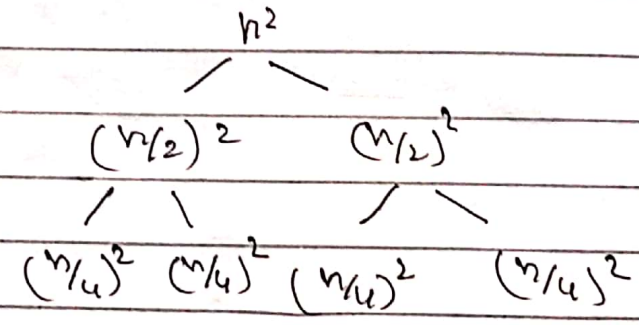
Recurrence Tree Method

In this Method, we draw a recurrence tree and calculate the time taken by every level of tree. Finally, we sum all the levels. To draw the recurrence tree from the given recurrence and keep drawing till we find a pattern among levels. The pattern is typically a arithmetic or geometric series.

Ex

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



$$T(n) = a\,T(n/b) + f(n)$$

~~If is~~

-> It is divided into a subproblems
-> f(n) will become f(n/b) (each level)
-> Prob starts with f(n)

Mastex Theorm

$T(n) = \Theta(n^d)$ if $a < b^d$

$T(n) = \Theta(n^d \log_b n)$ if $a = b^d$

$T(n) = \Theta(n^{\log_b a})$ if $a > b^d$.

$T(n) \leq a\,T(n/b) + \Theta(n^d)$.

AVL

→ Self balancing BST

→ Property : For any node in the AVL tree the balance factor which is the difference of the height of left subtree and right sub-tree should be atmost 1.

Balance Factor (node) $\in \{-1, 0, 1\}$.
$\forall$ node $\in$ Tree.

$F(n) = F(n-1) + F(n-2) + 1$

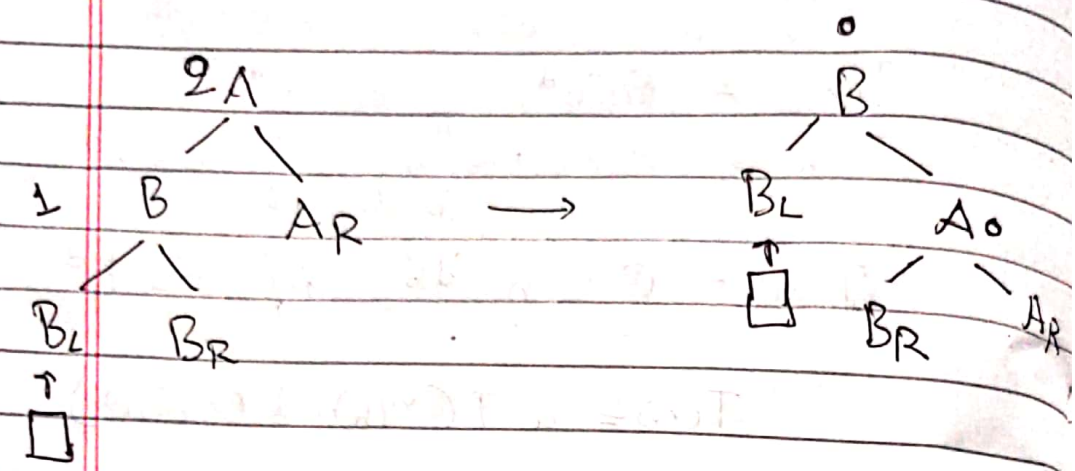
$n \rightarrow$ height.

