COL 106 Autumn 2017 Minor 2

Welcome to minor 2. The exam is for 1 hour and 10 minutes. Pl
answering questions. Do not use a pencil.

Before starting the exam, close your eyes and take three deep br
the exam is not an accurate reflection of your understanding of t
you are relaxed, you will likely perform better.

Question Number	Maximum Marks
1	14
2	06
3	07
4	05
5	08

10.5

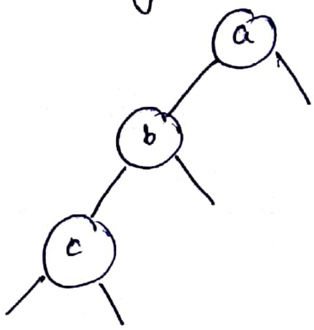
1. [14 points] Answer the following questions about AVL trees.

(a) [9 points] Recall that optimized implementations of AVL trees store *balance* (= height(left)-height(right)) in each node. They do not store the size of each subtree explicitly. Consider the case where a new AVLNode *s* has been inserted in the left subtree of left subtree of AVLNode *a* and balance values up to *a* have been updated in a bottom up pass. The algorithm finds *a* to be the first node in this pass where the updated balance is not between -1 and 1. Write the pseudo-code for the appropriate rotation to balance the AVL tree. You may assume these methods:

```
void setParent(AVLNode n);
void setLeftChild(AVLNode n);
void setRightChild(AVLNode n);
void setBalance(int b);
AVLNode getParent();
AVLNode getLeftChild();
AVLNode getRightChild();
int getBalance();
```

5.5

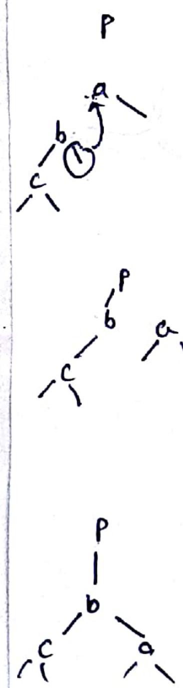
Since, insertion has been made into left of left subtree it is an outside case, and hence can be resolved by a single rotation.



b ← a. getleft Child()
c ← b. get leftChild()
p ← a. getParent()

a. set ~~Right~~ Left Child (b. get Right Child()) — transferred b's right child to a's left

~~b. set RightChi~~



if p. get leftchild() = a
p. set left child (b)
else
p. set Right child (b)
— 1.5 — promoted b

b. set parent (p)

b. set Right child (a)
a. set parent (b)
— demoted a

balance not updated

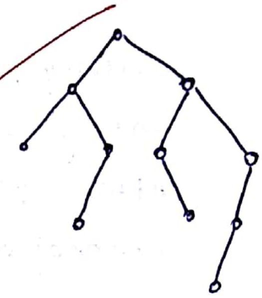
(b) [2.5 points] True or False: The AVL invariant implies that a tree's shortest and longest paths (from root to any leaf) differ in length by at most 1. Explain.

False.

AVL invariant only implies that for every node height of left and right subtree differ by atmost 1.

2.5

This places a constraint on max. path length in right and left subtree and not on all path lengths in general.



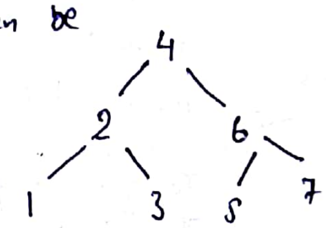
For eg. Tree drawn alongside is an AVL tree, though its shortest path is 2 units and longest 4 units.

(c) [2.5 points] What order should we insert the elements {1, 2, 3, 4, 5, 6, 7} into an empty AVL tree so that we don't have to perform any rotations on it? Explain.

There can be many possible orders, one of them can be

2.5

4 2 6 1 3 5 7



The one thing to be kept in mind is to insert median of the array first.
Then median of left half-array and right half-array.
Follow this recursively.

2. [6 points] Prove or disprove: Five elements cannot be sorted with at most seven comparisons in the comparison model.

Five elements can have 5! i.e. 120 possible combinations.

Initially, all the 5! combinations could be potential sorted arrangement.

After Ist comparison, we ~~know~~ narrow down to half the canbhations i.e. 60.

After IInd comperison, to 30.

Similarly after IIIrd, 15

IVth, 8 at max.

Vth, 4

VIth, 2

VIIth, 1

0 +2

Thus, in seven ~~step~~ comperisons, we can uniquely determine the permutation (in worst case) which is the sorted one

Hence the given statement is false.

To get a feel and intuition for answer
The binary decision tree made by the working of algorithm will ~~be~~ like an AVL tree and hence its height must be bounded $\log n! \sim O(n \log n)$. for 5, $n \log n$ ≈ 10. Hence, definitely the answer should be around 10.

