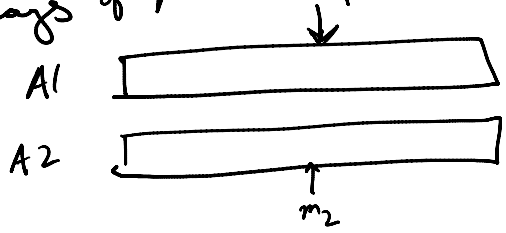
Lecture 3

Monday, August 17, 2015
6:04 PM

Divide-and-conquer

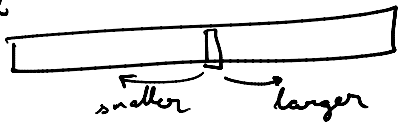
- Split input into parts, recursively solve on each, then merge
- Pruning: special case where one of the parts is empty and merging unnecessary.
- Basic example: binary search, $O(\lg n)$ worst case
- A little more thought: finding median of two sorted arrays of equal size



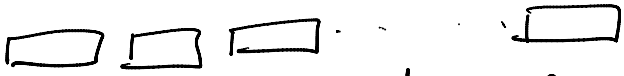
If $m_1 < m_2$, can discard elts left of m_1 in A_1 and right of m_2 in A_2.

$O(\lg n)$

- Median finding in worst case $O(n)$ time
  - Idea: pivot like in quicksort



  - But how to find good pivot?
  - Choose randomly! Will see in next lecture
  - Median-of-medians algorithm



$\frac{n}{5}$ groups of size 5

    - Find median in each group $m_1, \ldots, m_{n/5}$
    - Let m = median $(m_1, \ldots, m_{n/5})$ found recursively
    - Pivot around m.
  - # of elts smaller than $m \geq 3 \cdot \frac{1}{2} \cdot \frac{n}{5} = \frac{3}{10} n$
  - So, # of elts larger than $m \leq \frac{7}{10} n$. Sim, smaller.

$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10} n\right) + O(n)$

$= O(n)$ [exercise !]

- Karatsuba multiplication
  - Let x and y be two n-digit numbers. Usual algorithm takes time $O(n^2)$.
  - $x = 2^{n/2} x_1 + x_0$, $y = 2^{n/2} y_1 + y_0$, x_0, x_1, y_0, y_1 $n/2$-digits
  - $xy = 2^n x_1 y_1 + 2^{n/2}(x_1 y_0 + x_0 y_1) + x_0 y_0$
  - But: $x_1 y_0 + x_0 y_1 = (x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0$
  - $T(n) = 3T(n/2) + O(n)$
    $\Rightarrow T(n) = O(n^{\log_2 3})$
  - Better multiplication algorithms through FFT, another example of divide and conquer
- Review of master rule
  Suppose $T(n) = a \cdot T(\lceil n/b \rceil) + O(n^d)$.
  Then: $T(n) = \begin{cases} O(n^d), & \text{if } d > \log_b a \\ O(n^d \log n), & \text{if } d = \log_b a \\ O(n^{\log_b a}), & \text{if } d < \log_b a \end{cases}$

- Strassen's algorithm
  - X, Y two $n \times n$ matrices. Usual algorithm to compute $Z = XY$ takes $O(n^3)$ steps
  - $X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$, $Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$, $Z = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$
  - But! $P_1 = A(F-H)$, $P_2 = (A+B)H$, $P_3 = (C+D)\cdot E$, $P_4 = D\cdot(G-E)$, $P_5 = (A+D)(E+H)$, $P_6 = (B-D)(G+H)$, $P_7 = (A-C)(E+F)$

$$Z = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

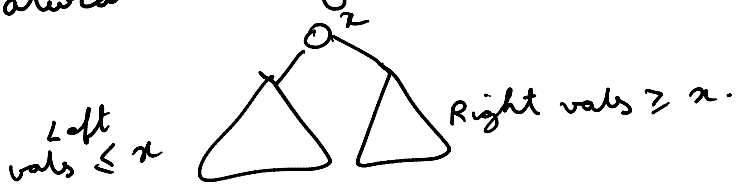
$T(n) = 7T(n/2) + O(n^2) \Rightarrow T(n) = n^{2.81}$

