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 *g* 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();
```

Right_Rotate (AVLNode a) {

AVLNode x = a.getParent(); // We would get Null if a = root

AVLNode b = a.getLeftChild();

~~AVLNode c =~~

~~If (x == NULL){~~

b.setParent(x); 1

a.setLeftChild(b.getRightChild()); 1

b.setRightChild(a); 1

a.setParent(b); 1

~~If(b.getBalance == 0)~~ a.setBalance(1); 0

b.setBalance(0); 1

}

5

0

(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.

True; AVL trees are formed when we maintain at each node the, difference in height of subtrees is <2 and >-2. This property ensures the tree is balanced as roughly height of both left and right subtrees are equal.

If at any point we get height difference >1 or <-1 we simply rotate the keys and update the heights which maintains the balance.

Now, if at starting we have 1 node, AVL invariant is true as height diff. in subtree =0.

At any i if while inserting we have diff >1 or <-1 while inserting, then we right rotate for (left-left imbalance), left rotate for (right-right imbalance), left and right rotate (left-right imbalance) and right-left for right left imbalance to regain balance.

(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.

We have to continuously find and add medians into the AVL tree to avoid any rotations as the BST formed automatically comes out to be balanced.

Order → (4, 2, 6, 1, 3, 5, 7)

2.5

Median is a specific value that resides exactly in the middle of data (sorted), so we know that there will be $\left(\frac{n}{2} \,\&\, \frac{n}{2}-1\right)$ elements on each side if n is even or $\left(\frac{n-1}{2}\right)$ on each side if n = odd i.e roughly half the elements reside to left of Root and right of root each. If such a property is obtained then we need not balance the tree by rotation but tree comes out balanced as recursively at each step half of the remaining keys go to left and right of root.

Prove for all input permutation

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

Consider the no. of permutation → 5! = 120

Now if we create a decision tree, then we can eliminate at best half possibilities among 120 permutation to reach desired permutation. At each leaf if we continuously remove half of possibilities at best, then in the best case height of a tree with 5! leaves = $\log_2(5!)$ as each permutation resides at a leaf and no. of leaves = no. of permutation = n! = 5!

So, if leaves = n → best height = $\lceil \log_2(n) \rceil$

so with n! → best / proper binary height = $\lceil \log(n!) \rceil$

~~so height of~~

0 +1.5 so best height of this decision tree = $\lceil \log(5!) \rceil$ = $\lceil \log_2(120) \rceil = 7$

~~but at each step or level we have to make n comparisons to divide the tree into 2 halves.~~

~~so Total time to reach will be or sort will be at best $n\lceil \log_2(n) \rceil$ and here $5\lceil \log_2(120) \rceil$~~

~~this $\log(n!)$~~ but $\log_2(n!)$ is a lower bound for this and proves

so a tree with n nodes can be sorted in $n \log_2(n)$ steps at best

so $n \log n > \log(n!) > 7$

So, No. of comparisons has to be greater than 7 in general

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

```
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.
}
```

2.5

(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 $T(n)$.

$T(n) = 3T\left(\frac{2n}{3}\right)$, $T(2) = C$ (constant)

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).

$T(n) = 3T\left(\frac{2n}{3}\right) = 3^2 T\left(\frac{2^2 n}{3^2}\right) = \cdots = 3^i T\left(\frac{2^i}{3^i} n\right)$

Let $n \sim 3^m$, then

$i = \log_3 m$

0

$T(n) = 3^i T\left(\frac{2^i}{3^i} m\right) = m T\left(2^{\log_3 m}\right)$

$2^{\log_3 n} \geq n/2$

$T(n) \geq m T(m/2) = \frac{n^2}{2} T\left(\frac{n}{4}\right) \cdots \frac{n^i}{2^{i-1}} T(1)$

$i \sim \log_2 m$

(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.

Yes, worst-time complexity of insertion sort and Avg-time $= O(n^2)$

Here, we are following the divide and conquer policy of dividing array in 3 parts and then sorting smaller parts and creating a tree sort of structure with height $O(\log_3 n)$.

and $O(\log_3 N) < O(n^2)$

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 average 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.

1

We would follow a binary search procedure.

At every recursion we will compare the key with middle element of present array and if key < arr[mid] we would proceed on with left half array as main array and again compare the key with its middle element. So, at every step we divide our problem into half and stop when arr[mid] == key or if arr[mid] > key and arr[mid-1] < key or vice versa to say key doesn't exist.

