02.01.2024
Tuesday

DATE
PAGE

ADVANCED DATA STRUCTURES (Meenu Chawla mam)

Dictionary operations
(i) Search (ii) Delete (iii) Add / Insert.

Array	O(n)	Deletion & Insertion	sorted data O(log n) search
Linked list	O(n)	searching	
Trees BST	O(Height)	Searching, Deletion, Insertion.	

O(n) BST — O(log n) AVL Tree {Balanced BST}

Complete Binary Tree :- All h-1 levels are full & last level filled from left to right.

Heap :- Complete Binary tree (Structural prop.)
↳ max Heap (ordering prop.)
↳ Parent value > Both children ~~of~~ all nodes)

No. of internal nodes = $\lfloor \frac{n}{2} \rfloor$
Leaf nodes = $\lceil \frac{n}{2} \rceil$.

$$\frac{n}{2} \times 0 + \frac{n}{4} \times 1 + \frac{n}{8} \times 2 + \cdots + \frac{n}{n} \times \log n$$

$$\frac{n}{2}\left(0 + \frac{1}{2} + \frac{1}{4} + \cdots \frac{\log n}{(\frac{n}{2})}\right)$$

$$\frac{n}{4}\left(1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{\infty} \log n\right) = \frac{n}{4}\left(\frac{1}{1-\frac{1}{2}}\right)$$

$= \frac{n}{4} O(2) = \frac{n}{2}$.

O(n) amortised

10.01.2024 Wednesday

AMORTIZED COMPLEXITY

⊙ Average the time required on a series of operation.

⊙ Do not depend on I/p (or O/p)

n operations: 1, 1, 1, 1, 1, 1, ... n

$= \frac{2n}{n+1} = O(1)$ Amortized complexity

① Aggregate Method $\frac{\Sigma}{n}$ cost to operation → Create/delete object of data structure

② Accounting Method

③ Potential Method. Potential of data structure

credits & Potential should not be -ve

Aggregate method.

Stack

push(s, x) - O(1)

pop() - O(1)

Argument Stack

multipop(s, k) - O(n)

↳ ~~for pop~~

$\frac{2k}{k+1} = O(\frac{2}{2}) = O(1)$ = constant.

Dynamic Table / Array - Doubled

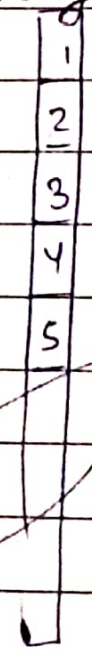
Insert → 1

Insert → 2

Insert → 3

Insert → 4

Insert →



Size of array

cost of insertion

Asymptotic → $O(n^2)$

i	1	2	3	4	5	6	7	8	9
size;	1	2	4	4	8	8	8	8	16
cost;	1	2	3	1	5	1	1	1	9

cost of i ~~th~~ insertions = i when i-1 is an exact power of 2

= 1 ~~for all other~~ cases

18.01.2024 Thursday

1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = n

$S = 1 + 2 + 4 + 8 \ldots (\log n - 1) + 1$ terms

$= \frac{1\,(2^{\log(n-1)+1} - 1)}{2-1}$

$= n \log(n-1)$

$1 + 2 + 4 + \cdots < 2n$

Cost of i^{th} insertions $\leq n + 2n$

Cost per insertions $= O(1)$ amortised $\frac{3n}{n}$

ACCOUNTING METHOD

C_i is the actual cost

$\hat{C}_i$ - amortised cost for a given operation

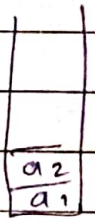
Credit should not be -ve in a series of operation at any time.

$$\sum_{i=1}^{m} C_i \leq \sum_{i=1}^{m} \hat{C}_i$$

↳ (upper bound on actual cost)

	C_i	$\hat{C}_i$	
Push	1	2	O(1)
Pop	1	0	O(1)
Multipop	k	0	O(1)

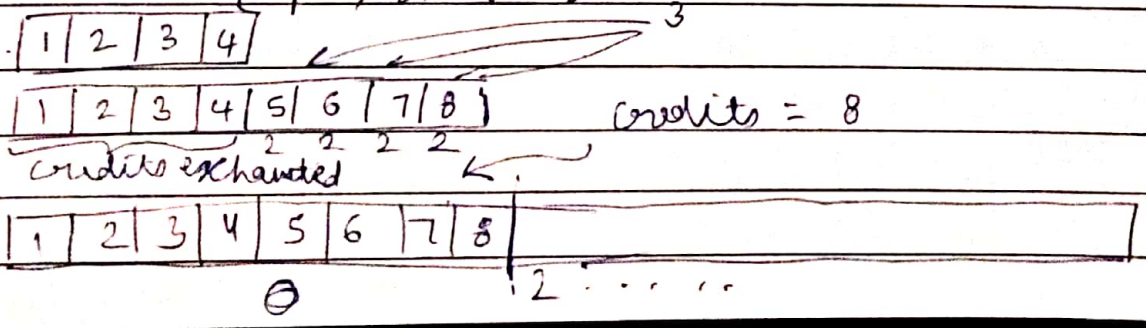
Credit



#Array Doubling

By assigning the cost of any insertion $\hat{C}_i = 3$ units ($3n$).

C_i { i, if i-1 is an exact power of 2; 1, otherwise



Binary Counter

Maintain a BC under 'n' increment operations.

Assume initial value of counter = 0

an array A of bits having value 0 or 1.

A[0] is the LSB $\quad x = \sum_{i \geq 0} A[i] \times 2^i$

A[0] = A[0] + 1

i = 0

while A[i] = 2 do

A[i+1] = A[i+1] + 1

A[i] = 0

i = i + 1

0)	0 0 0 0	
1)	0 0 0 1	1
2)	0 0 1 0	2
3)	0 0 1 1	1
4)	0 1 0 0	3
5)	0 1 0 1	1
6)	0 1 1 0	2
7)	0 1 1 1	1
8)	1 0 0 0	4
9)	1 0 0 1	1
10)	1 0 1 0	2
11)	1 0 1 1	1
12)	1 1 0 0	3
13)	1 1 0 1	1
14)	1 1 1 0	2
15)	1 1 1 1	1
16)	1 0 0 0 0	5

0 0 0 0 +

0 0 0 1 +1

0 0 1 0

Worst case = $(\log n)\, n$

$n + n/2 + n/4 + \cdots$

$n\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots \frac{1}{2^{\log n}}\right)$

$n \; \frac{1\left(1 - \frac{1}{2}\log n\right)}{1 - \frac{1}{2}}$

$2n$

Amortised cost per increment $= \frac{2n}{n}$

$= n$

16.01.2024
Tuesday

Accounting Method

	$\hat{C}_i$	C_i
Cost (Bit flip from 1 to 0) =	0	1
Cost (Bit flip from 0 to 1) =	2	

0000

0001

Potential Method

Potential Energy of a DS.

Initial State of DS = D_0

At the i^{th} separation state of DS = $D_{(i-1)}$ to D_i

C_i is the cost incurred for i^{th} separation.

$\phi\{D_i\} \Rightarrow R$

$\phi\{D_0\} = 0 \qquad \phi\{D_i\} \geq 0$

$\hat{C}_i = C_i + \Delta\phi$

Amortized cost = Actual cost + change in potential $(\phi D_i - \phi D_{(i-1)})$

Amortized cost is more than Actual cost then potential of DS ↑ else

$$\sum_{i=1}^{n} \hat{C}_i = \sum_{i=1}^{n} (C_i + (\phi D_i - \phi D_{(i-1)}))$$

$$= \sum_{i=1}^{n} C_i + \phi D_n - \phi D_0$$

We want Amortized cost is more than Actual cost (upper bound) then $\phi D_n - \phi D_0 \geq 0$

$\Rightarrow \phi D_n \geq \phi D_0$

$\Rightarrow \phi D_n \geq 0$

Stack with multipop

$\phi(D_0) = 0$

$\phi(D_i) \geq 0$

Potential function = No. of elements in the stack

$\hat{C}_i$ push $= C_i + \Delta\phi$

$= 1 + (1)$ ← 1 Element in Stack

$= 2 = O(1)$

$\hat{C}_i$ pop $= C_i + \Delta\phi$

$= 1 + (-1) = 0 = O(1)$

$\hat{C}_i$ multipop $= C_i + \Delta\phi$

$= k + (-k) = 0 = O(1)$

Binary counter (BC)

0 0 0 0 $\quad D_0 = 0$

$D_i \geq 0$ for all values

Potential function (PF) = No. of 1's in BC

$\hat{C}_i = C_i + \Delta\phi$

$= 1 + 1 = 2$

$\hat{C}_i = C_i + \Delta\phi$

$= k + 1 - (k-1)$

$= 2 = O(1)$

"str" 0 1...1 (k one)

↓

"str" 1 00..00 (k zeroes)

$\Delta\phi = t + 1 - t + k$

$\Delta\phi = 1 - k$

10 1 1 1 0

10 1 1 1 1

1 0 0 1 1 1

1 0 1 0 0 0

* No. of flips = position of 1st zero

Potential Method to Table Doubling :-

Amortized cost per insertion

17.01.2024 Wednesday

$\phi(D_i) = 2i - 2^{\lceil \log i \rceil}$ (Assume $2^{\lceil \log 0 \rceil} =$)

Potential Function

$\hat{C}_i = C_i + \Delta\phi$

$= C_i + (2i - 2^{\lceil \log i \rceil}) - (2(i-1) - 2^{\lceil \log(i-1) \rceil})$

$\begin{cases} i & \text{when } i-1 \text{ is an exact power of 2} \\ 1 & \text{for all other cases} \end{cases}$

Case 1 Let $i-1$ is exact power of 2

$\Rightarrow \hat{C}_i = i + 2 - 2^{\lceil \log i \rceil} + 2^{\lceil \log(i-1) \rceil}$

$= i + 2 + (i-1) - 2(i-1)$

$\Rightarrow \hat{C}_i = 2 - 1 + 2 = 3$

Case 2 Let $i-1$ is not the exact power of 2

$\hat{C}_i = 1 + (2i - 2^{\lceil \log i \rceil}) - (2(i-1) - 2^{\lceil \log(i-1) \rceil})$

$= 1 + 2 + 2^{\lceil \log(i-1) \rceil} - 2^{\lceil \log i \rceil}$

$\Rightarrow \hat{C}_i = 3$

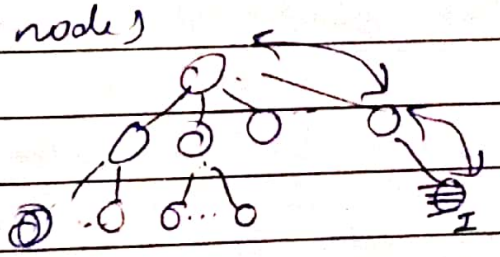
Heaps

- Binary Heaps — Priority Queue Operations
  - Find_min — $O(1)$
  - Delete_min — $O(\log(n))$
  - Insert — $O(\log(n))$
  - Create — $O(n)$ Amortized

Generalisation of Binary Heaps :-

d-Heap (d children of each node)

- Find_min — $O(1)$
- Delete_min — $d\ O(\log_d n)$ ~~$O(\log n)$~~ $O(d \log_d n)$
- Insert — $O(\log_d n)$



→ meld

Merging of 2 Binary heaps

n_1 , n_2

$(n_1+n_2)\log(n_1+n_2)$

↳ Top Down of all elements

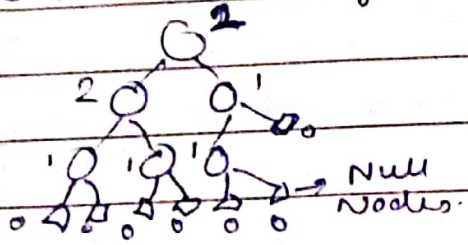
$\min(n_1, n_2)\log(n_1+n_2)$

↳ Bigger heap put smaller heap elements.

Heaps

Leftist Trees - Height biased & Weight Balanced leftist Trees
(HBLT) (WBLT)

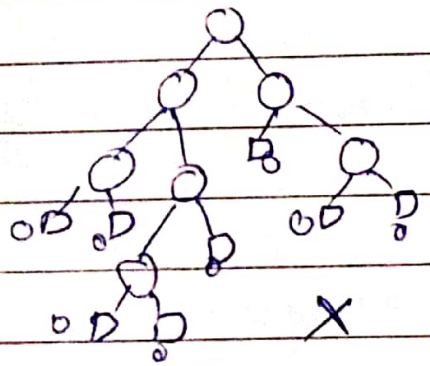
Extended binary tree

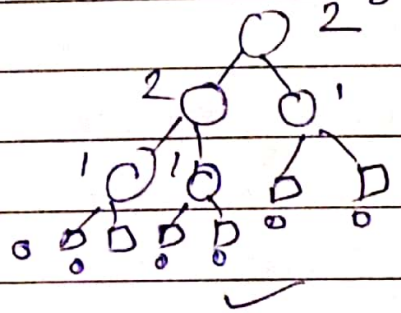


x in BT

$S(x)$ - min. dist. from leaf node

Binary Tree is a HBLT iff the S-value of left child is ≥ S-value of Right child at every internal node





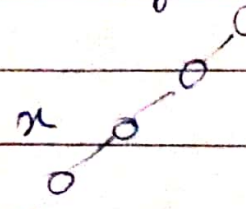
X

Observation -

① Left part is heavier.