- Another familiar example: Merge Sort

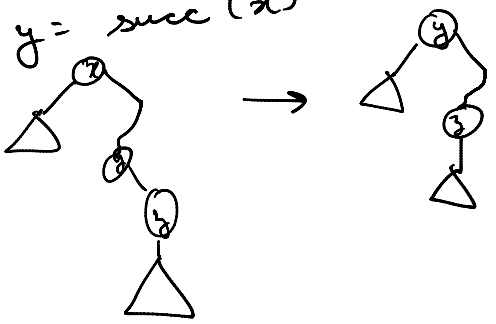
$$T(n) = 2T(n/2) + O(n)$$

Binary search revisited

- Suppose set on which we're searching is <u>dynamic</u>: elements inserted and deleted
- Want data structure where insertions, deletions and search queries are fast.
- Natural: binary search tree



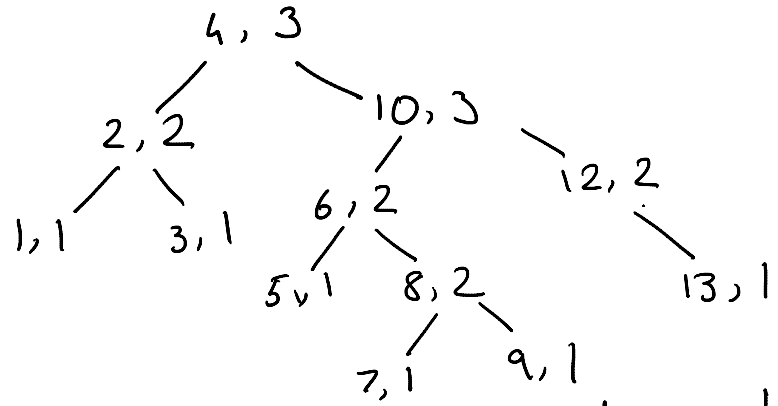
- Search: $O(\log n)$
- Insertion: Attach as leaf $O(\text{height})$
- Deletion: (1) x is a leaf. Just remove it.
  (2) x has only one child. Then move the child up to take x's place
  (3) Let y = succ(x)



$O(\text{height})$

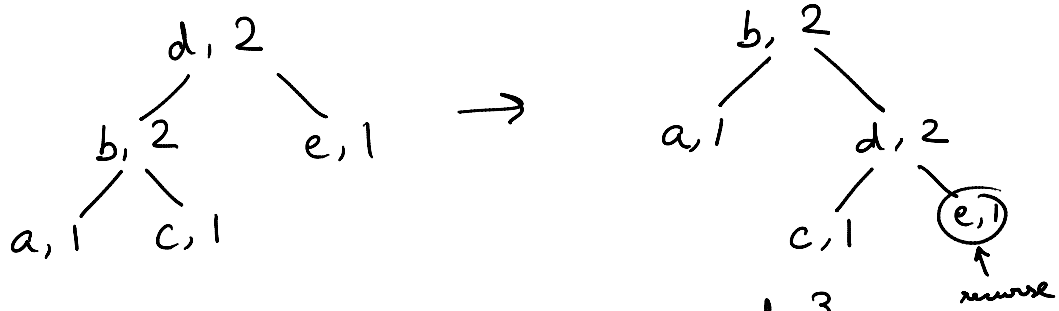
- But of course issue is that if above, height could be $O(n)$.
- Several self-balancing trees where height maintained to be $O(\log n)$. E.g. Red-Black trees [CLRS]
- Here, AA Trees, a simpler variant of red-black trees.
- Each node has a <u>level</u>.

