LL :
insertion : O(n)
deletion : O(n)
search : O(n)

Trees
insertion : O(logn)
deletion : O(logn)
search : O(logn)

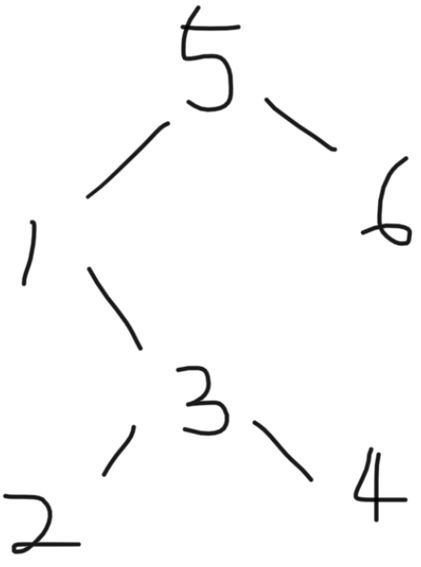
} balanced tree

# Balance

## 1. Randomising

123456

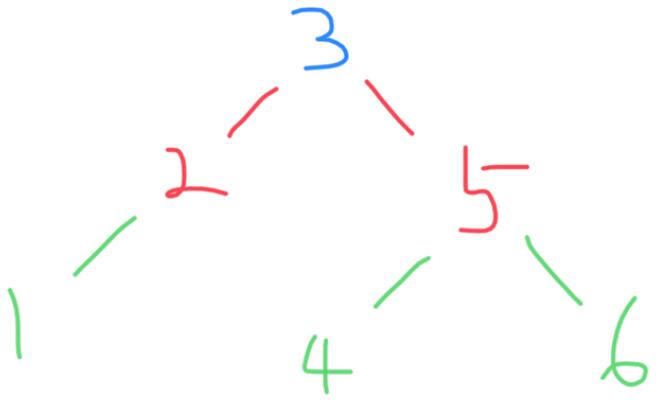

513462




## 2) Sorting → Taking medians

123456

Prefix order



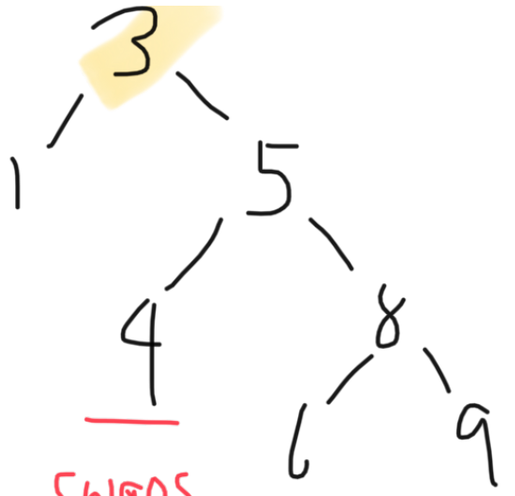

# 3. Self-balancing Trees

Insertion