② ~~For~~ Right most part would be shortest

① No. of nodes in subtree with root x is at least $2^{S(x)}-1$ → (I)



(m)

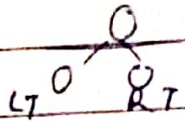
min. dist. from root to ext node = S(x)

∴ All branches should have x upto atleast

$m \geq 2^{S(x)} - 1$ → (I)

Find-min — O(1) — root element

Delete-min — HBLT structure should be restored again



Merge LT & RT

Insert — New HBLT with single node is created

merge it.

Merging (Melding)

2 HBLTs : A & B $n(A) \neq 0$ $n(B) \neq 0$

root(x) : root of tree x $= r(x)$

C : HBLT after melding A & B

$n(x)$ = No. of nodes in x

(i) $r(C) = \min(r(A), r(B))$

LC(x) = ~~Left~~ Left child of node x

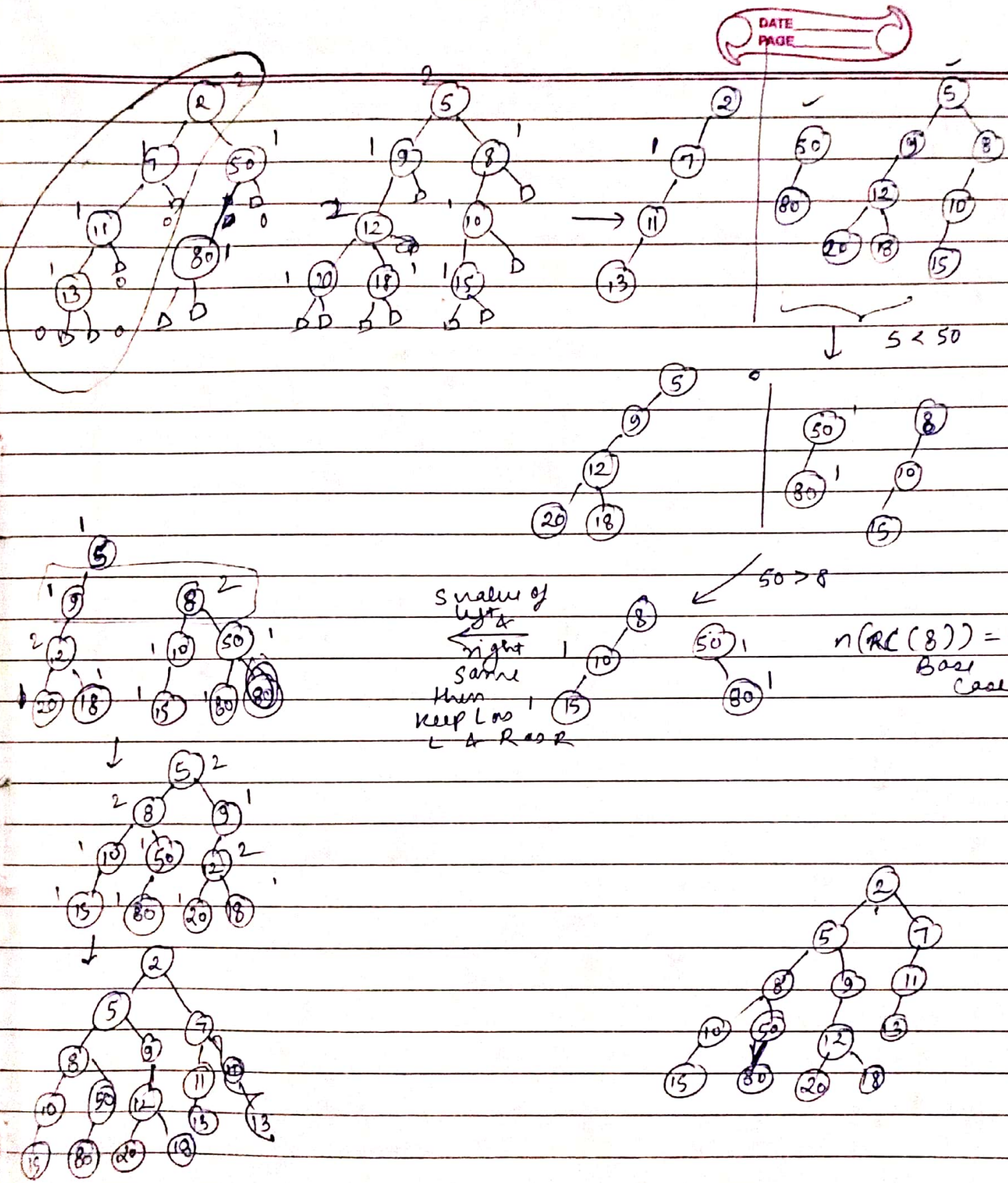
RC(x) = Right child of node x

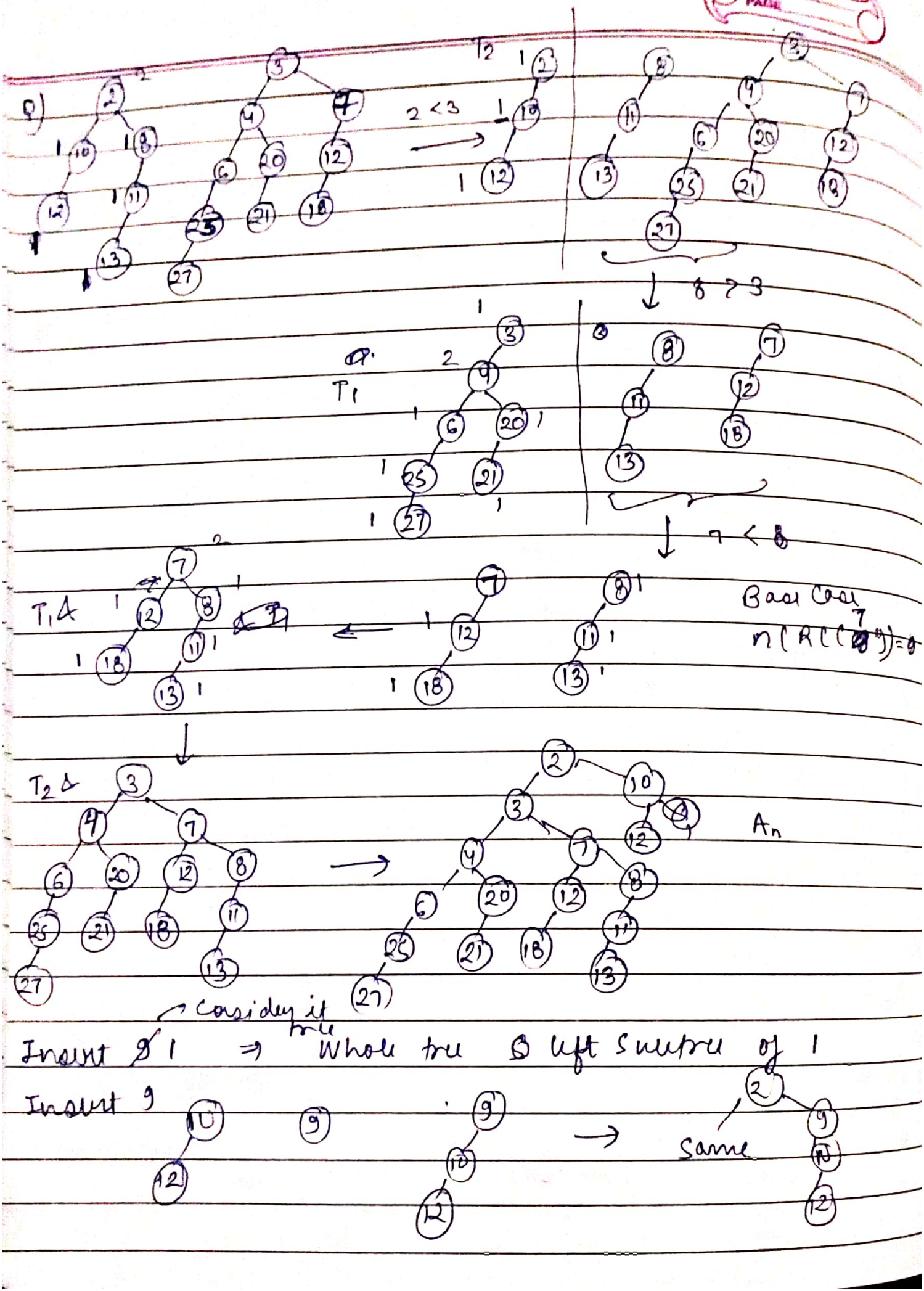
(ii) C : $r(C) + LC(r(C))$

Meld (RC(r(C)), mian (r(A), r(B)))

Repeat this recursively

(iii) $n(X_i) = 0$ at any step, other becomes final boolean





Delete_min(An) LT(An) RT(An) → Meld(LT(An), RT(An))

TC(melding) = $\log(n_1+1) + \log(n_2+1)$

$\leq 2\log(n_m+1)$ where $n_m = \max(n_1, n_2)$

⇒ $\boxed{TC(Melding) = \log(n_m)}$

Creating a new HBLT

9 2 7 11 15

M.1. $O(n \log n)$

continuous merging is done

M.2.

9	2	7	11	15	...

Queue

~~Enqueue 2 & deq~~

pop 2 & merge & push(meld(2 trees))

DO this continuously until only 1 size of Q. That is HBLT

23.01.2024 Tuesday

BINOMIAL HEAP

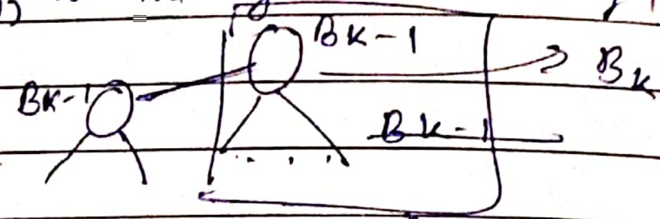
↓

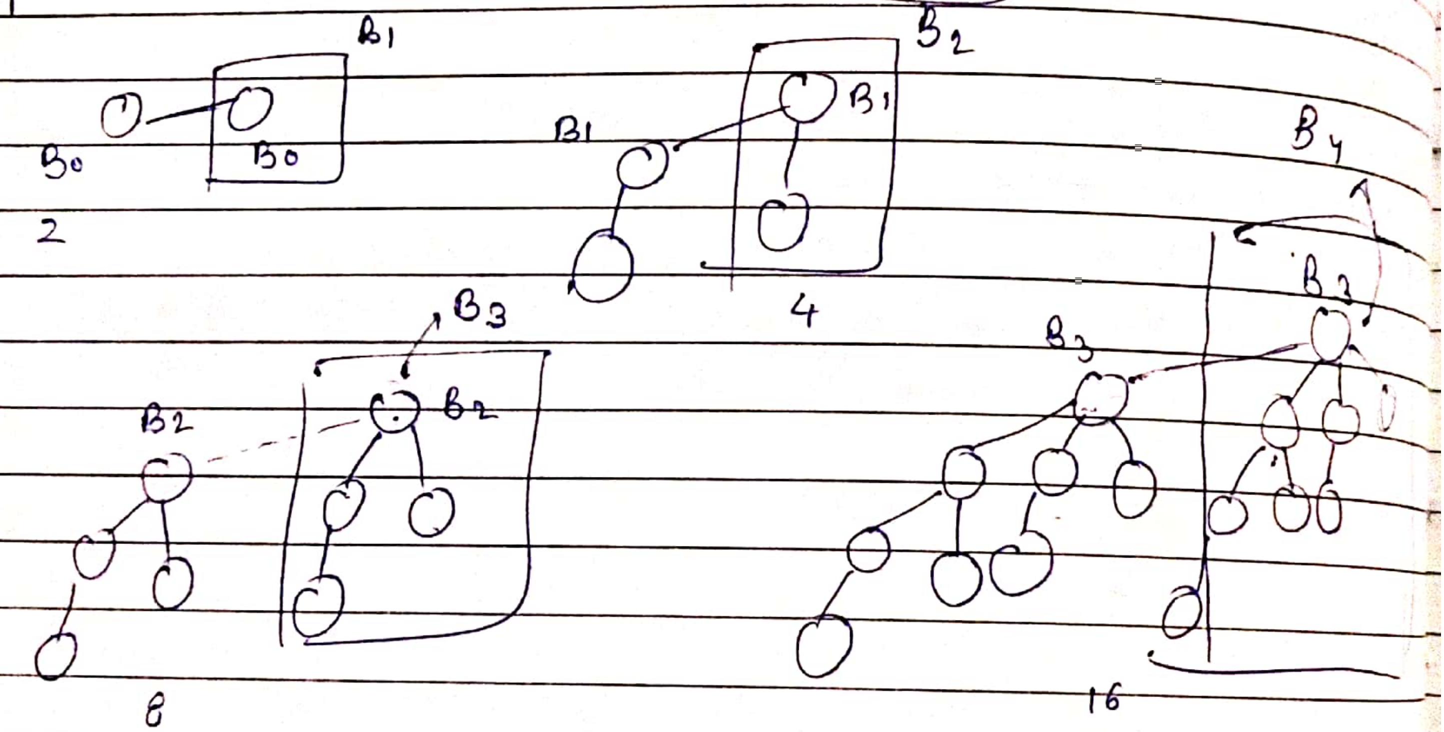
Binomial Tree — B_k

ordering: Child follows same order. ex: Binary tree

B_0 : Single Node tree O B_0

B_k : consists of 2 B_{k-1} Binomial trees such that root of one $B(k-1)$ is the leftmost child of the root of the other $B(k-1)$





