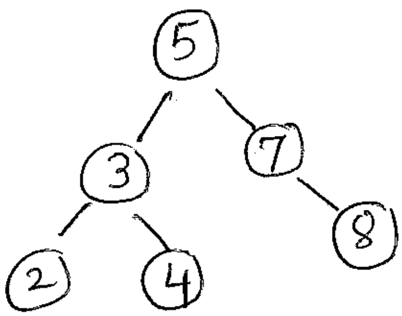
Binary Search Trees

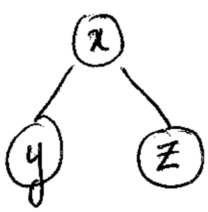
- Store n keys so that SEARCH, INSERT, DELETE can be performed fast.

Example



- A binary tree with key at each node

-



$y < x < z$.

- If h is height of B.S.T. then

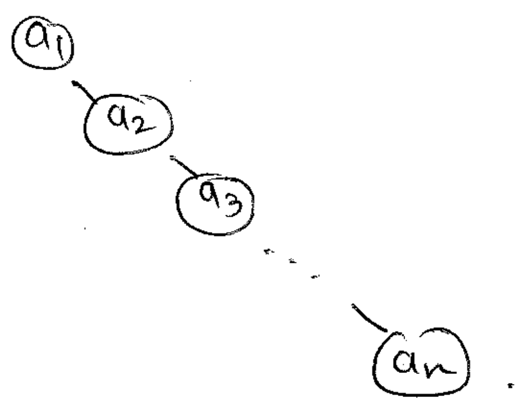
$\underbrace{\text{INSERT, SEARCH,}}_{\text{straight forward}}$ $\underbrace{\text{DELETE}}_{\text{more intricate, exercise.}}$ can be done in $O(h)$ time.

- The tree could be very imbalanced and height may be too large.

- E.g. if n keys are inserted

$a_1 < a_2 < a_3 \cdots < a_n$ in that order

then the B.S.T. would look like



How to make sure $h = O(\log n)$?

Red-black trees — 2-3 trees

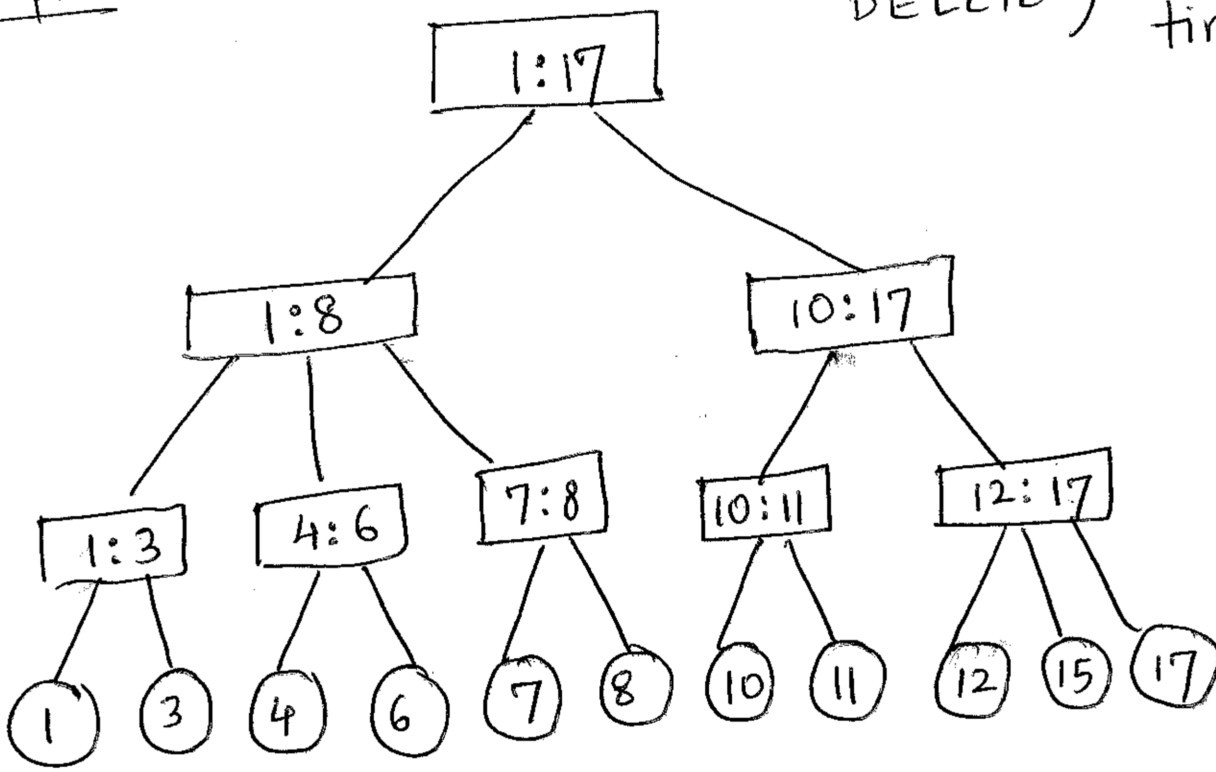
we won't do in this course.