Search
Deletion
{ Rotation
{ Balancing

## Rotation

### Left Rotation

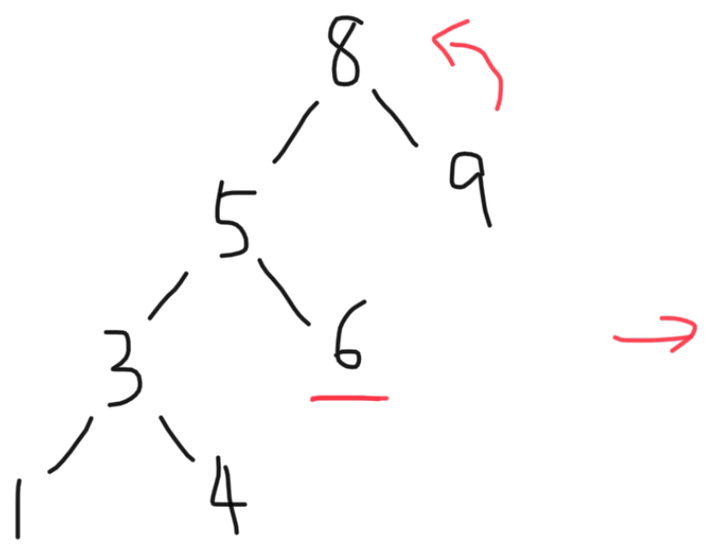

### Right Rotation

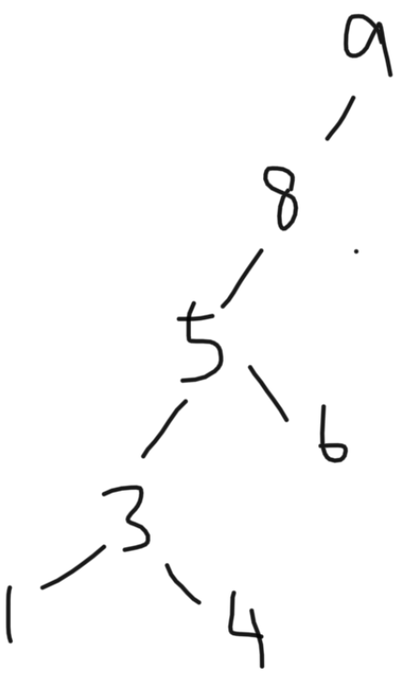




swaps sides

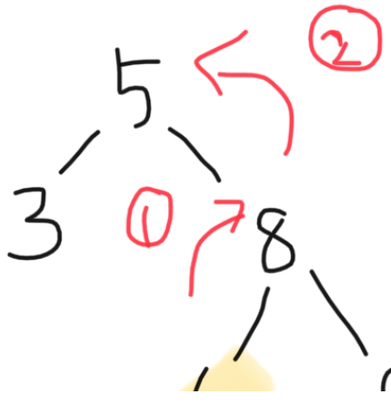
LL


splay trees: conventionally root-root (1,2) rotation for LL, RR

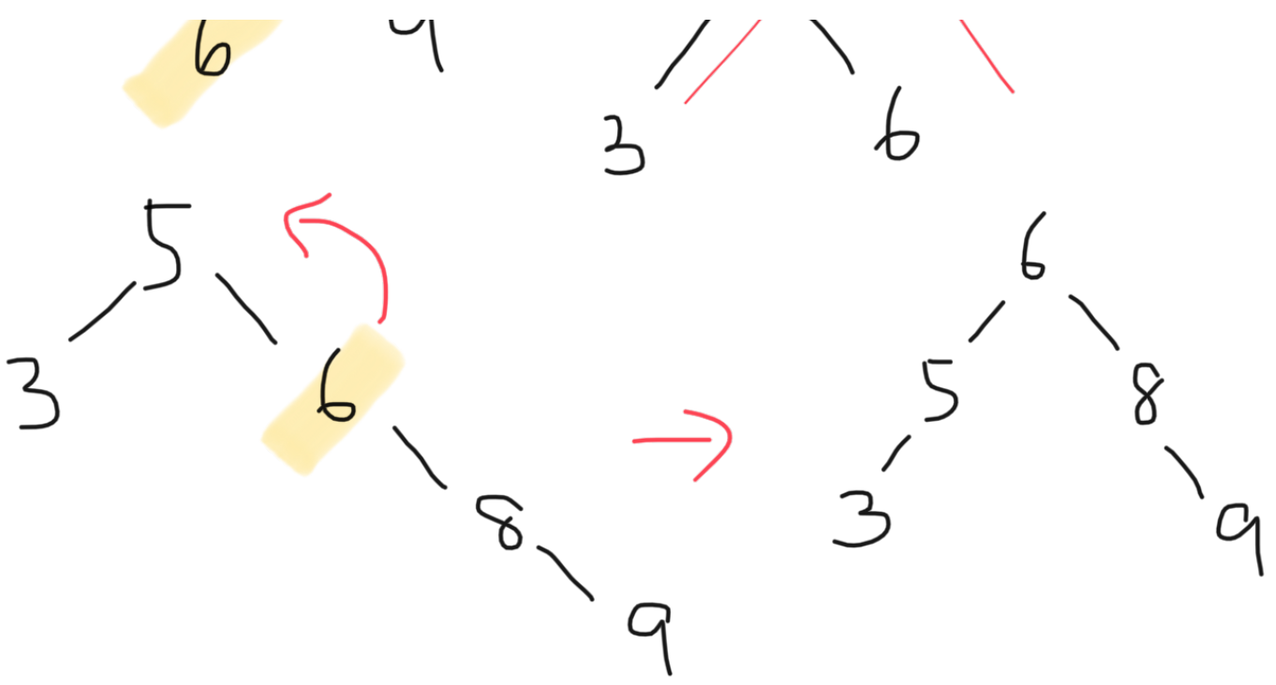



RL




(2) first:




Splay Trees:
LL, RR
LR, RL

