33 > n > 2-1 2-3 TREE: A let T be a 2-3 TREE With n LEAVES and >> h-1 > log3 N => h>log3 N+1 a height & h. $\Rightarrow \log_2 n \ge (h-1) \Rightarrow -h \le \log_2 n+1$ The LEAST # & LEAVES 15 = 2^{k-1}. >bg,n+15 h 5 bg, n+1 The MOST # of LEAVES is = 3^{k-1}. H2 x) > h= O(log N).

SEARCH TAKES 27FT has a Single node Mogn) TIME. Then create a new not NSERT (X): (H, Hz ROOT OZT is empty then T becomes X(x))Y (\mathbf{x})

INSERT (34) : SEARCH FOR X in the tree. 3) 3032 sos let p be a node in the 112 1516 (182) 904 tree st. x should be 1/3 (8) Ale) (0) 22 (13) (10) a Child of \$. 51



3a): phas Two CHILDREN RIGHT Now 36 \$ Has 3 CHIOREN. create a new node X (X), MAKE X AN APPROPRIATE CHILD OF INSERT X (x) AS AN , CHANGE H, and Hz VALVES OF APPROPRIATE CHILD OF P. NODES AS NEEDED.

INSERT (34) : SPLIT Pinto P. & E With 2 children Rach. Now of has one more 112 Sos Child. Deal with This 6) (6) (2) problem recursively 32



SPLIT & into K & & With 2 children each. Now g has one more Child. Deal with This problem recursively.

The TIME SPENT IS (1) PER LEVEL.) TIME FOR INSERT IS (log n).) ELETE (X): 1) If T = (x) then the tree becomes Empry $B \ H T = PROT \Rightarrow OY$

3) SEARCH FOR 2. let p be the parent of X. 3a): phas 3 CHILDREN. of So, delete X, MODIFY H, & Hz Volues as needed and quit;

phas only 2 CHILDREN. 36) 361): phas a SIBLING & with 3 Children. Du This Case one Child is bourowed from p and given to p, H, & He values Ne Modified as needed. (X) 15 Deleted.

Case (362): phas no SIBLING WAT 3 CHILDREN. p deletes x; p gets deleted. let g be the parent of p. g Now has one less child. Deal with This problem Decursively



CSE 3500 Algorithms and Complexity – Fall 2016 Lecture 6: September 15, 2016

In the last lecure we completed our discussion on heaps and heap sort. We also introduced 2-3 trees and saw how to perform a search operation in a 2-3 tree.

2-3 trees

- A 2-3 tree is a tree with the following propoerties: 1) Each non leaf has at least 2 and at most 3 children; 2) All the leaves are at the same level; 3) Each leaf stores a key;
 4) The keys in the leaves are in sorted order; and 5) Each non leaf x stores two keys H₁(x) and H₂(x), where H₁(x) is the largest key in the first subtree of x and H₂(x) is the largest key in the second subtree of x.
- Let T be a 2-3 tree of height h with n leaves. The maximum number of leaves in T is 3^{h-1} and the least number of leaves in T is 2^{h-1} . Thus it follows that $(\log_2 n + 1) \ge h \ge (\log_3 n + 1)$ and that $h = \Theta(\log n)$.

Searching in a 2-3 tree

• Let x be the key searched for.

```
N = root;

repeat

if N is a leaf then

if x equals the key of N then output 'yes' else output 'no';

quit;

if x \le H_1(N) then N = N_1 else

if x > H_1(N) and x \le H_2(N) then N = N_2 else

if N has a third child N_3 then N = N_3 else

{ output 'no' and quit};
```

forever

• The above algorithm takes $O(\log n)$ time since we spend O(1) time per level of the tree.

Inserting into a 2-3 tree

- Let x be the key to be inserted into a 2-3 tree T. Let r be the root of T. Create a new node X and store x in it;
- If T is empty, then X is the new tree;
- If T is a single node, then create a new root with r and X as children;
- Search for x in T. Let p be the node such that x should be a child of p;
- If p has currently two children then insert Y as an appropriate child of p; Adjust the H₁ and H₂ values of the ancestors of Y as needed and quit;
- The remaining case is when p has three children. In this case insert Y as an appropriate child of p. p now has 4 children. Replace p with two nodes p_1 and p_2 and assign two children for each of them. If p was the root of the tree, then create a new root r with p_1 and p_2 as children. If p was not the root, then, set the parent of both p_1 and p_2 to be the parent of p (before deletion). Now the parent of p_1 has one more child. Deal with this problem recursively.
- The above algorithm also takes $O(\log n)$ time since the search takes $O(\log n)$ time and followed by this we spend O(1) time per level.

Deleting from a 2-3 tree

- Let x be the key to be deleted from a 2-3 tree T. Search for x in the tree. If x is not in the tree then output 'error' and quit else let X be the node that has the key x and let p be the parent of X;
- If T is a single node X then delete this node and the tree becomes empty;
- If T has a root and two leaves (X being one them) as children then delete X and the root. Now the tree becomes a single node;
- If p currently has three children, then delete X, adjust the H_1 and H_2 values of p and its ancestors as needed, and quit;
- The remaining case is when p has only two children.

Check if p has a sibling, either to the left or right, with three children. If there is such a sibling s, let it be to the left. (The case of a right sibling can be handled in exactly the same manner). Make the right most child of s as the first child of p, make adjustments to the H_1 and H_2 values of nodes as needed (note that we only have to consider the ancestors of p), and quit;

If p does not have such a sibling, delete X from p and donate the only remaining child of p to a sibling of p. Now the parent g of p has one less child. Deal with this problem recursively starting from g.

- In the above algorithm also, we spend O(1) time per level of the tree and hence the total time is $O(\log n)$.
- Theorem: A dictionary can be implemented such that each operation takes $O(\log n)$ time. \Box .
- Note that a 2-3 tree can also used to realize a priority queue.

Algorithms Design: Divide-and-conquer

- Let π be any problem with $|\pi| = n$. To solve π using divide-and-conquer the following steps are involved:
 - 1) Generate k subproblems form π (for some $k \ge 1$). Let these subproblems be $\pi_1, \pi_2, \ldots, \pi_k$;
 - 2) for i = 1 to k do Recursively (or otherwise) solve π_i ;
 - 3) Combine the solutions obtained in step 2 to create a solution for π .
- A classical example of divide-and-conquer is binary search. For this problem the input are a sorted sequence $X = k_1, k_2, \ldots, k_n$ and another element x. The problem is to check if $x \in X$.