- Both these enable us to store n keys so that SEARCH, INSERT, DELETE can be performed in $O(\log n)$ time.

2-3 Trees

SEARCH
INSERT
DELETE
} All in $O(\log n)$ time

Example

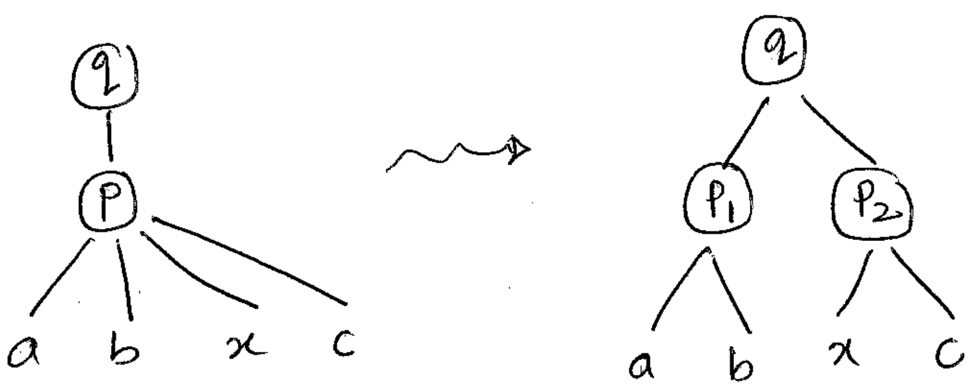


- Each internal node has 2 or 3 children.
- Keys stored only at leaves. Sorted in left to right order.
- All leaves at the same level (depth).
- Each internal node contains "range = [min : max]", the min & max values in its subtree.
- Height = $O(\log n)$ if n = #leaves.

SEARCH. Use the range information at internal nodes.

INSERT

- Insert value x as a child of appropriate internal node p.
- If p still has ≤ 3 children, done.
- Else p now has 4 children. Split p into two nodes, each with two children.



- Now the parent of p, say q, has one more child. Repeat the same process upwards.
- Finally, if the root has 4 children, split it into two and create new root. Height of the tree increases by 1.

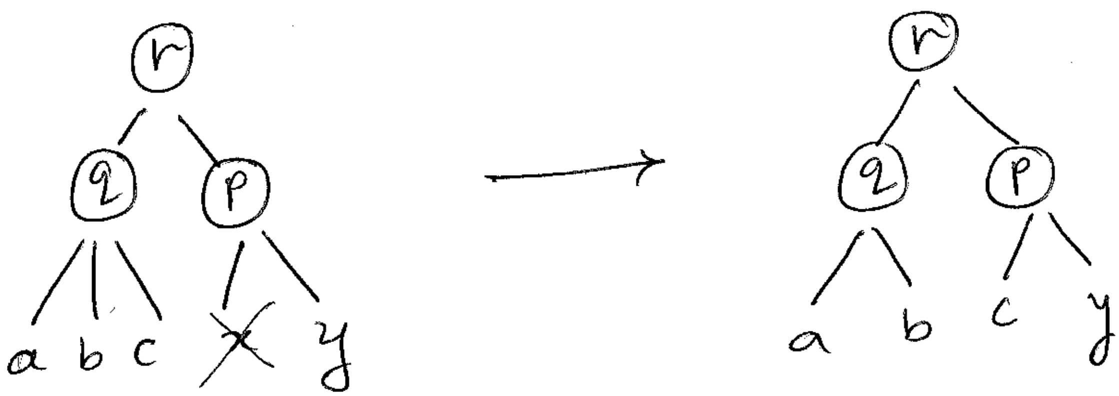
DELETE - Delete x from parent p.

- If p still has 2 children, done.
- Else p only has 1 child now.

    ① If sibling of p has 3 children then p can borrow a child from its sibling.

    ② Else p gives away its child to its sibling. Now p can be deleted. However, parent of p now has one less child, and the process is repeated upwards.

①



②