- Invariants:
  - Leaf level
  - Every path has the same # of levels
  - Left child is always one level less. Right child is at equal level or one level less.
  - For no x, $level(x) = level(right(x)) = level(right(right(x)))$
- Check: Height of an AA tree = $O(\lg n)$
- Example:

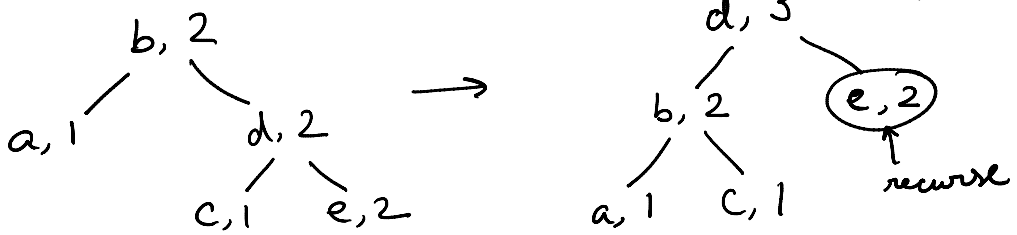


- Upon insertion/deletion, some invariant may break.
  To fix AA trees, two operations:

Skew:



Split:



- Insertion

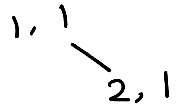
```
INSERT (root, data):
    if root == null
        root = MakeNewNode(data, 1)

    else if data > root.data
        INSERT(root.right, data)
```

else INSERT (root.left, data)
root = skew (root)
root = split (root)

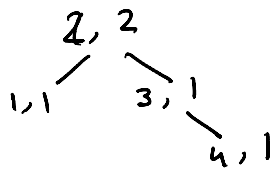
INSERT (1):

INSERT (2):

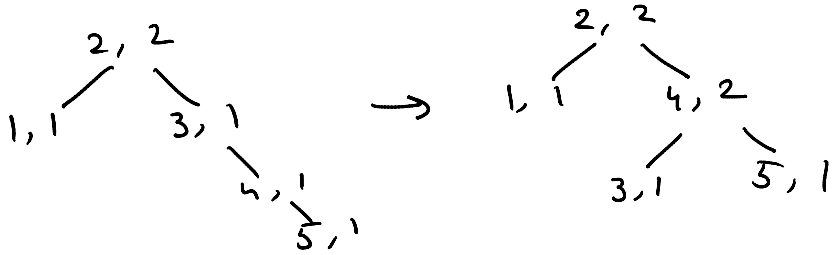


INSERT (3):

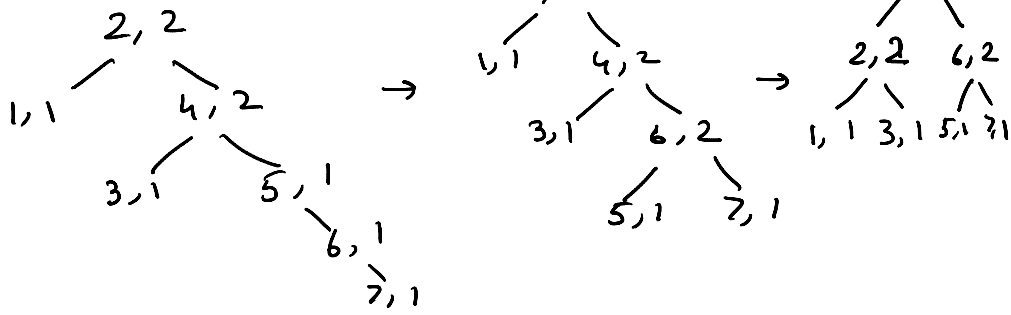
INSERT (4):



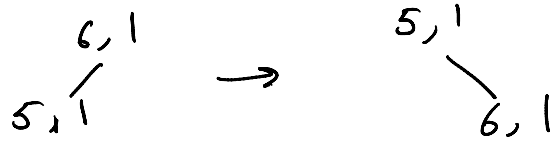
INSERT (5):