~~On an average~~ In best case we find key at 1st comparison ~~X~~ not needed

In worst case we find key after $\log_2 n \approx m$ comparison ~~X~~ not needed

So, let a key be found after i passes in any permutation

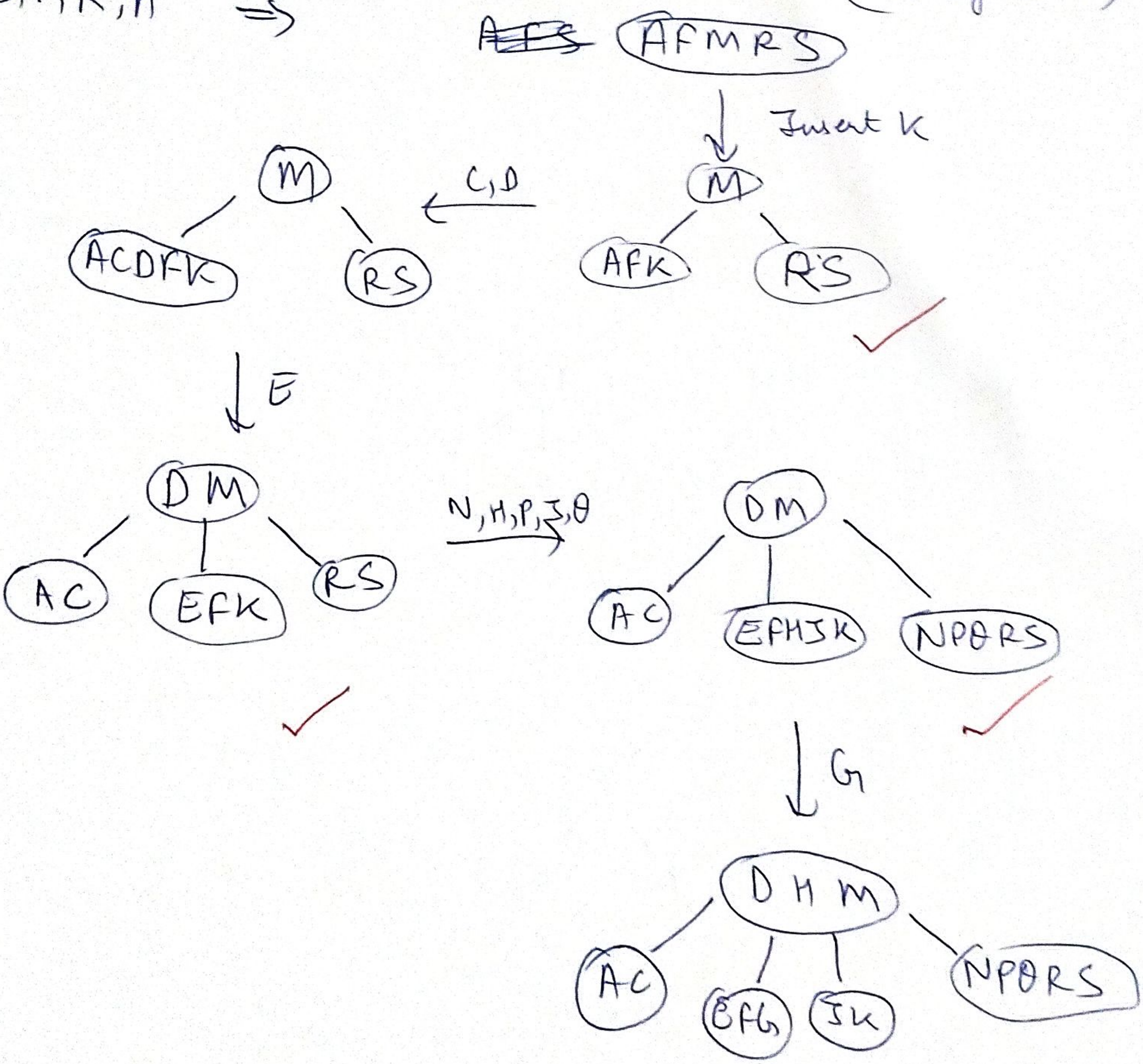
Total permutation = n!

and each i can be a probability $= \frac{1}{\log_2 n}$ ~~X~~

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.

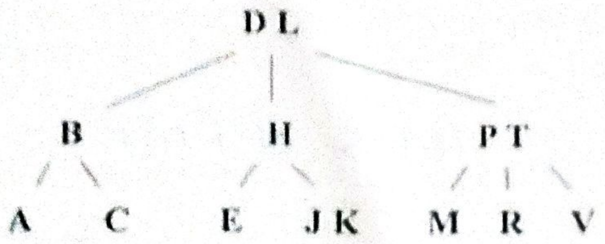
Insert → M, S, F, R, A ⇒ (2 ≤ keys ≤ 5)



↑ Final tree

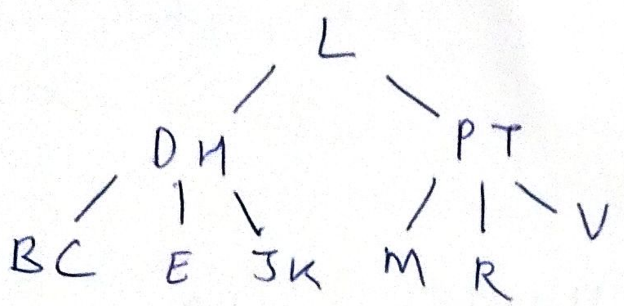
5

(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.

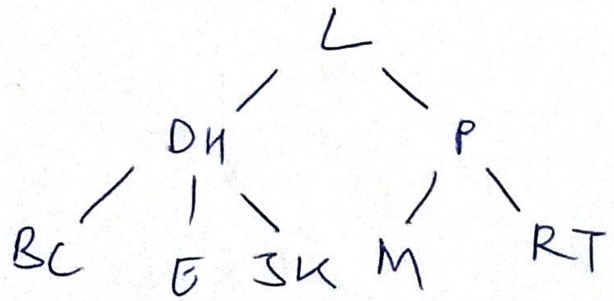


3

↓ A



↓ V



↓ P