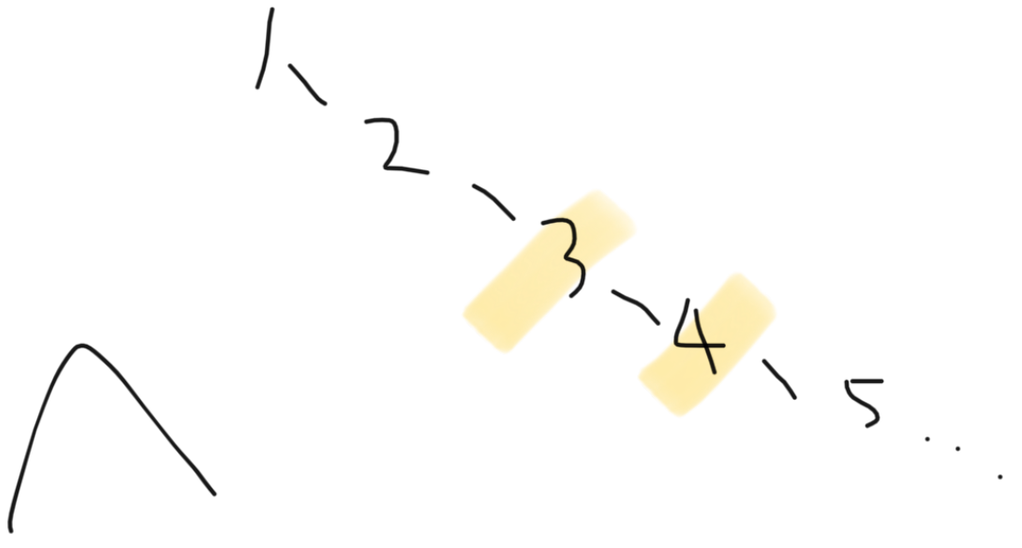
$$s\left(n, \begin{array}{c} t \\ l \quad r \end{array}\right)$$

$$= \begin{cases} s\left(n, \begin{array}{c} t \\ s(n,l) \quad r \end{array}\right) & n < t \\ s\left(n, \begin{array}{c} t \\ l \quad s(n,r) \end{array}\right) & n > t \end{cases}$$

# Splay Trees:

Insertion: insert + splay to root
Search : splay to root
Delete : splay parent to the top