यह है कठिनाई। B_k: A tree with k degree

Properties:

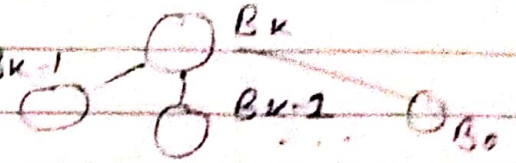
1. Binomial Tree B_k $\forall k$ B_k has 2^k nodes $\Rightarrow n(B_k)$
2. Height of B_k is k $\Rightarrow$ $h(B_k) = k$
3. No. of nodes at depth i in a B_k is $\binom{k}{i}$ Binomial coeff.

$$\frac{k!}{i!(k-i)!}$$

$^nC_r + {}^nC_{r-1} = {}^{n+1}C_r$

(4) Root has degree k which is greater than any node

(5) If children are numbered from left to right then each node is the root of $B_{k-1}, B_{k-2}, \ldots, B_0$ (in order)



[Use Induction ~~for~~ to prove]

(1) $B_0 = 2^0$ nodes $= 1$ node

$\vdots$

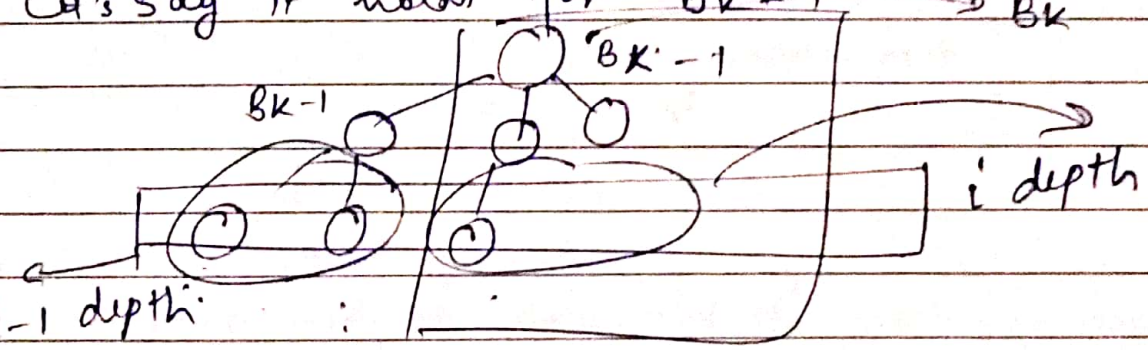
$B_{k-1} = 2^{k-1}$ nodes — say it holds

Then $B_k = 2^{k-1} + 2^{k-1} = 2^k$

(2) Same for height

(3) Let $D(k, i)$ - is the no. of nodes in a B_k at depth i

(4) say it holds for $B_{k-1} \rightarrow B_k$



$\therefore D(k, i) = D(k-1, i) + D(k-1, i-1)$

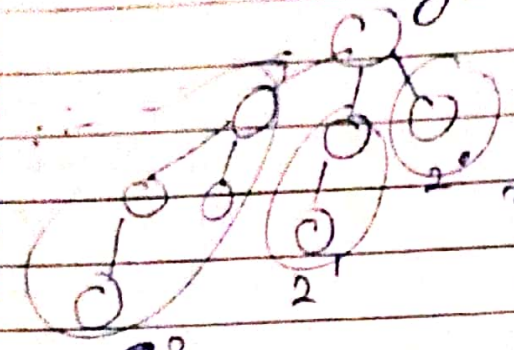
$= {}^{k-1}C_i + {}^{k-1}C_{i-1}$

$= {}^kC_i$

$\frac{(k-1)!}{i!(k-1-i)!} + \frac{(k-1)!}{(k-1-i+1)!(i-1)!} = \frac{(k-1)!}{(i-1)!(k-1-i)!}$

$\frac{k!}{i!(k-i)!} = {}^kC_i$ ⟵ $= \frac{k}{i(k-1-i+1)} \left\{ \frac{i}{i} + \frac{k-1-i+1}{k-1-i+1} \right\} \left(\frac{(k-1)!}{(i-1)!(k-i)!} \right)$

23. ~~binomial~~
B ∴ For a tree with n nodes the degree is $\log_2(n)$



$1 + 2^0 + 2^1 + 2^2 + \dots 2^{k-1}$
root ↓ child nodes

$= 1 + 2^k - 1$

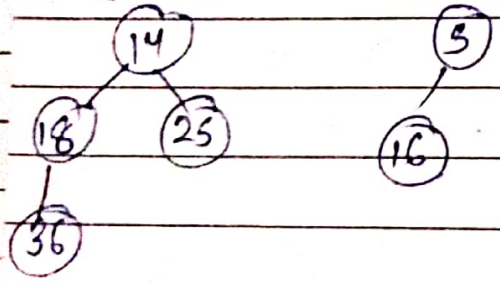
$= 2^k$

$\therefore 2^k = n$

Analogy of Binary Counter ⇒ $\boxed{k = \log_2 n}$

Binomial Heap is a set of Binomial tree.
↳ ordering property should be fulfilled
↳ min / max heap order
∀ ~~all~~ trees

Min-Heap



Two B_is can't be in Binary heaps
Combine both to form B_{i+1}.

~~There is a~~
∀ BH, ∃ unique k value
∴ it would be only one

Binomial ~~Binary~~ tree

Q) Binomial heap of ~~18~~ 13 nodes has which BT

Sol: Get Binary equivalent of no. of nodes.

B_3, B_2 & B_0 - BT

2 | 13
2 | 6 +
2 | 3 0
2 | 1 1
0 +

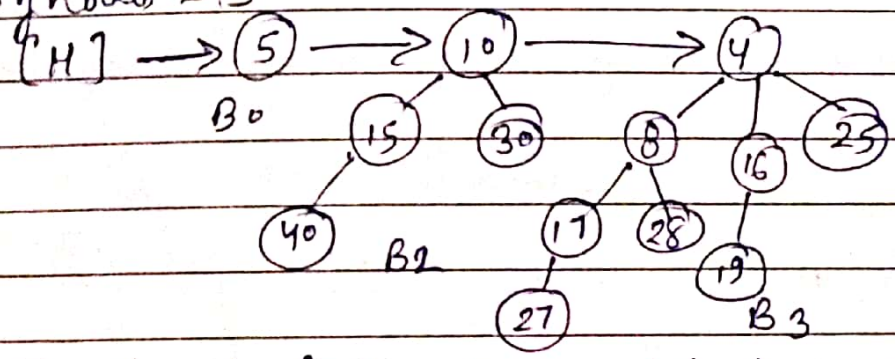
1 1 0 1
B_3 B_2 B_1 B_0

For Binomial heap of n nodes max^m Binomial subtrees = $\lfloor \log_2(n) \rfloor$

↳ n nodes → Binary (n)

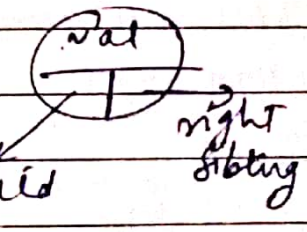
Linked list is used to construct Heaps.

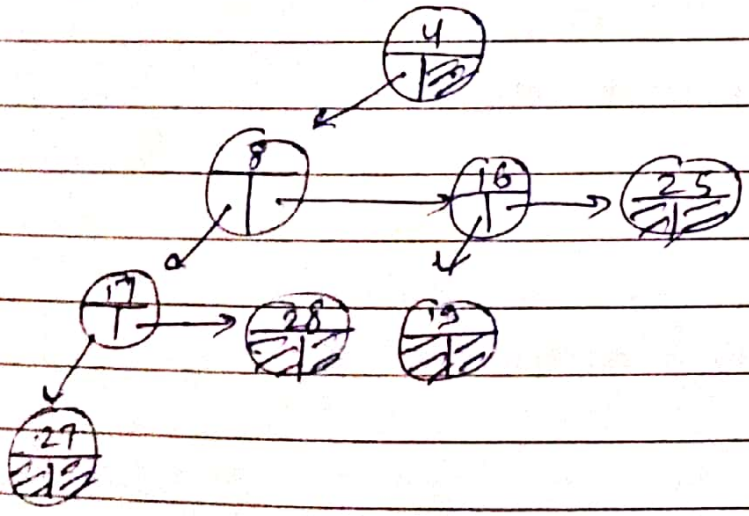
no. of nodes = 13 : 13 nodes Binomial heap (min. ordered)



In linked list Binomial trees are attached in increasing order of degrees.

Left Child Right Sibling Representation





Space Efficient Approach

Use this approach to store binomial trees.

24.01.2023 Wednesday

Assume:

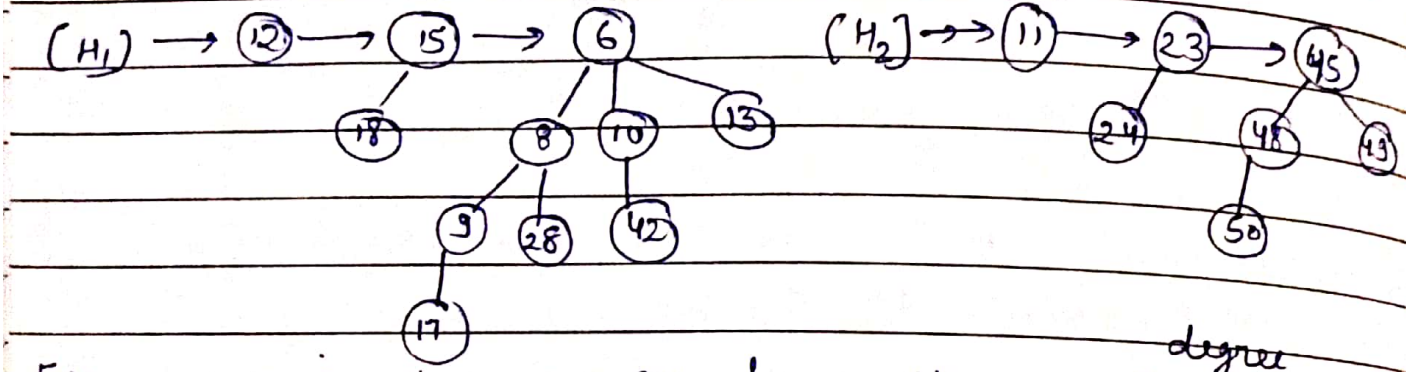
① CREATE ;
② Insert (H, x)
③ Minimum (H)
④ Delete-min (H)
⑤ Union (H_1, H_2)
⑥ Decrease-key (H, x, k)
⑦ Delete-key (H, x)

MakeHead [ ] O(1) constant

→ pointer to the key values we want to perform

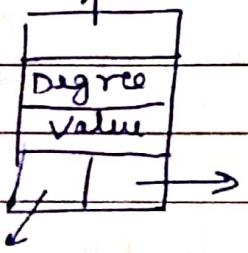
- Find-min O($\log_2(n)$)
  ↳ Find min. in root list

n Nodes Then there are atmost "$\log_2(n)+1$" Binary trees



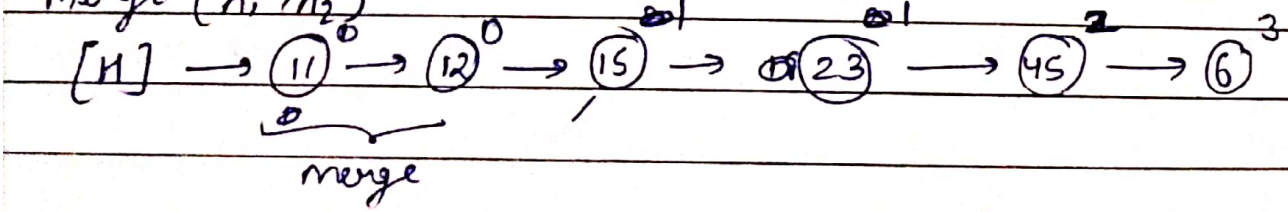
[At any given time we can have atmost 3 same degree trees]

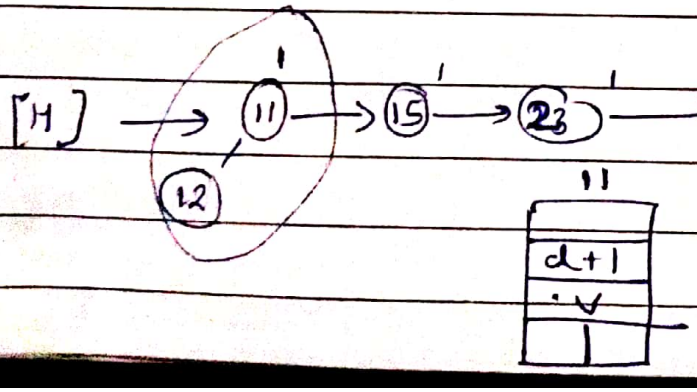
Node Struct



Merging is O(1) time only link changes

Merge (H_1, H_2)





Fibonacci Heap → O(1).