3. [7 points] Answer questions about the procedure Stooge-sort

5

```
Input: array A[0..n-1] of n numbers
Output: A is sorted in increasing order.

If n = 2 and A[0] > A[1], then swap (A[0], A[1])
If n > 2 then {
        Stooge-sort(A[0..ceil(2n/3)]) // sort first two-thirds.
        Stooge-sort(A[floor(n/3)..n]) // sort last two-thirds.
        Stooge-sort(A[0..ceil(2n/3)]) // sort first two-thirds again.
}
```

(a) Let *T(n)* denote the worst case number of comparisons (A[0] > A[1]) made for an input array of *n* numbers. Give a recurrence relation for *T(n)*.

$$T(n) = T(2n/3) + T(2n/3) + T(2n/3) + O(1)$$
$$= 3\,T(2n/3) + 4O(1)$$

1.5

(b). Solve the recurrence – give a tight (Θ) asymptotic bound for *T(n)*. You are not allowed to use Master theorem (if you know it).

$$T(n) = 3\left(T(2n/3)\right) + \Theta(1)$$
$$= 3\left(3\,T\left(\tfrac{2}{3}\cdot\tfrac{2}{3}n\right) + \Theta(1)\right) + \Theta(1)$$
$$= 9\,T\left(\left(\tfrac{2}{3}\right)^2 n\right) + (3+1)\,\Theta(1)$$
$$= 27\left(T\left(\left(\tfrac{2}{3}\right)^3 n\right)\right) + (9+3+1)\,\Theta(1)$$
$$= 3^r\left(T\left(\left(\tfrac{2}{3}\right)^r n\right)\right) + \sum_{i=0}^{r-1} 3^i \cdot \Theta(1)$$
$$\cancel{= 3^{\log n/3}\,\Theta(1) + \sum_{i=0}^{\log n/3 - 1} 3^i} = \sum_{i=0}^{r} 3^i\,\Theta(1)$$

where $r \geq \log_{3/2}(n/2)$

$$= (3^{r+1}-1)\,\Theta(1)$$
$$= \left(3^{\log_{3/2}(n/2)+1} - 1\right)\Theta(1) = \Theta(n^2)$$

incorrect

2.5

we know at $n=2$
$T(n) = O(1)$
$\left(\tfrac{2}{3}\right)^r n \leq 2$
$\Rightarrow n \leq 2\cdot\left(\tfrac{3}{2}\right)^r$
$\log_{3/2} n \leq \log_{3/2} 2 + r$
$r \geq \log_{3/2}(n/2)$

(c) Is Stooge-sort a correct sorting algorithm? (no explanation needed)

Yes

1

(d) Complexity-wise is Stooge-sort a better algorithm than insertion sort? Explain.

4. [5 points] We are given a sorted array A of size $n=2^m-1$. We are given one of the elements of A as the search input *key* and our goal is to find the index in the array at which *key* is present. Describe a recursive divide and conquer procedure for this problem (no pseudo-code necessary). Find its <u>average</u> case time complexity (in terms of number of comparisons). What are the various cases for computing the average complexity? Show your work. You may assume that the input will always be present in the array.

Since we know, that the array is sorted, we can employ binary search to find the index of element.

```
def bsearch (low, high, key)
    mid ← (low + high)/2
    if A [mid] = key
        return mid
    elsif A[mid] < key
        return bsearch (mid+1, high, key)
    else
        return bsearch (low, mid-1, key)

bsearch (0, 2^m-1, key)  ← # init call
```

1

We pass the range of possible values of key.
It checks if middle value of range is equal to key.
If yes then we are done.
If no, and middle value is less than key, then key must be in right half and we call the right half of array.
Else, we call the search in left half.

Best case: when key is median
Worst case: when key is at 0 or 2^m-1 index.

1

cases

Avg. case can have uniformly distributed key across the array.

5. [8 points] Insertions and deletions in balanced binary search trees.

(a) Show the series of B-trees (with t=3) when inserting M,S,F,R,A,K,C,D,E,N,H,P,J,Q,G (in that order) in an empty tree. Use top down insertion. You need to only draw the trees just before and after each split.

$t-1=2$

$2t-1=5$

1. M

2. M S

3. F M S

4. F M R S

5. A F M R S

6. ~~A F K M R S overflow~~

A 6 node seen hence a split followed by insertion of k

M

AFK RS

7. M

ACFK RS

8. M

ACDFK RS

9. D M

AC EFK RS

10. D M

AC EFK NRS

11. D M

AC EFHK NRS

12. D M

AC EFHK NPRS

13. D M

AC EFHJK NPRS

14. D M

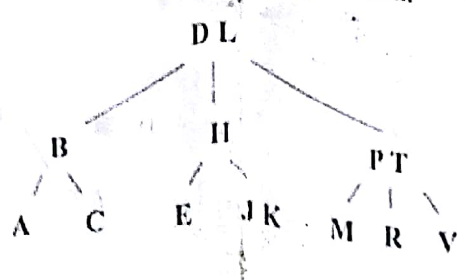
AC EFHJK NPQRS

15. D H M

AC EFG JK NPQRS

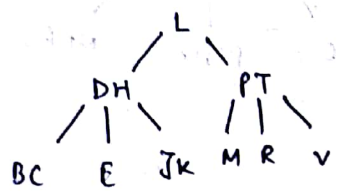
5

(b) Delete the keys A, V, and P from the following 2-4 tree using the top down deletion algorithm discussed in class. Show the result after each deletion.



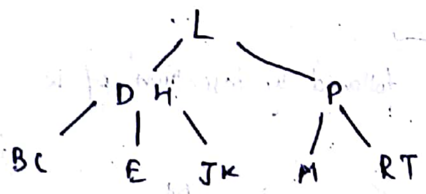
3

1.



Deletion of A

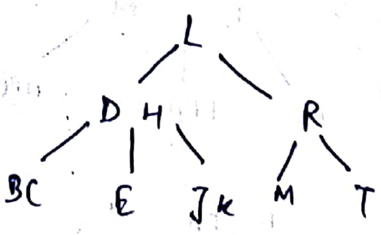
2.



Deletion of V.

3/

3.



Deletion of P

Scanned by CamScanner