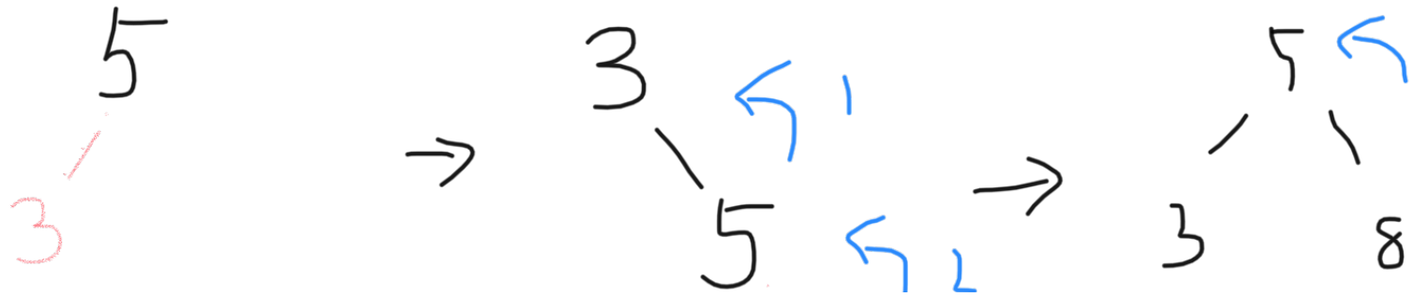
80% of accesses are for
20% of the data

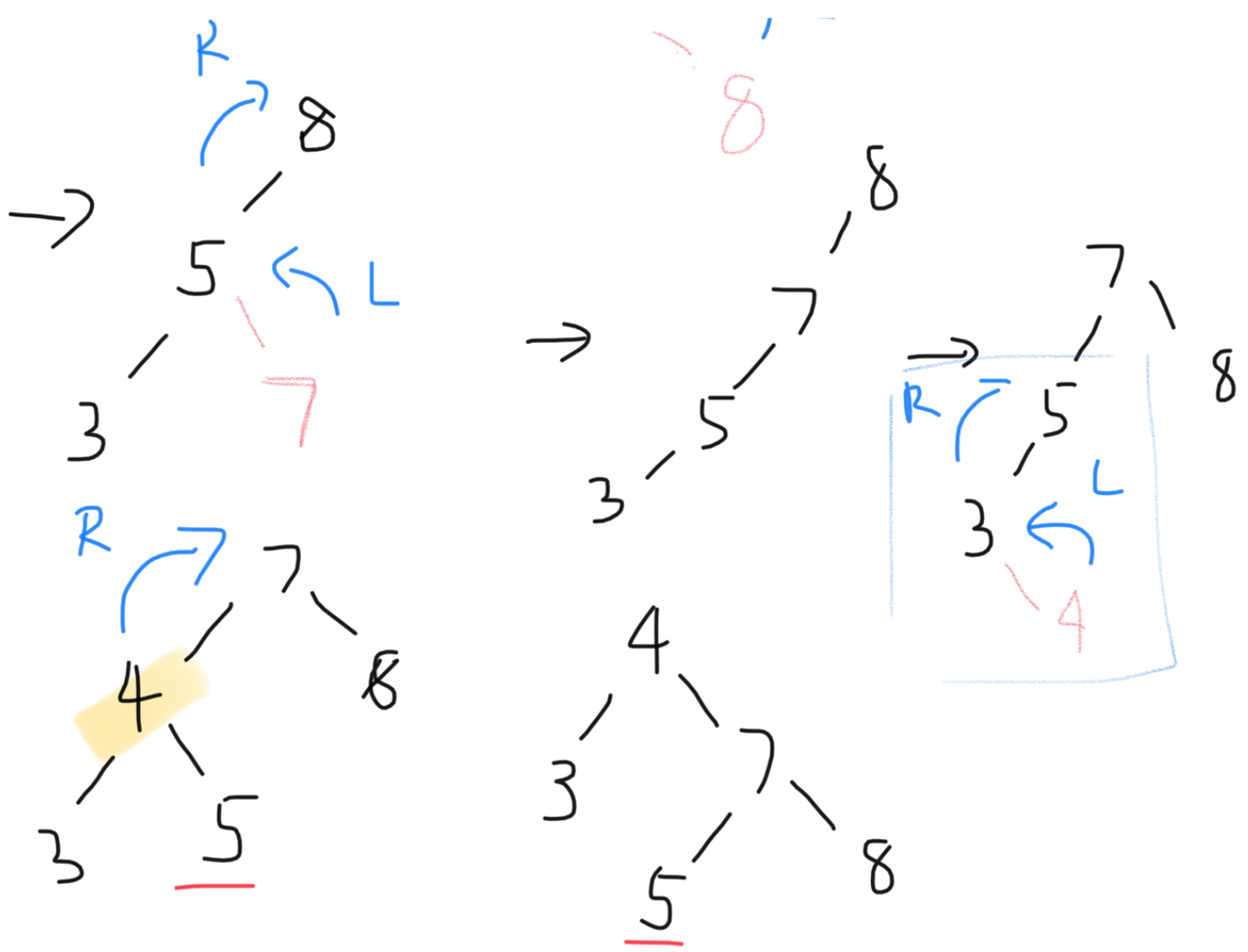



In the long term, tree is ~balanced

→ amortized O(logn) performance

5 3 8 7 4






# AVL Trees

- Always balanced