Heaps

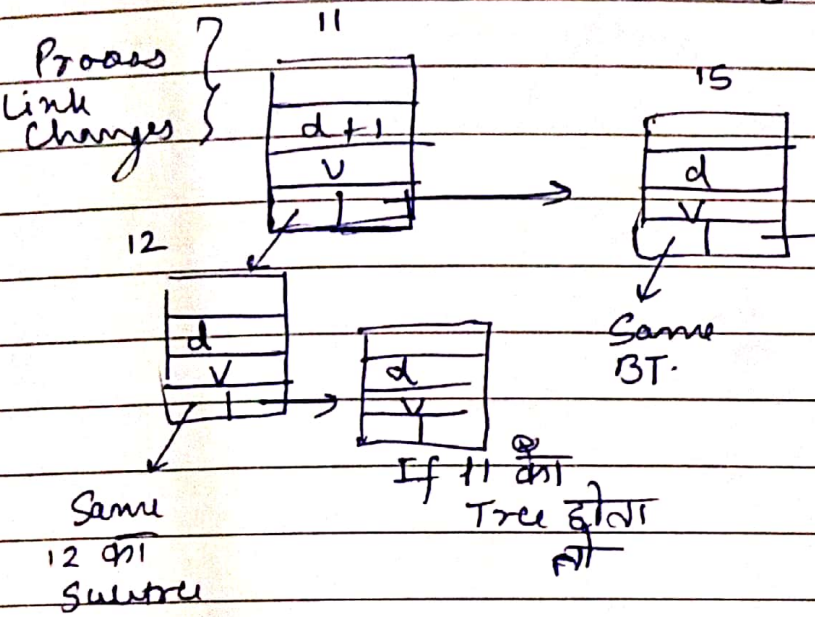
n_1 n_2 Nodes

$\log_2 n_1$ $\log_2 n_2$ BT

Maxm no. of merging $= 2\log_2(\max(n_1, n_2)) + 2$

$= 2\log_2 n + 2$

$= O(\log_2 n)$



* If three trees are of same degree then keep the first same & merge next 2.

② Insert → Create it as new BH & merge → $O(\log_2 n)$

③ Delete min

① Find-min → $O(\log_2 n)$

② Delete min → $O(1)$

③ Reverse child list → $O(\log(n))$

④ merge (Reverse Child list, Original) → $O(\log(n))$

Total :- $O(\log_2(n))$

④ Find-min → $O(\log_2 n)$

Search in root list.

⑤ Decrease-key (H, 28, 7) $O(\log_2(n))$

Heapify type.

Delete (H, 17)

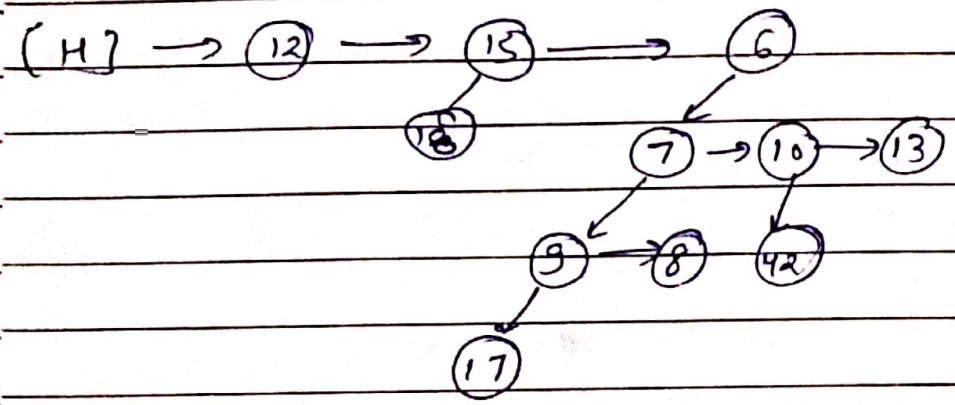
① Decrease_key (H, 17, -∞) $O(\log_2(n))$

// Root पर आ जायेगा।

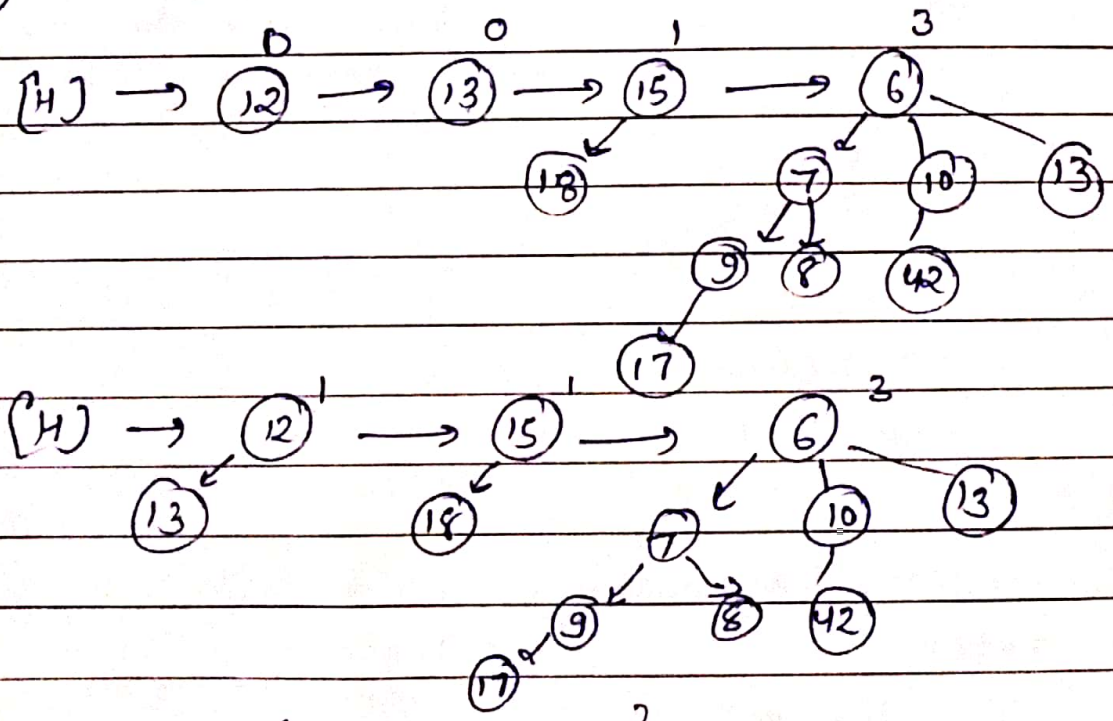
② Delete_min (H) $O(\log_2(n))$

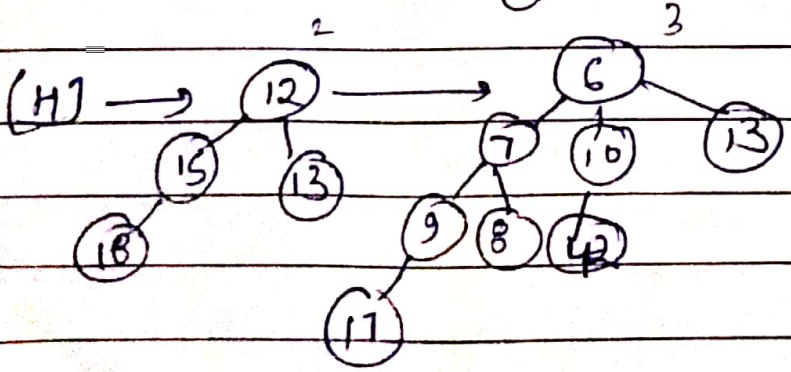
$O(\log_2(n))$

$O(\log_2(n))$



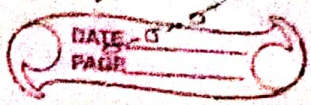
Q. Insert (H, 13)





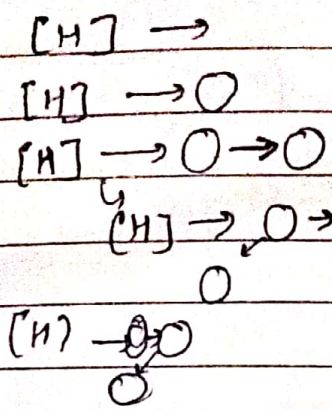
Ans.

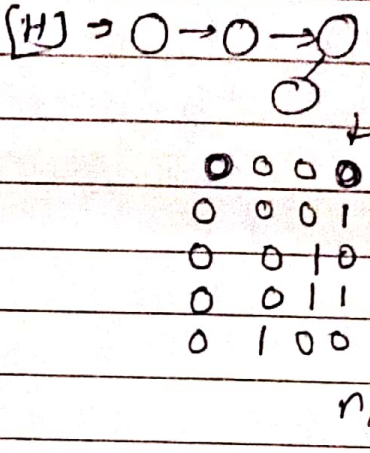
Right Skew Trees → Only right child of each
Left Skew Trees → Only left child of each



Q) Explain the amortised TC in of a series of "n" insert operations in an initially empty Binomial heap.

Sol:





n/2

AVL Trees → Self balanced BST
Insertions & Deletions ⇒ several rotations
To decrease this we use RBT
Searching more operations → AVL tree

RED BLACK TREES

(AVL is more balanced)

What are Binary search Trees → 1 bit to store colour
Why BST? (i) Inorder traversal sorted.
(ii) search, ~~insert~~

What is the TC of diff. operations ? → O(h)
$h \in \log_2 n$

O(n) → Skewed tree "Worst case"
$O(\log_2 n)$ → Complete binary tree "Best Case"

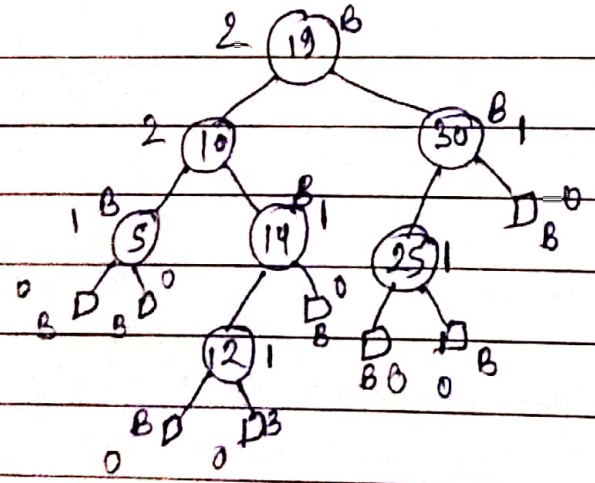
Height Balanced BST → $O(\log_2 n)$ Balance factor = {-1, 0, 1}

Red Black Trees : Extended BST (Null Pointer)
Balanced BSTs

Colouring on the nodes (1 bit to store in each node)
Any path is more than twice the shortest path

RULES :-

- Root is always black.
- All leaf nodes are black. (NULL)
- No 2 consecutive red nodes on any simple path.
- from an internal node x to leaf node.



- Black height Bh[n] remain same over all paths

Same → (Red Node ⇒ child both black)

- No. of black nodes from all nodes to leaf

Complete Binary tree :- Each node divides tree in half subparts.