↳ minimum possible nodes

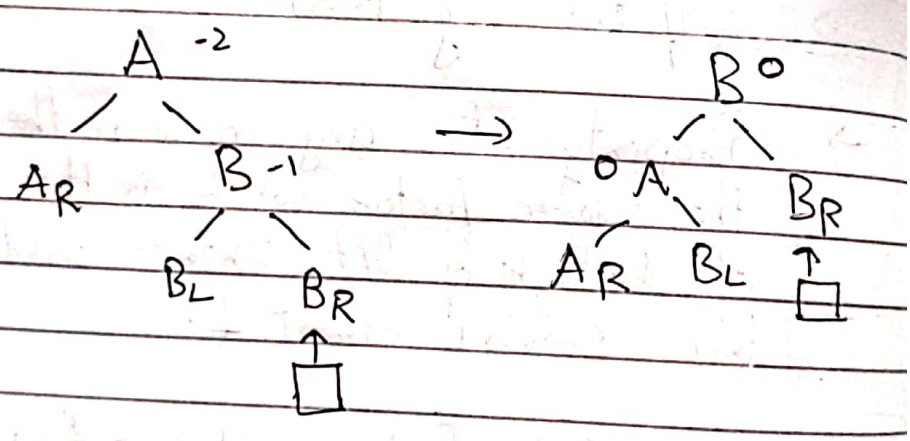
→ $N(h) = N(h-1) + N(h-2) + 1$

$N(0) = 1 \quad N(1) = 2$.

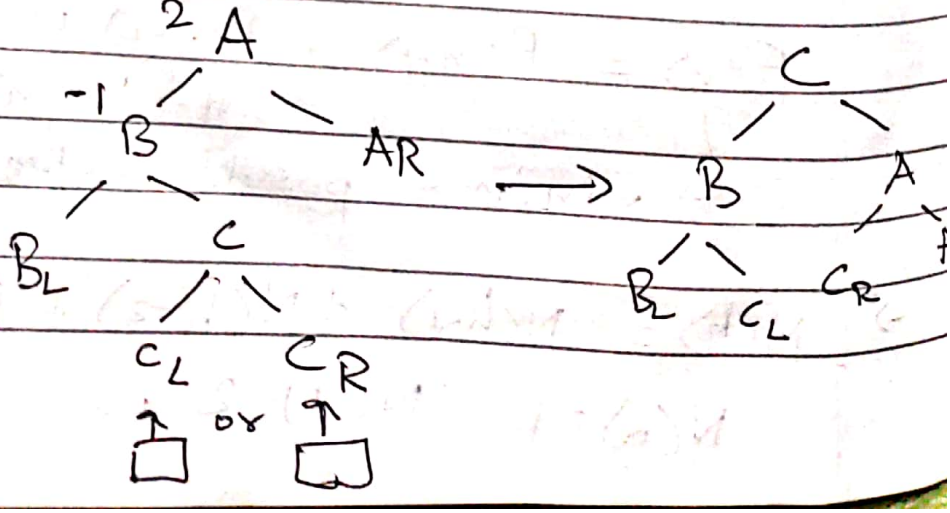
LL Rotation



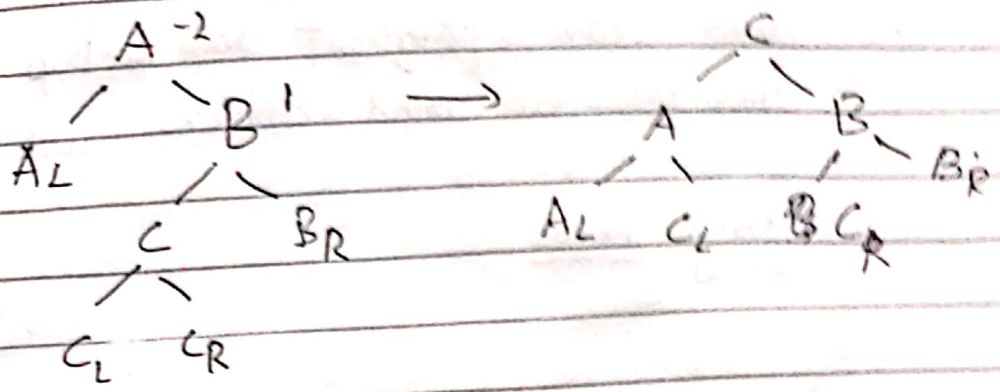
RR Rotation



LR Rotation



RL Rotation



RType (The node is deleted is a right child of unbalanced node)

If the left child of unbalanced node has BF.

0 : LL
1 : LL
-1 : LR

LType

If the Right ~~child~~ child of unbalanced node has BF

0 : RR
1 : RL
2 : RR

~~Analysis~~

-> Used when frequent data lookup rather than insertions and deletions.