- After every op, fix balance if necessary
- requires a 'height' field in tree struct

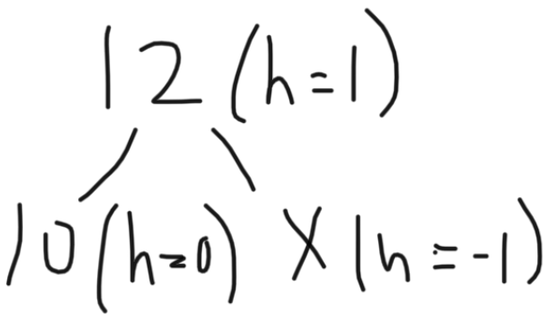
12 10 8 6 4 2

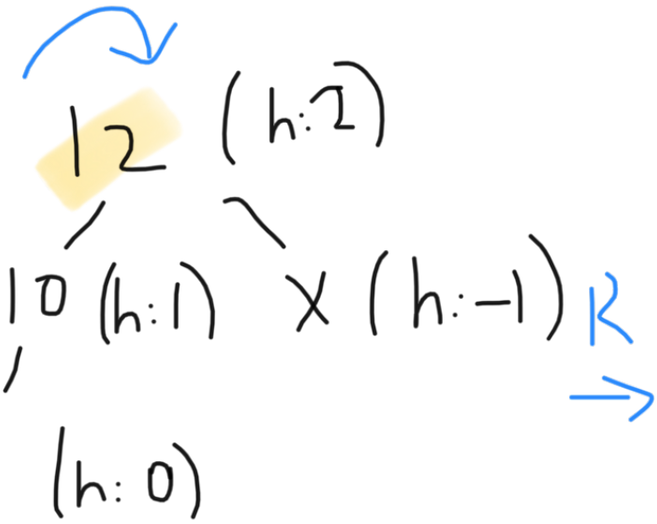
12 (h=0)

height(leaf node) = 0
height(NULL) = -1
balanced:

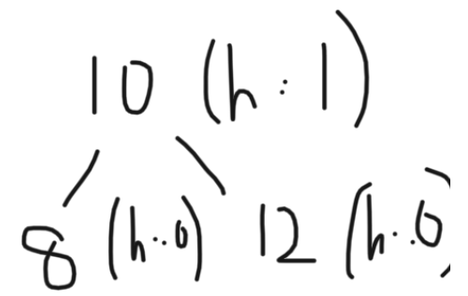
$|h(l) - h(r)| \leq 1$

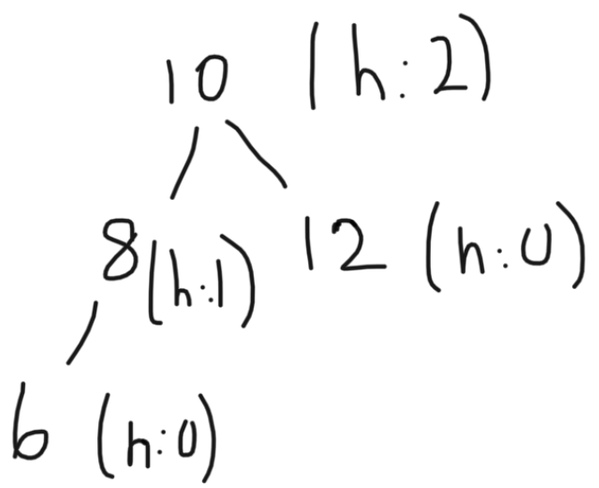
$h(n) = 1 + \max(h(l), h(r))$

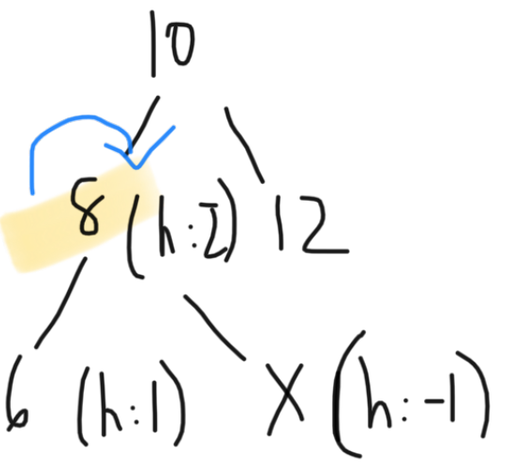


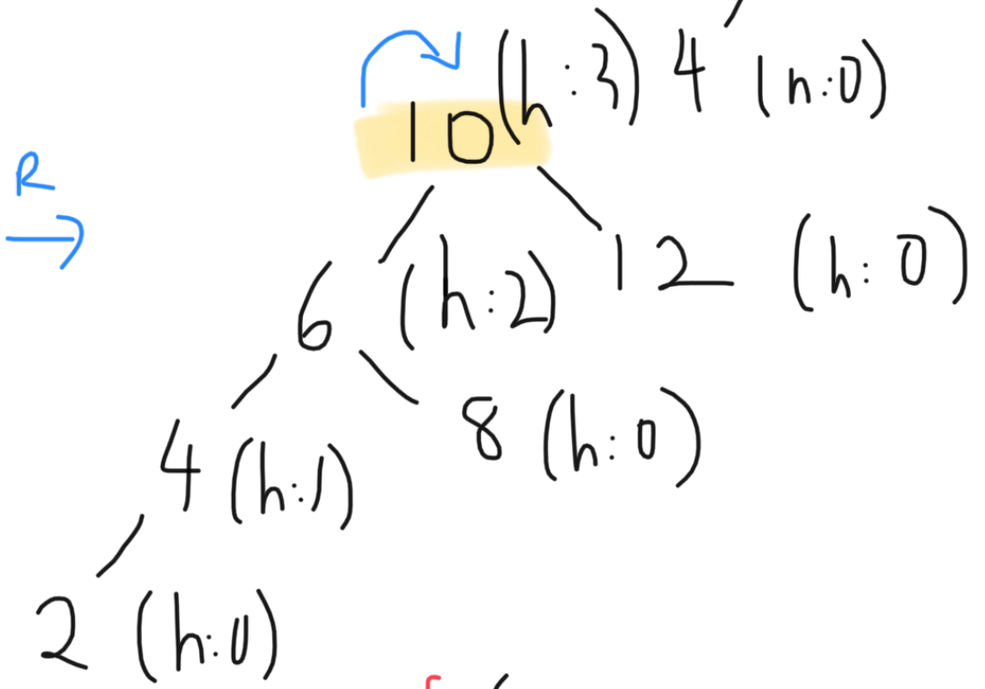




R →



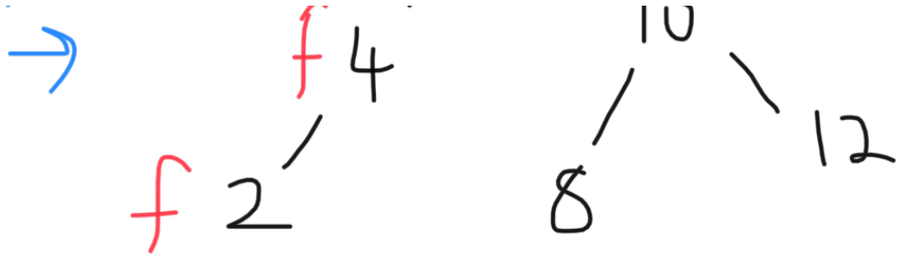




R →


R

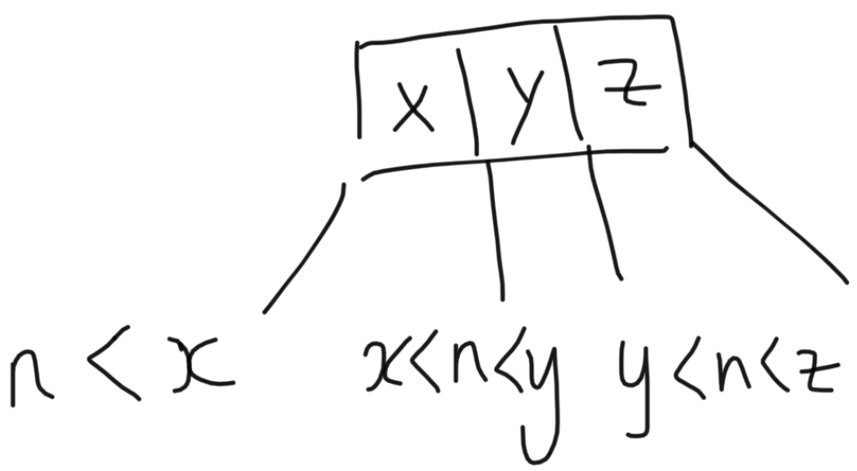



AVL Trees:

→ rebalance at the pt of imbalance

