULATOR VALUES. HINT: Use inequalities like $\log_8(x) < \log_6(x)$ for $x > 1$ and $\log_6(2) < 1/2$.

SOLUTION: The watershed constant for the two recurrences are $\alpha_0 = \log_6(18) = \log_6(36/2) = 2 - \log_6(2)$ and $\alpha_1 = \log_8(32) = \log_6(64/2) = 2 - \log_8(2)$.

For $i = 0$ or 1, we have $\alpha_i > 1.5$ because $1 > \log_6(4)$ implies $1/2 > \log_6(2) > \log_8(2)$. Hence α_1 is clearly larger than α_0, and we are in CASE (-1) of the Master Theorem. Thus $T_i(n) = \Theta(n^{\alpha_i}$, and so $T_0(n) = O(T_1(n))$.

REMARK: you should know basic properties of log to do this problem properly.

(iii) Consider the AVL Tree in Figure 1

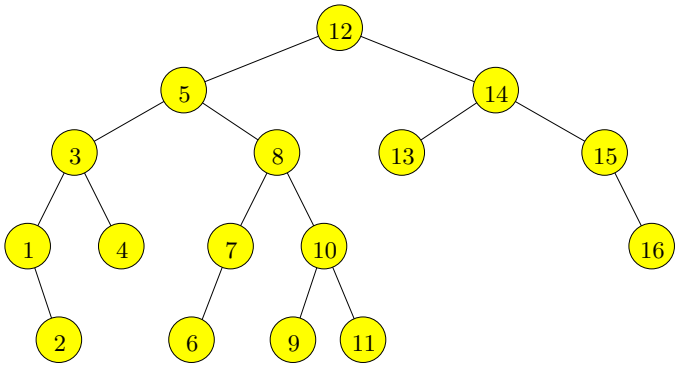

Figure 1: AVL Tree on 16 keys

Insert a new key 9.5 into the tree, and show the result after each single or double rotation.

SOLUTION: After you insert 9.5, you are no longer AVL-balanced at the root (key 12).

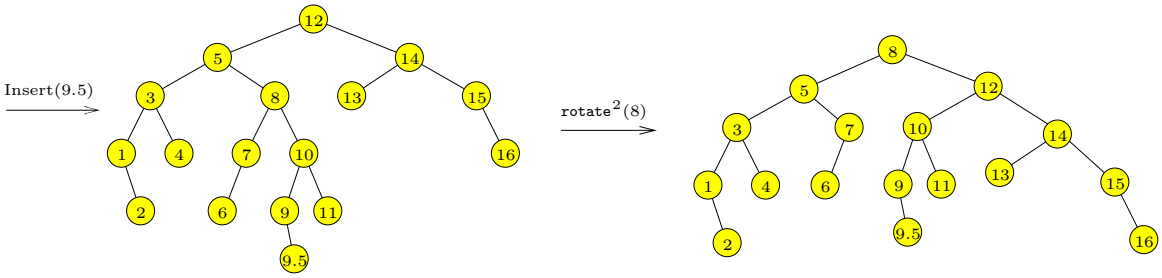

Figure 2: Inserting 9.5

A double rotation at key 8 will restore balance.

(iv) Again starting from the AVL Tree in Figure 1, delete the key 13, and show the result after each single or double rotation.

SOLUTION: After deleting 13, the node 14 is unbalanced. This is restored by a single rotation at 15. Now, the root 12 is unbalanced. Another single rotation at 5 will restore balance.

REMARKS: some of you do a double rotation at node 8. This will also give you a balanced tree, but obviously, it is more efficient to do a single rotation (that is the "official" algorithm for this case. No points taken off for this error.

(v) Give the inorder, and preorder postorder listing of the AVL tree in Figure 1.

SOLUTION: Almost everyone got this right:

INORDER: $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16$.

PRE-ORDER: $12, 5, 3, 1, 2, 4, 8, 7, 6, 10, 9, 11, 14, 13, 15, 16$.