Splay trees

-> Splay is also a self balancing BST

-> The objective of splay tree is to bring the recently accessed item to root of the tree.

-> Splay trees are used in
  -> caches
  -> memory allocators
  -> data compression

Amortized Analysis

Goal is to show that although some individual operations are costly on an average the algorithm is cheap.

-> probability is ~~required~~ involved.

Aggregate Method

In this, we show that for all n, a sequence of n operations take worst case time T(n) in total. In the worst case, the avg cost, or amortized cost, per operation is Therefor T(n)/n.

Multiway Search Trees

-> Extension of BST (Generalization)
-> Contains m-1 keys in each node (order is m)
-> Data should be in ascending order in each node.

-> Maximum number of elements if height is $m^{h-1}-1$
   └> m is order

-> In each level max number of nodes is $m^{h-1}(m-1)$

$h_{min} = h$. (skewed)
$n_{max} = m^h - 1$ ->

Best case.

$n = m^{h-1}$

$\log_m n + 1 = h$.

Worst case = O(h)

$h_{min} = \log_m (n+1)$ $h_{max} = n$.

B-Trees

→ Is an m-way search tree
→ If not empty holds the following condn
→ B tree is a fat tree.
→ The root node must have at least two children or atmost m children.

→ All internal nodes must have atleast $\lceil m/2 \rceil$ non-empty child node and atmost m non-empty children.

→ All ~~external~~ leaf nodes at the same level

→ Insertion done in leaf nodes.

Uses of B trees

→ The main idea of using B-Trees is to reduce the number of disk access (used in ~~RAM~~ secondary data storage).

→ $h_{min} = \left\lceil \log_m (n+1) \right\rceil - 1$ ($\log n$ for existing)

$h_{min} = \left\lfloor \log_{\lceil \frac{m}{2} \rceil} \frac{(n+1)}{2} \right\rfloor$

B+ Trees

-> In B trees we store key value and a pointer to the next node in the children slot. In B+ trees it is not needed as leaf node point to other

-> ~~structurally stored~~

-> Leaf nodes stored structurally as linked list.

-> Redundant keys are stored in B+

~~and~~ analysis -> O(log n).

Divide and Conquer

-> If problem instance small enough, solve problem directly, return solution

-> Otherwise divide a prob. instance into smaller instances of the same problem.

-> Recursively solve the smaller instances.

-> Combine the solution to the smaller instances to get sol. for the original instance, return solution.

$\log^{n}(n-1)\log n$

Merge Sort

$T(n) = 2T(n/2) + n$ → For calling merge function.

All $O(n) = n \log n$ For calling merge sort

Quick sort

-> Fix the first element as pivot

-> initialize i as 1 and j as last element

-> increment i till i^{th} position is greater than pivot

-> Decrement j till j^{th} element is less than pivot.

-> Then exchange them. (pivot & j).

WC -> when array is always sorted.

$T(n) = T(n-1) + n$

$O(n^2)$.

BC $T(n) = 2T(n/2) + n$.

Karat Suba

$a = a_1 \times 10^{n/2} + a_0$

$b = b_1 \times 10^{n/2} + b_0$

$c = c_2 10^n + c_1 \times 10^{n/2} + c_0$

$c_2 = a_1 \times b_1$

$c_0 = a_0 \times b_0$

$c_1 = (a_1 + a_0) \times (b_1 + b_0) - (c_2 + c_0)$

$n \rightarrow$ no. of digits

$T(n) = 3T(n/2) + n$

$T(1) = 1$

Use substitution method.

we get $n^{\log_2^3} = n^{1.585}$

Strassen's

$$\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}$$

$$= \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

$m_1 = (a_{00} + a_{11}) \times (b_{00} + b_{11})$

$m_2 = (a_{10} + a_{11}) \times b_{00}$

$m_3 = a_{00} \times (b_{01} - b_{11})$

$m_4 = a_{11} (b_{10} - b_{00})$

$m_5 = (a_{00} + a_{01}) \times b_{11}$

$m_6 = (a_{10} - a_{00}) \times (b_{00} + b_{01})$

$m_7 = (a_{01} - a_{11}) + (b_{00} + b_{11})$

$T(n) = 7\,T(n/2)$