→ <u>height</u>

234 Trees




$l < n < r$

```
struct {
    int val[3];
    tree children[4];
};
```

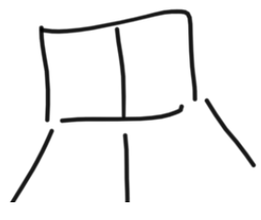
2: # promoted

3: # values

4: # children

23 Trees

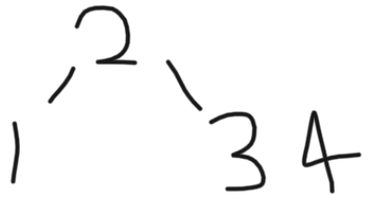
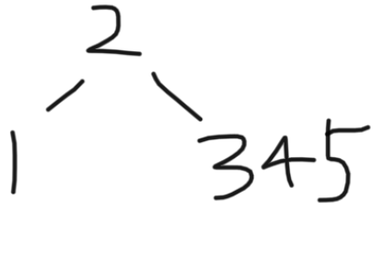
BTrees

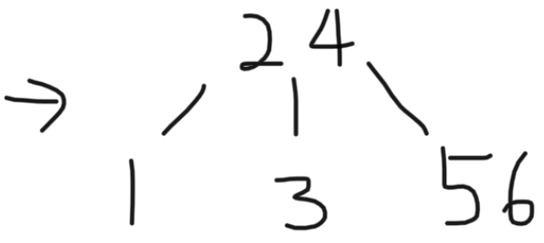
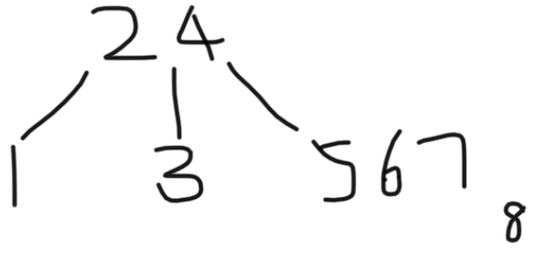