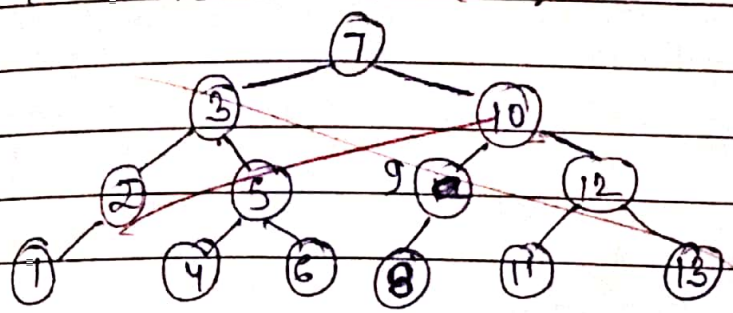
Insertion & Deletion are more in RBT than AVL

Black height $Bh[x]$ = No. of black nodes from x to leaf node not considering colour of x

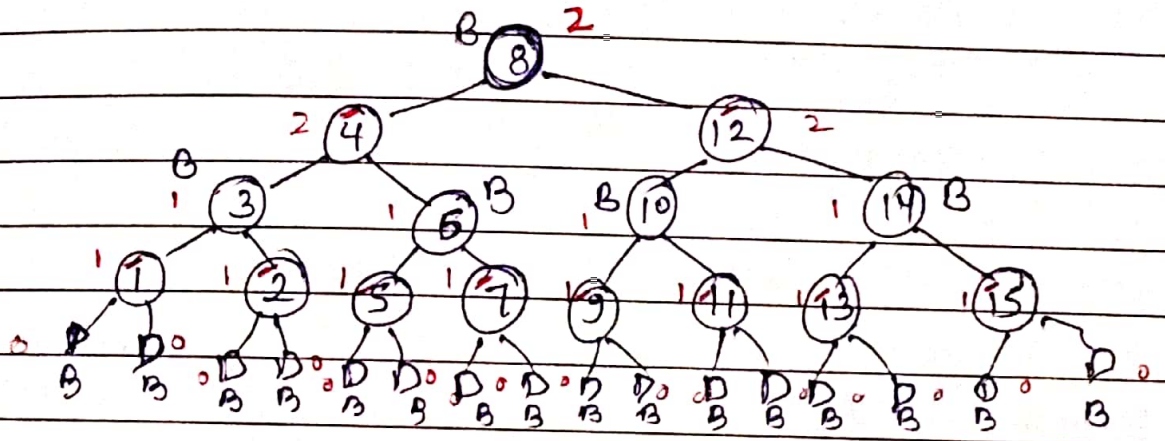
Violation of Black height then we need to do balance

Q: 1 to 15 $Bh(h) = 2$ Red Black Tree

Sol:



9 10 11 12 13 14 15



Red Black Trees: Are good search trees.

A R.B Tree with n internal nodes has height at most $2 \log(n+1)$.

We show that a subtree rooted at 'x' has at least $2^{bh[x]} - 1$ internal nodes ↳ 'x' is any internal node

Proof: By Induction

No. of internal nodes in all leaf nodes $= 2^{bh(x)} - 1$

$= 2^0 - 1 = 0$

If black child, No. of —u— $= 2^{bh[x-1]} - 1$

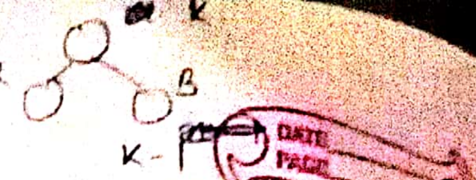
If red child, no. of —u— $= 2^{bh[x]} - 1$

Children = {BB, BR, RB, RR} (minimum no. of internal nodes)

Worst case

30-01-2024
Tuesday

$bh(x) = k$



No. of internal nodes in a tree rooted at x $= (2^{bh(x-1)} - 1) + (2^{bh(x-1)} - 1) + 1$

$= (2^{bh(x-1)} \cdot 2) - 2 + 1$

$= 2^{bh(x)} - 1$

Hence proved.

min. no. of internal nodes in x rooted tree.

[Same can be proved by $s(n)$ → previous tree]

No. of internal nodes, $n \geq 2^{bh(x)} - 1$

$\Rightarrow 2^{bh[x]} \leq n+1$

$\Rightarrow 2^{h/2} \leq n+1$

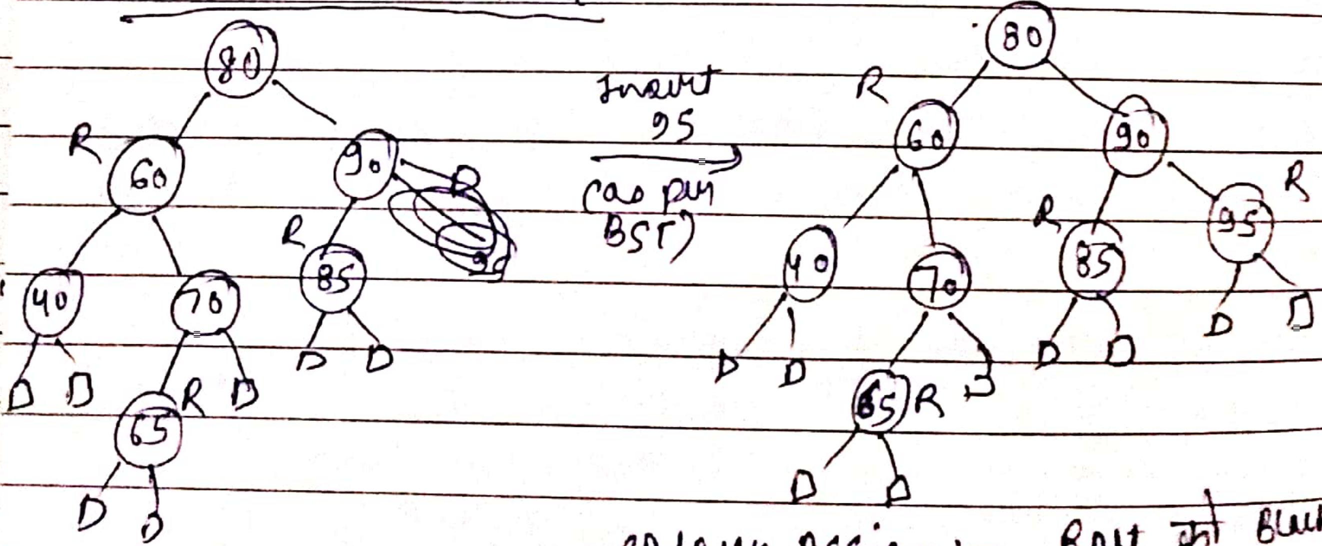
$\Rightarrow \boxed{h \leq 2 \log(n+1)}$

height of tree = alternate red & black node

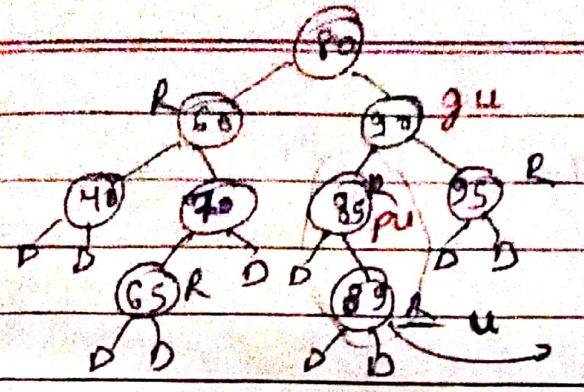
All the operations where tree structure can be performed in O(log n)

↳ BST Height balanced.

Insert in a RB Tree



Colour Assign:- Root को Black
बाकियों को Red
positions पर [illegible] दो।



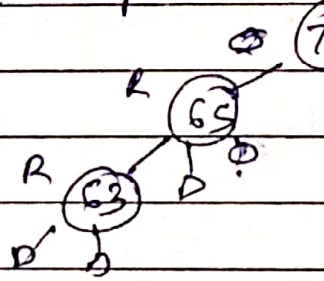
Imbalance has occurs in RBT

Imbalance Removal

↳ Colour change
↳ Rotation

Newly inserted node = u
Parent = pu & Grandparent = gu
Check path, gu → pu → u = LR } Path
Colour of ~~sibling~~ child (gu) = Red } LR_r is colour

Example



Imbalance LL_b

All possible imbalances =
$\{ XY_C : X \in \{L, R\}, Y \in \{L, R\}, C \in \{red, black\}_b \}$

~~$\{ LL_r, LR_r, RL_r$~~

Set-1

LL_r	LR_r
RR_r	RL_r

↳ Colour Change (Apply)

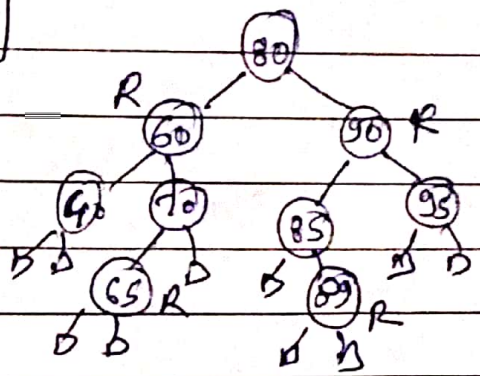
Set-2

LL_b	LR_b
RR_b	RL_b

↳ Rotation (Apply)

double rot^n

Single rot^n

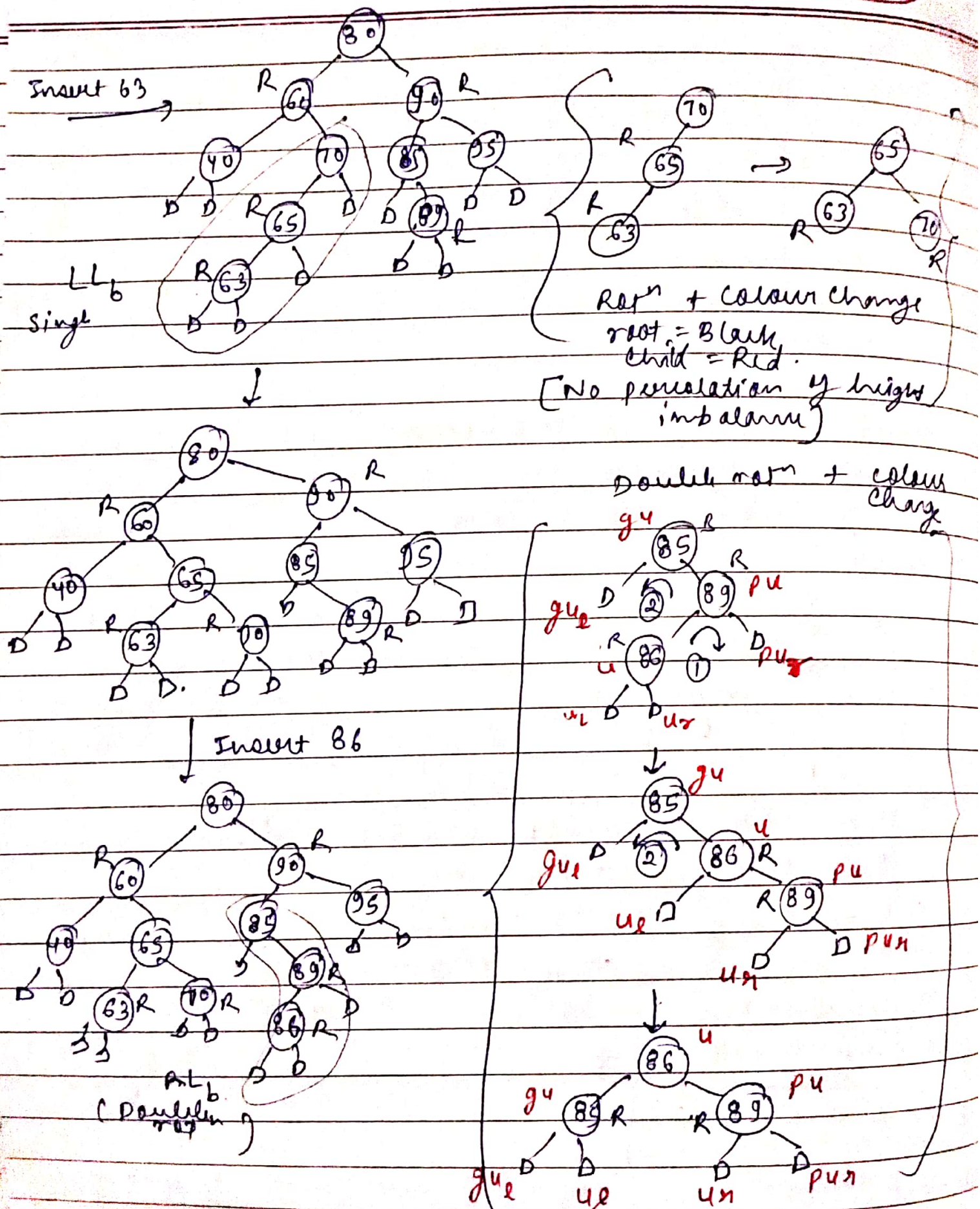


Change color of the gu & its both ~~sibling~~ child
{ pu & sibling (pu) } *

Problem:

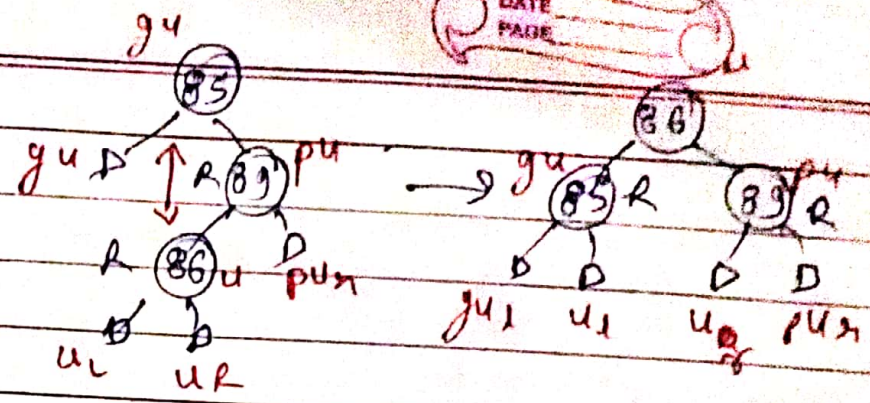
① g root हो तो Height एक से Increase हो जाता है।

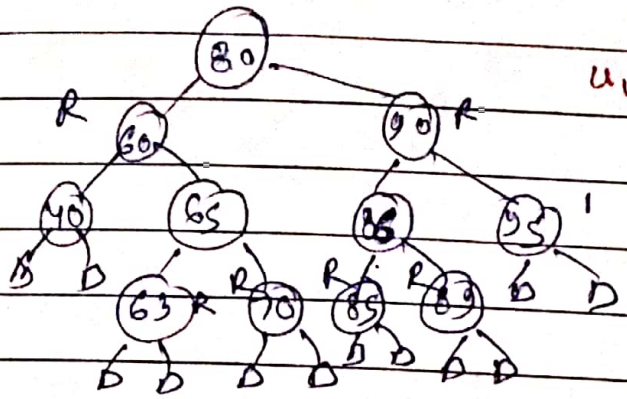
② Imbalance of colour can still occur above.
{ Classify in set 1 or 2 }



Imbalance तब जब दो Red हो।

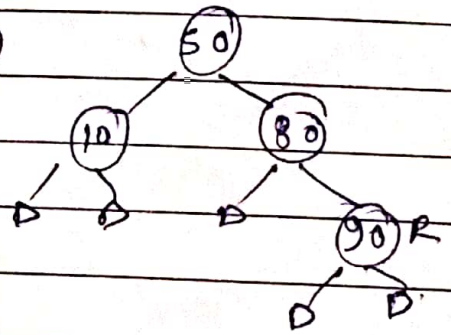
can be done directly



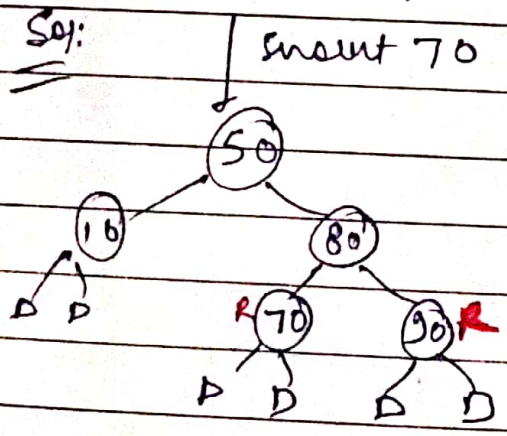


Insert in order:
70, 60, 65, 62, 63, 64

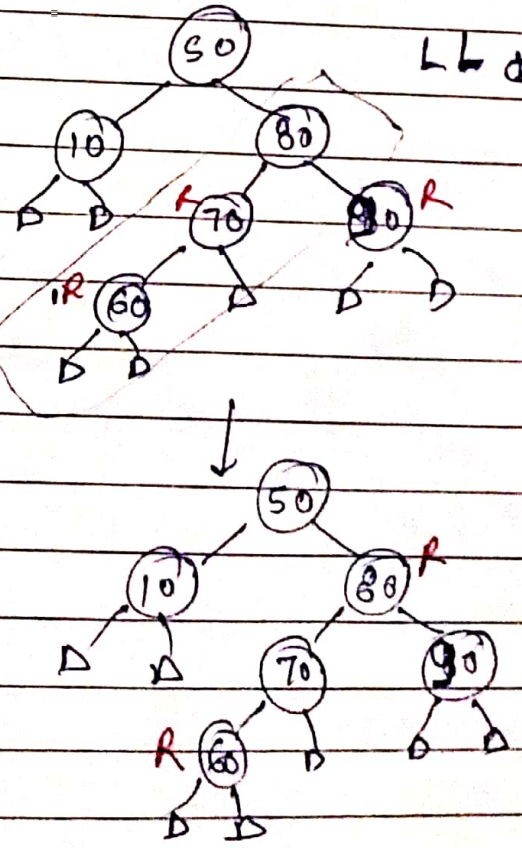
Q)

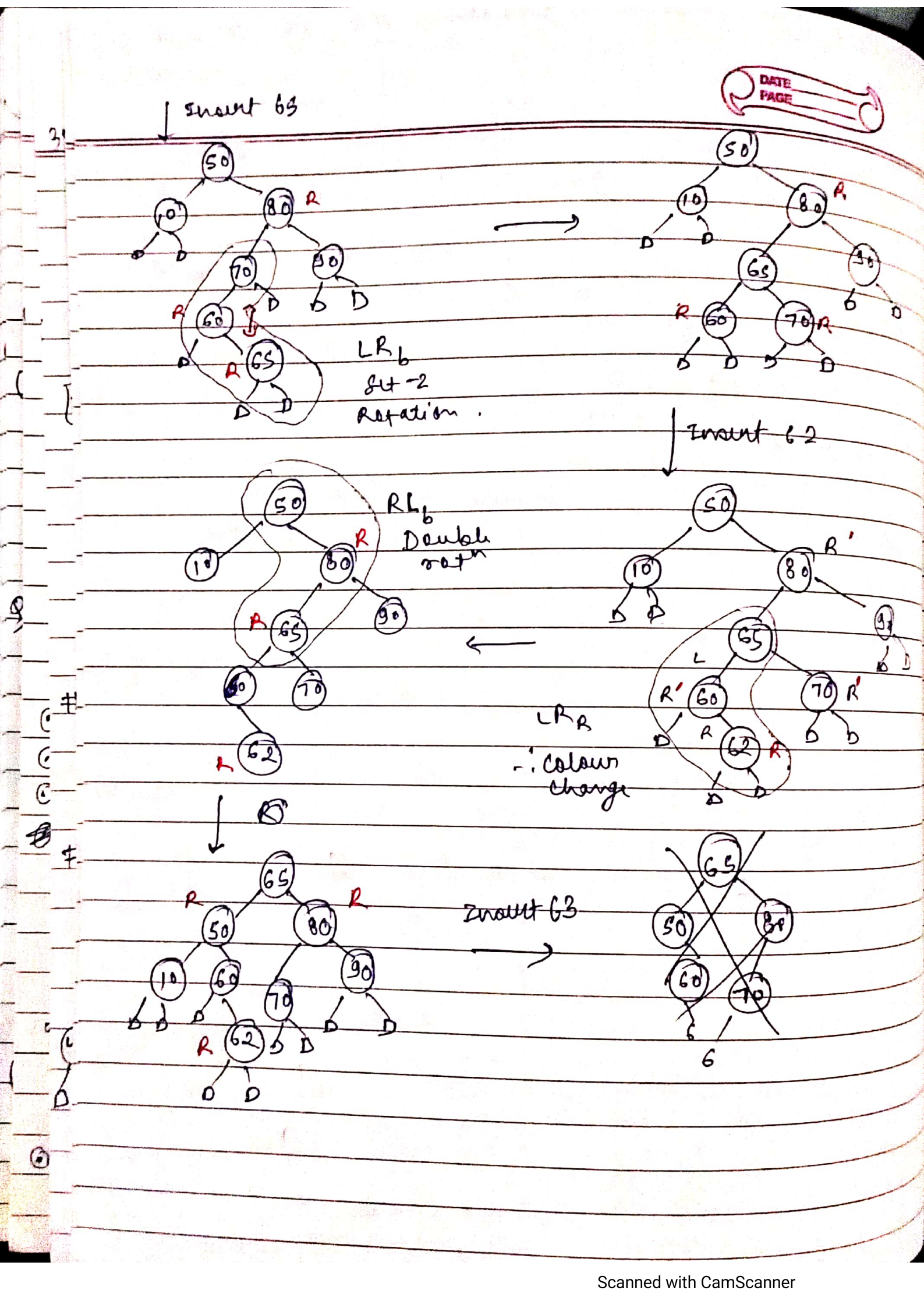


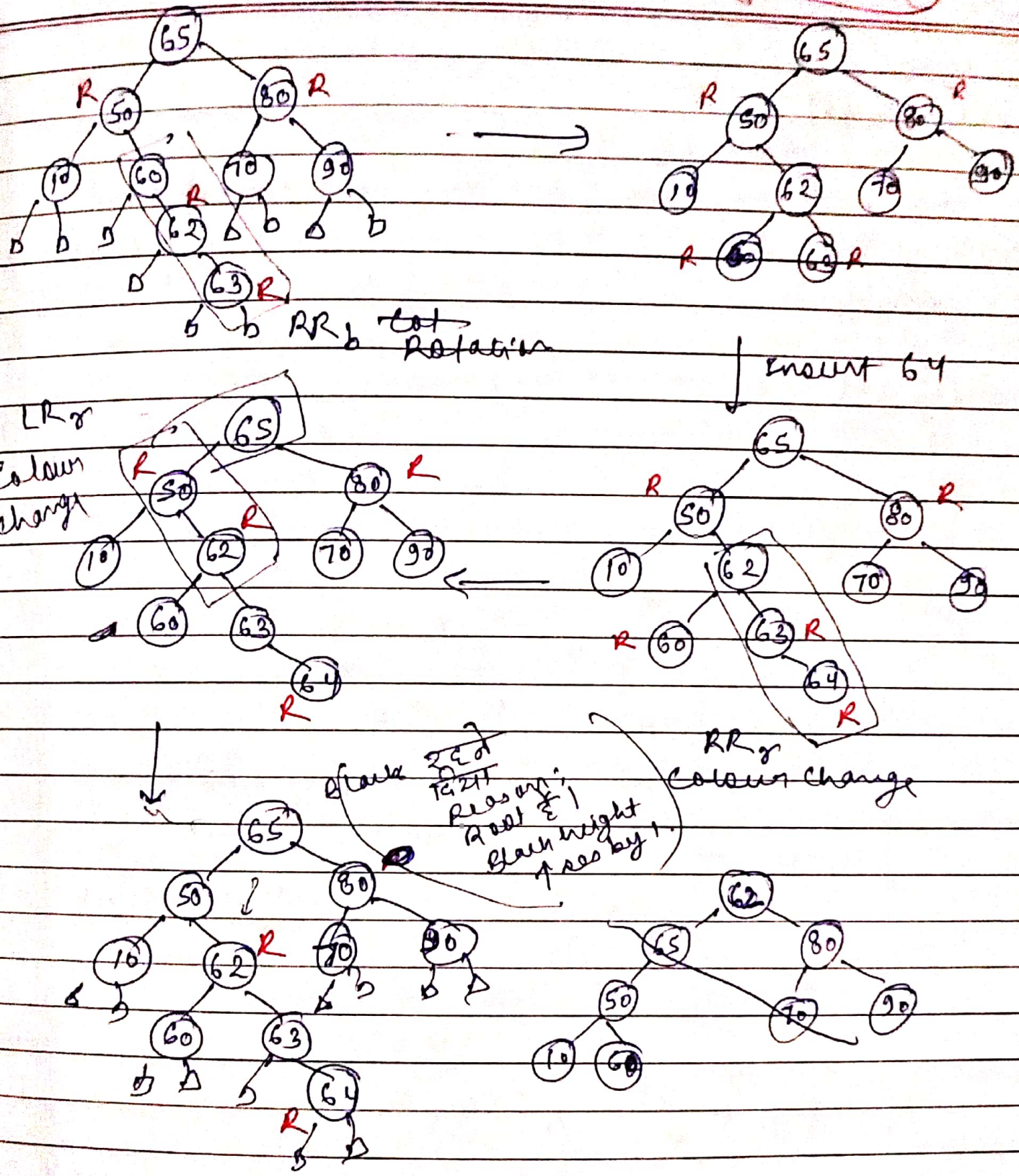
Sol:



Insert 60







31.01.2023
Wednesday

Single child parent can't be red.