POST-ORDER: $2, 1, 4, 3, 6, 7, 9, 11, 10, 8, 5, 13, 16, 15, 14, 12$.

(vi) Consider the $(3, 4)$-search tree in Figure 3, subject to the 2/3-splitting and merging algorithms. Draw the resulting tree after we insert a new key 24 into it.

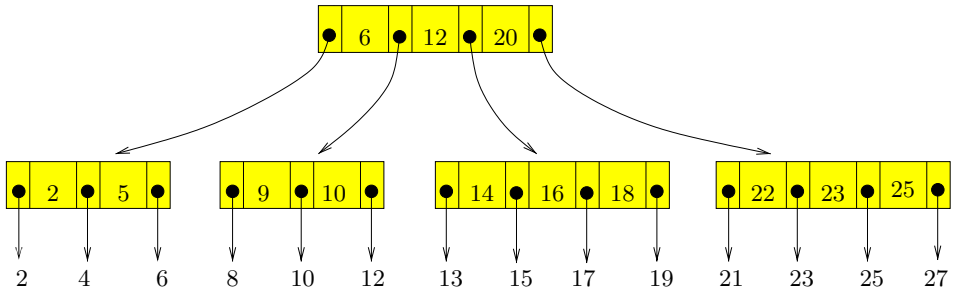

Figure 3: $(3, 4)$-search tree on 14 keys

SOLUTION: First you insert 24 into the leaf, and produce an overfull node whose keys are $u = (22, 23, 24, 25)$. We first try to donate keys to an adjacent sibling. But this sibling $v = (14, 16, 18)$ is already full. Hence you must merge these nodes u and v, and then split into three new nodes:

The merged node is $(14, 16, 18, \underline{20}, 22, 23, 24, 25)$. NOTE that we need to bring the key 20 from the parent w of u and v! Now you split w into three nodes, $(14, 16)$, $(20, 22)$ and $(24, 25)$. The keys 18 and 23 are sent to the parent w!

Now the parent w is $(6, 12, 18, 23)$. Since w is overfull, you try donate a key from w to a sibling. Since w is the root, it has no siblings, and you can only split w, and create a new root. You can split w into (6) and $(18, 23)$, or into $(6, 12)$ and (23) (we really don't care which one you pick). In first case, the new root is (12), and the second case the new root is (18).

REMARKS: Many got this question wrong. The main problem that you do not have a general algorithm (the official 2/3 algorithm) in mind, but perform an adhoc donation or borrowing of keys.

(vii) The function $Verify(u)$ is supposed return True iff the binary tree rooted at node u is a binary search tree:

```
VERIFY(Node u)
    if (u = Nil) return(True)
    if ((u.Left ≠ Nil) and (u.Key ≤ u.Left.Key)) return(False)
    if ((u.Right ≠ Nil) and (u.Key ≥ u.Right.Key)) return(False)
    return(VERIFY(u.Left)∧VERIFY(u.Right))
```

Please EITHER argue for it's correctness, OR give a counter-example showing it is wrong. For simplicity, assume we require distinct keys in the binary search tree.

SOLUTION: There is a potential ambiguity in my phrasing of the question, so I grade this question generously.

Students also do not seem to understand the concept of counter example – if you think the function is wrong, you are NOT supposed to argue why it goes wrong. You are to give an explicit example. The simplest example only require three nodes: the root has key 2, its left child u has key 1, and the right child of u has key 3. Then our algorithm will claim that this tree is a binary search tree. Of course, it is wrong.

REMARKS: The idea is to test your understanding of the definition of a binary search tree. The **standard error** in this area is to belief that it is sufficient to compare the keys at each node u with the keys at the children of u only (this is what our algorithm actually does). This is necessary, but not sufficient.

Many students misinterpreted my question to require that u cannot be the child of another node. But what we want to check is that the SUBTREE rooted at u is a binary search tree!