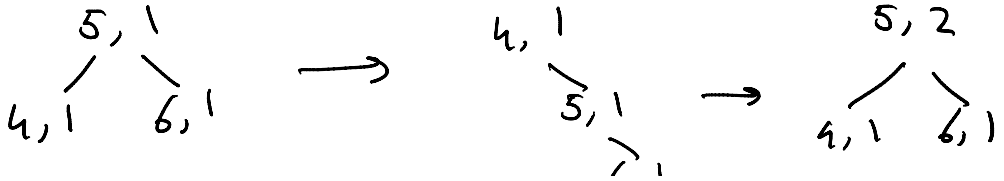
INSERT (6)
INSERT (7):

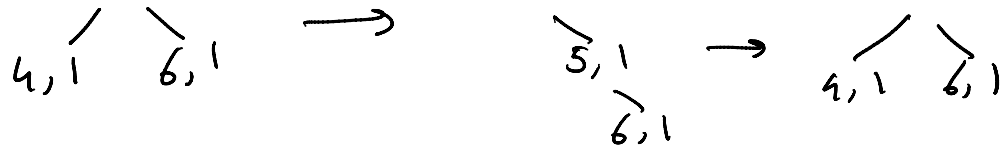


INSERT (6)
INSERT (5)

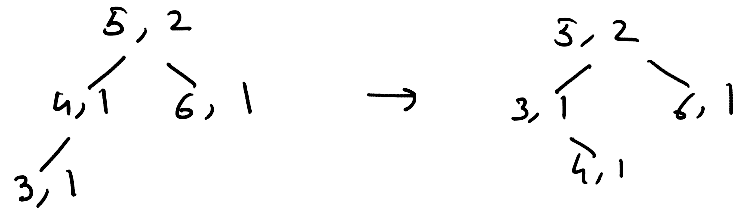


INSERT (4)

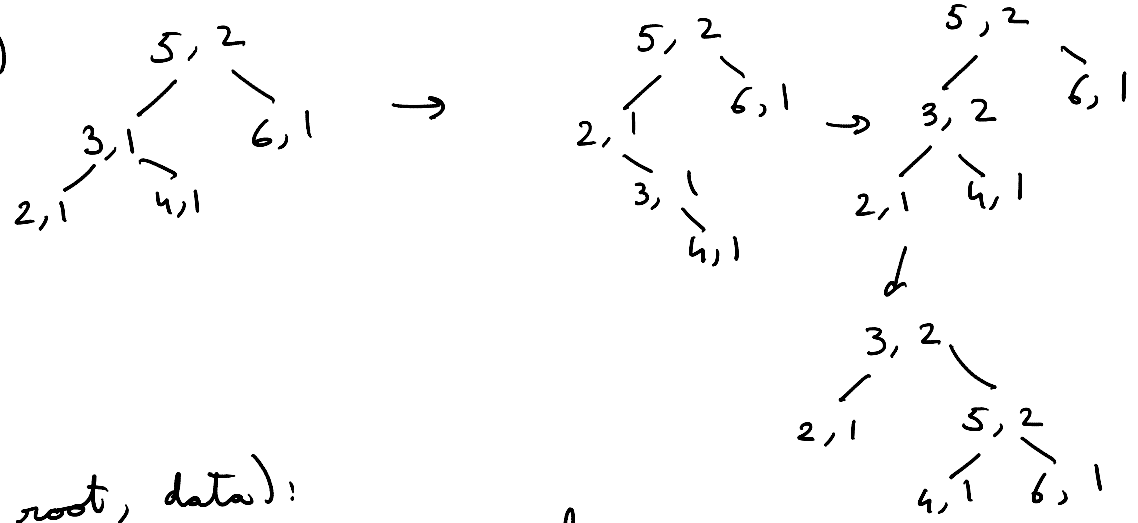




INSERT (3)



INSERT (2)



Deletion

DELETE (root, data):

Delete node with data as usual

If (root.left.level < root.level - 1

OR root.right.level < root.level - 1) {

root.level --

If root.right.level > root.level

root.right.level --

skew (root)

split (root)

Example

More on BST's later:

- Randomized versions (Skip lists)
- Amortized analysis (Dynamic Optimality Conjecture).