DATE
PAGE

Deletion

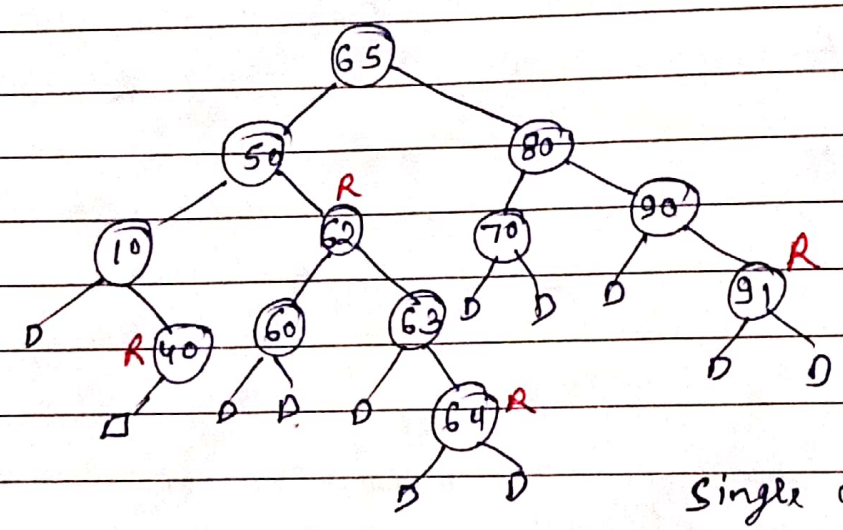
Deletion in BST:

(i) Leaf Node → directly del.

(ii) Node with one child → del. that & append child

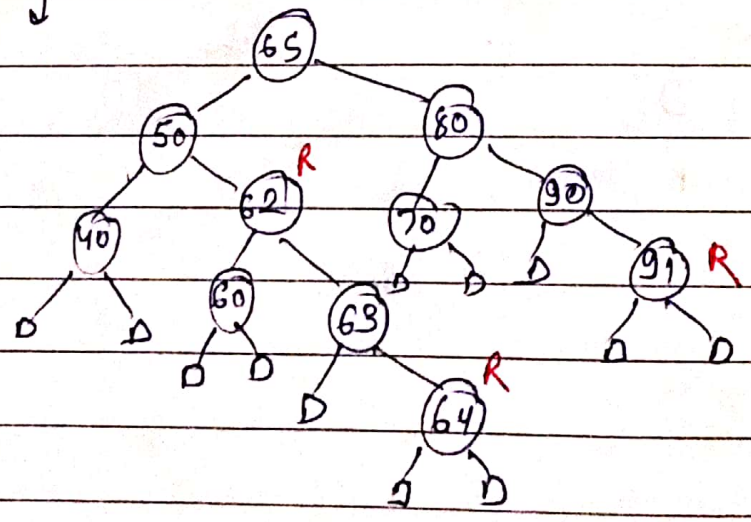
(iii) Node with 2 child → See inorder successor & predecessor & replace & del. using (i) & (ii)

Deleting red node 64 → No problem.



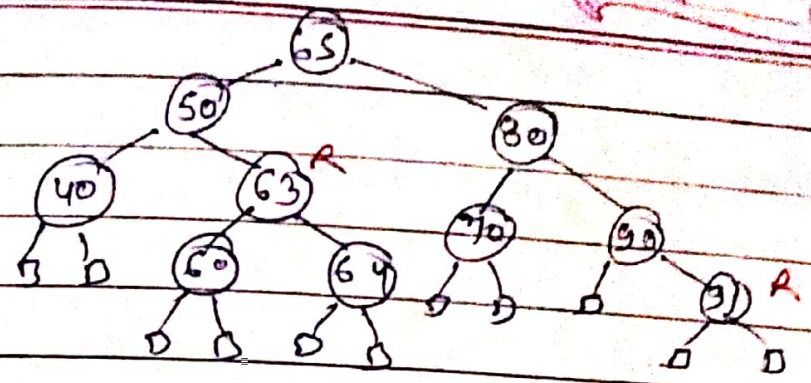
Delete 10

Single child of to be deleted node Replace with child & change its colour to black



Red Node will be Leaf Node or Its both child would be Black

Delete 62 →



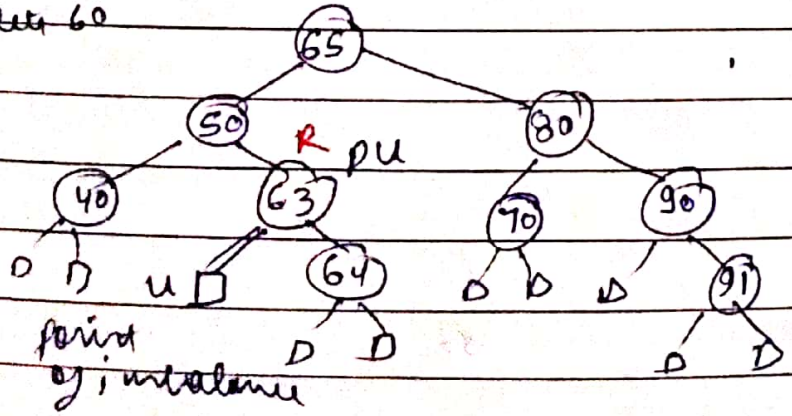
If the node to be deleted is ~~black~~ node or if ~~the~~ it is internal nod.

(1) If node to be deleted is a red leaf - Delete it. Tree remains balanced.

(2) If it is a single child parent, it must be black & its child must be red. Replace with its child & recolor the child black.

(3) If it is an internal node with 2 children
Swap the node to be deleted with its inorder successor
i. if inorder successor is red leaf ~~node~~. Delete it. Done
ii. if inorder successor is ~~blac~~ a single child parent.
do as in case (2)
iii. if inorder successor is black OR node to be deleted is black leaf ⇒ "Imbalance occurs"
[Black height property is violated]

Delete 60



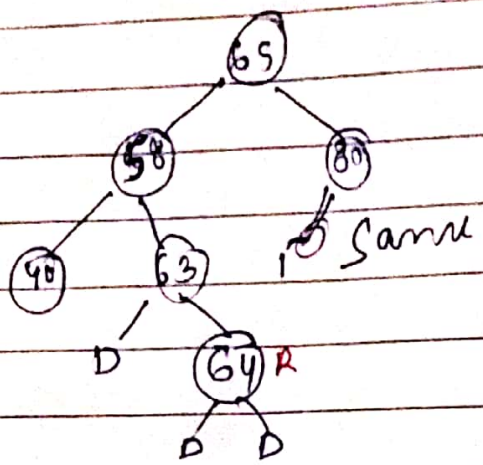
Possible imbalance
(4-PU)
color(sibling) of U
L_b L_R
R_b R_OR

Confirm ⇒ Node Red है तो child path → Black

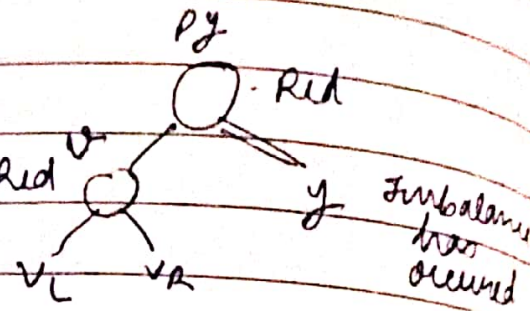
3) Nomenclature

{path (pu - u)} color (sibling(u)) · (no. of red child (pu))

i) Lbao



63 Red हुआ तो change हो गया।
Black ने problem.

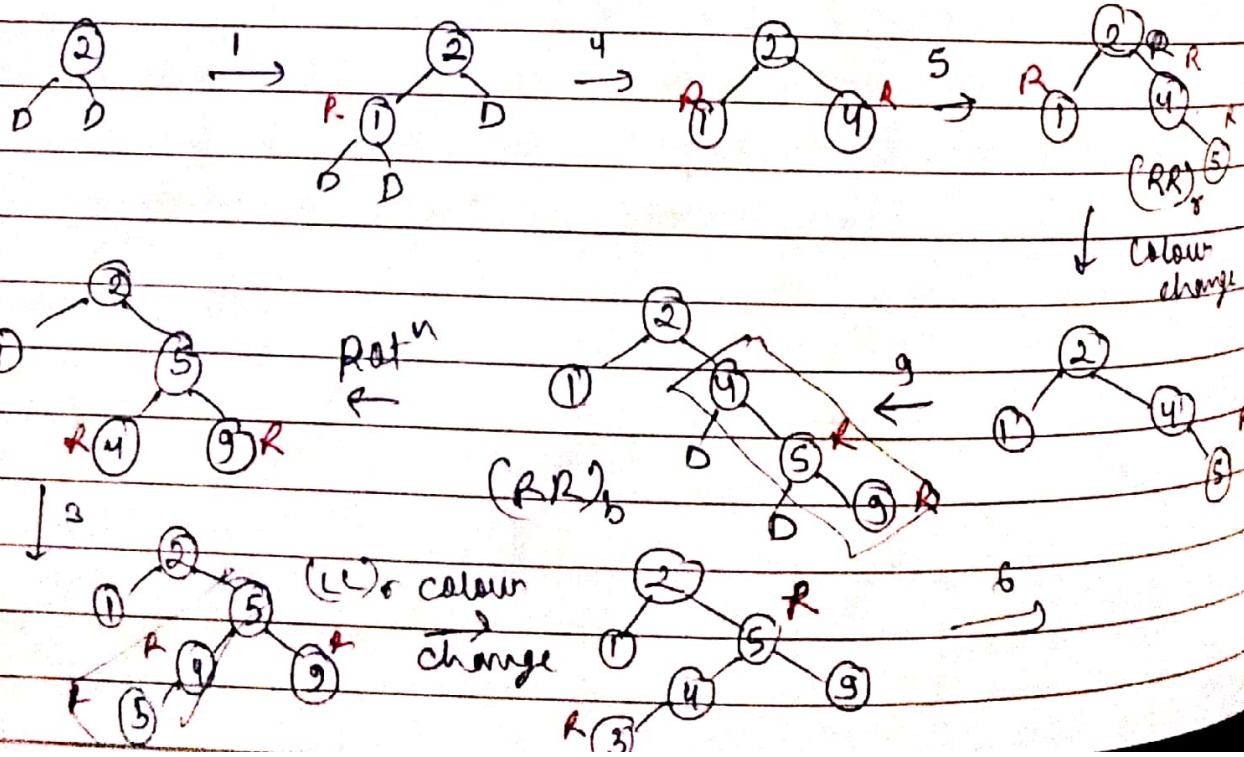


Do colour change y v

4. if Py is Red then no problem

Q) 2, 1, 4, 5, 9, 3, 6, 7 Insert in RBT.

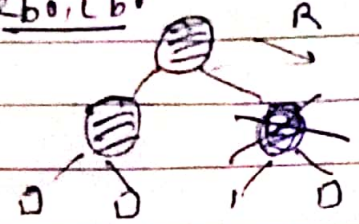
Sol:

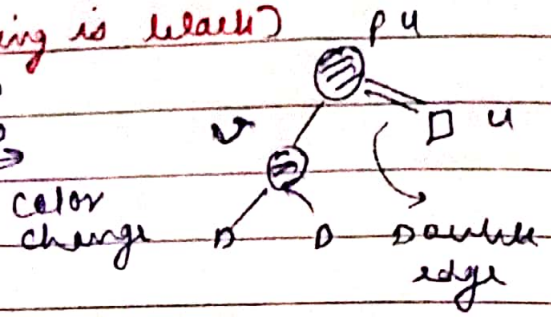


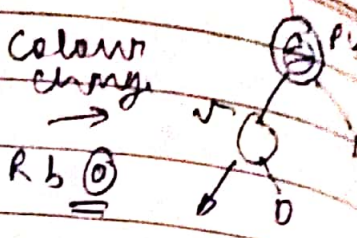
06.02.2024
Tuesday

Deletion :- (Sibling is black)

Rb0, Lb0

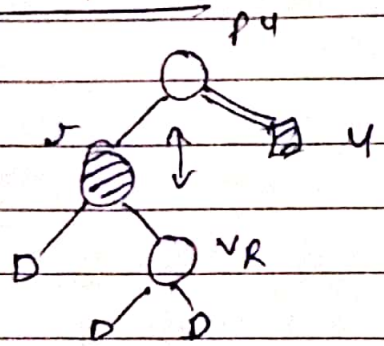




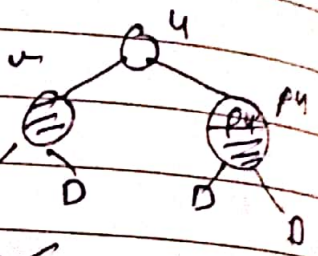


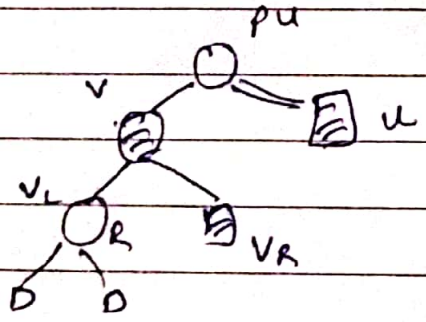
pu = 2 Root to BH ↓ as by 1 else ✓ Percolation.