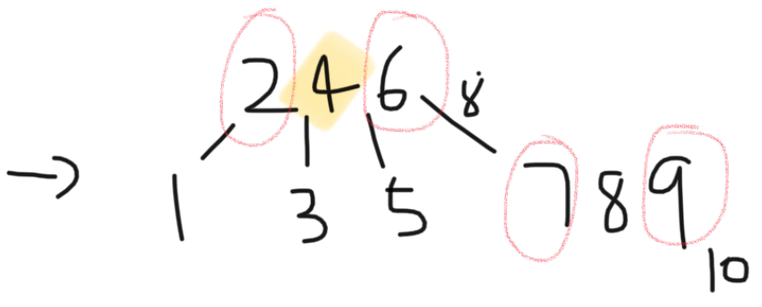
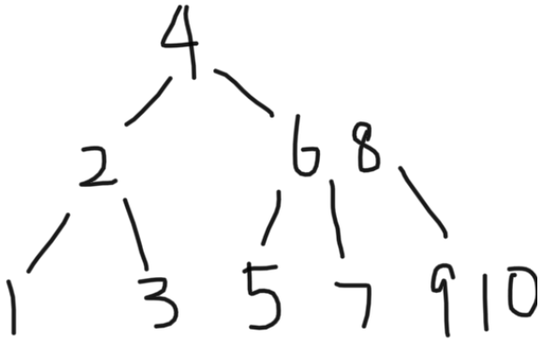
1~12

1 11 1

1 → 12 → 123 4

→  →  6

→  →  8

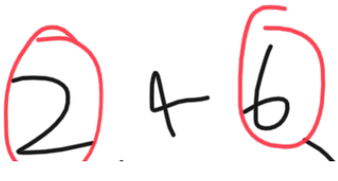
→  → 

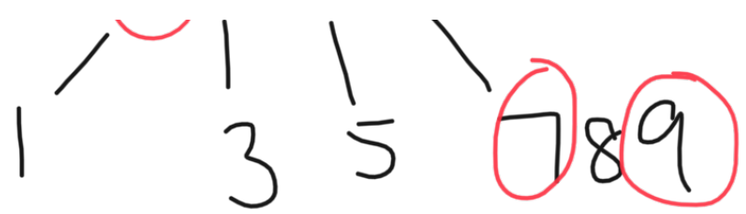
# Red-Black Trees

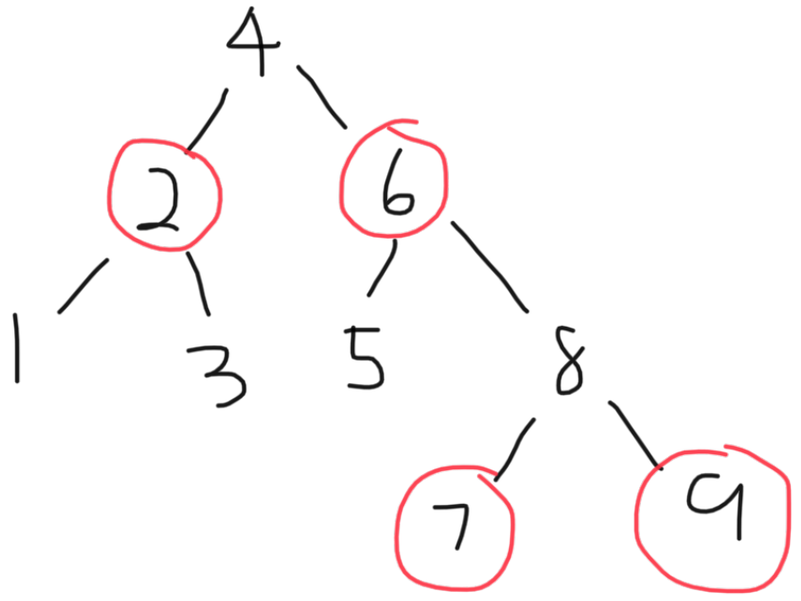
Colour: R, B

Red: siblings

Black: direct children






Balance:

L & R have the same # of black nodes

$h(l) = 2 \times h(r)$ (worst case)

Pros:

- insertion + deletion are faster than AVL
- memory

  1 bit: R/B

  int: height

Cons:

- Hard to implement
- Slightly worse performance for search (imp. AVL)