Some students get confused by the fact that we return True for a Nil node. There is nothing wrong here – the empty tree can be regarded as a valid binary search tree.

In the end, I gave points if your answer shows that you understood the nature of the recursive algorithm! That was not the original intention of the question, of course.

2. **(15+20 points) Recurrences**

(i) Let

$$s(n) = \sum_{i=1}^{n} \frac{\lg i}{i}$$

Prove that $s(n) = \Theta(\lg^2 n)$. HINT: For the lower bound, you can use real induction, and the fact that for $n \geq 2$, there are constants $0 < c < C$ such that

$$\lg(n) - (C/n) < \lg(n-1) < \lg(n) - (c/n)$$

(ii) Using the transformation methods to solve the following recurrence:

$$T(n) = 2T(n/2) + n\frac{\lg\lg n}{\lg n}.$$

SOLUTION: (i) The upper bound is easy: $s(n) \leq \lg n \sum_{i=1}^{n}(1/i) = O(\lg^2 n)$. For lower bound, suppose we want to show $s(n) \geq K \lg^2 n$ (ev.. n).

$$\begin{aligned} s(n) &= s(n-1) + \lg(n)/n \\ &\geq K\lg^2(n-1) + \frac{\lg n}{n} \\ &\geq K\left(\lg(n) - (C/n)\right)^2 + \frac{\lg n}{n} \\ &> K\lg^2(n) - 2KC\frac{\lg n}{n} + \frac{\lg n}{n} \\ &\geq K\lg^2 n \end{aligned}$$

provided $K \leq 1/(2C)$.

REMARK: Another way to do the lower bound is to split the summation into two sums: sum A goes from $i = 1$ to $\lceil\sqrt{n}\rceil$, and sum B takes the remaining terms. Clearly, $s(n) \geq B$. We can easily show that $B \geq K\lg^2 n$ for some K.

Few got this problem correct.

(ii) By domain transformation $t(N) = T(2^N)$, we transform the recurrence into

$$t(N) = 2t(N-1) + 2^N \frac{\lg N}{N}.$$

By range transformation $s(N) = t(N)/2^N$, we transform it further into

$$s(N) = s(N-1) + \frac{\lg N}{N}.$$

The solution is $s(N) = \sum_{i=1}^{N} \frac{\lg i}{i}$, which we show in part (i) is $\lg^N$. Plugging back into the original equation, we obtain

$$T(n) = n(\lg \lg n)^2.$$

REMARK: these transformations are completely standard, so there is no reason not to get it correct. Surprisingly, some people who derived $s(N)$ correctly did not realize that $s(N)$ can be summed using the answer from part (i).

3. **(5+20 points) Relaxed AVL trees**
In homework, we derived the algorithm for the insertion algorithm for a "relaxed" AVL tree in which the balance of each node satisfies $|balance(u)| \leq 2$.
(i) Is there any reason why you might restrict relaxed AVL trees so that the balance of each node is a value in $\{-1, 0, 1, 2\}$ rather than $\{-2, -1, 0, 1, 2\}$?
(ii) Derive the deletion algorithm for relaxed AVL trees.

SOLUTION:

As usual, we delete from the tree using a standard BST deletion algorithm and then trace the root path from deleted node to look for unbalanced nodes.

If u is unbalanced, there are 3 cases. Two of them are exactly the same as the cases for insertion. Recall from homework (on relaxed AVL tree insertion) and from lecture notes (on standard AVL tree deletion) that we have Cases (I.1) and (I.2). Exactly the same situation arise in this problem, although we call them Cases (D.1) and (D.2), respectively. So, I will be rather brief (you can refer to the same figures from the Lecture Notes).

For Case (D.1), we do a single rotation on a child of u. For Case (D.2), we do a double rotation at a grandchild of u. But unlike insertion, we cannot terminate after these rotations, but continue searching along the root path from u.

The new case is Case (D.3). In this case, we do a single rotation at a child of u but now we can terminate.

REMARK: I take off some points for not discussing the termination conditions.