Rb1, Lb1

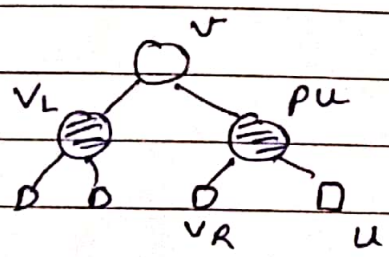


Rb1
Lb1
⟶
Double rotn
VR → color same as pu by one rotn
pu → change to black
~~VR → color son~~

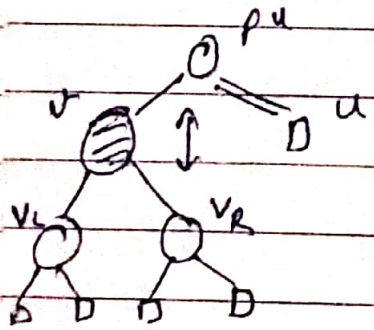




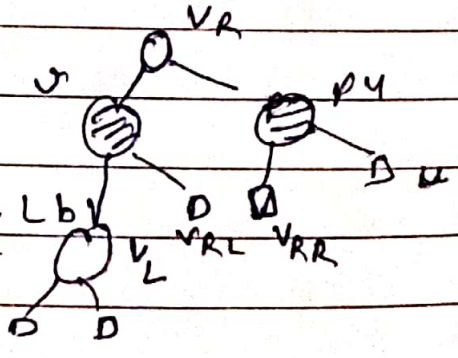
⟶
Single rotn
[Color change same]



Rb2, Lb2



⟶
Same as Rb1, Lb1 but do not change color of other



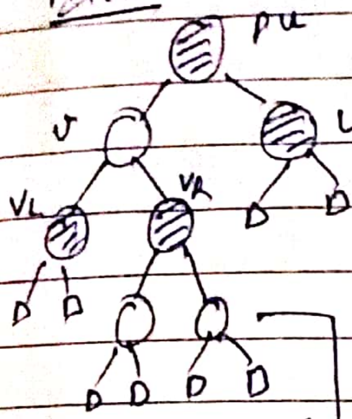
{ In Rb1 & Lb1 change along red child }
Double rotn ← or → Single rotn

V = Sibling(u)

If V == Red:
PU = black (confirm)
children(V) = internal black (confirm)

Sibling is red:-

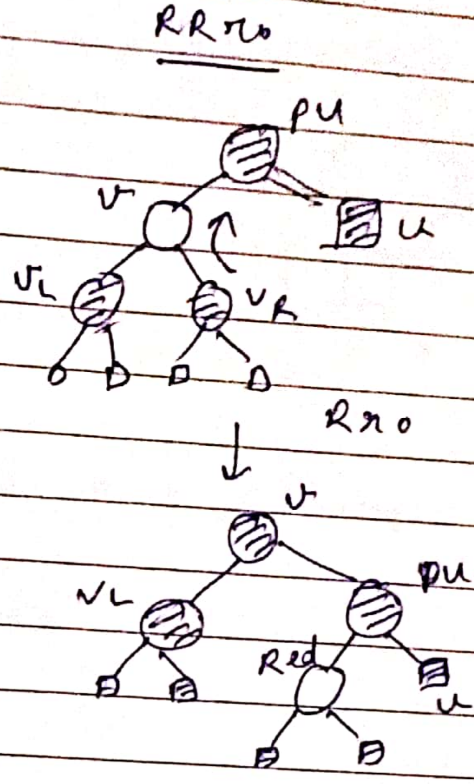
Rrr0

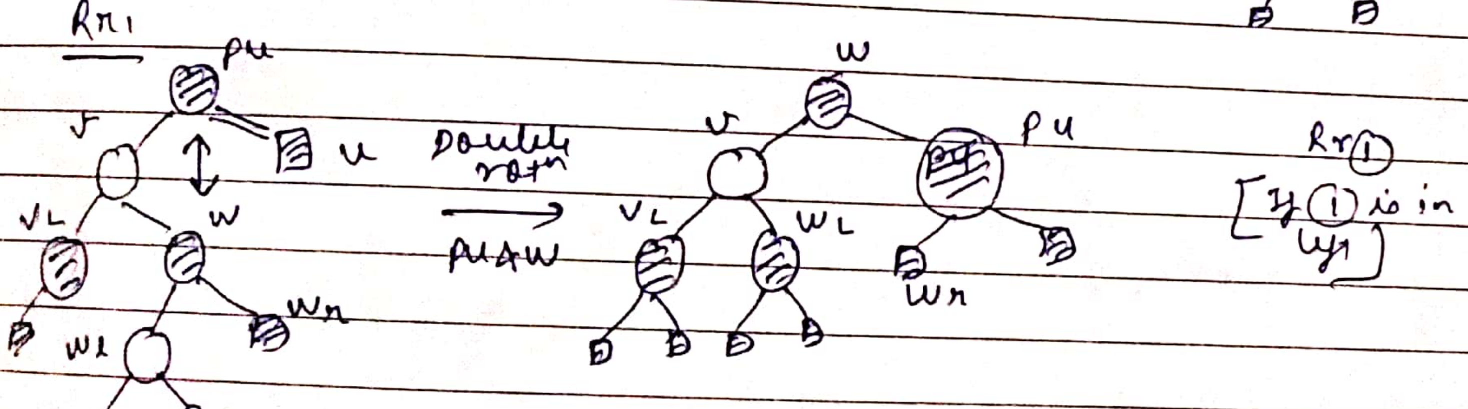


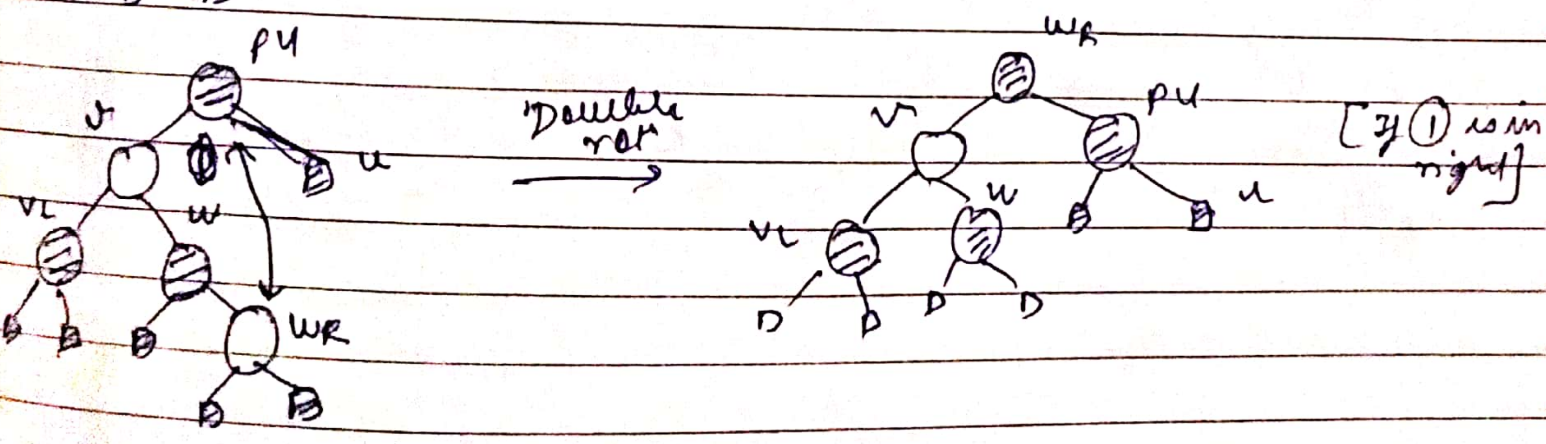
LRr0 or Lr1 or L
RRr0 or Rr1 or Rr2 → color(v)

color उसका निर्धारित Nomenclature है।

(V_R) = Right of v
children (V_R) = VRCs
No. of red child in VRCs

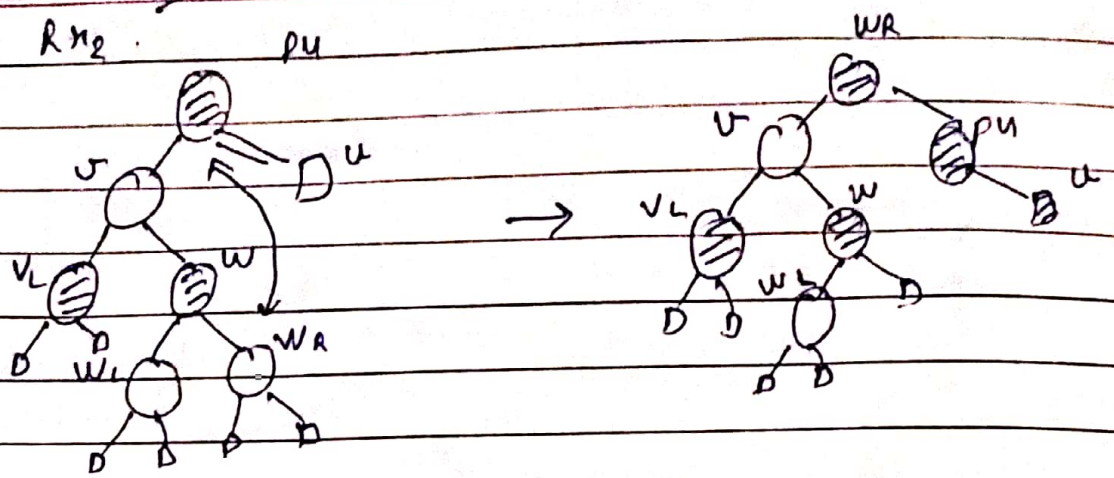






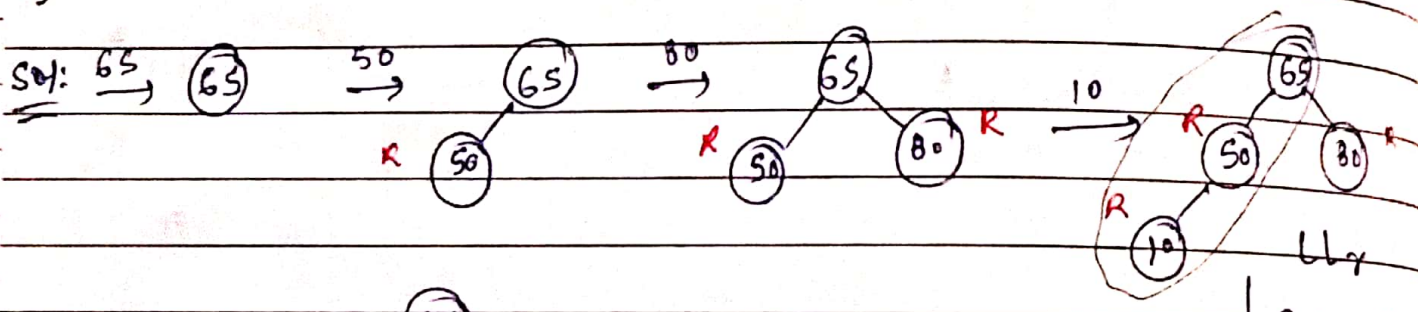
Same RR1

RR2

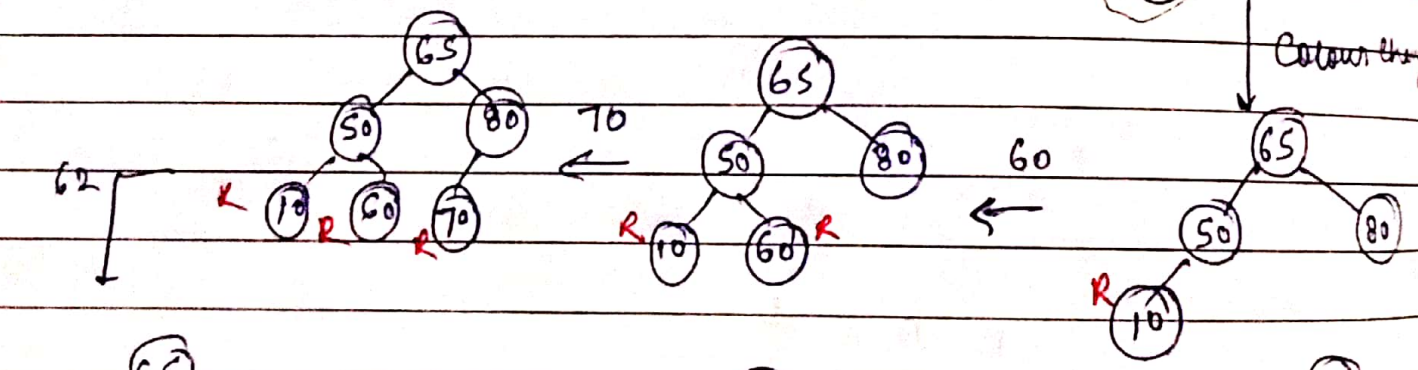


Q) Insert 65, 50, 80, 10, 60, 70, 62, 90, 63, 59, 58, 57, 56

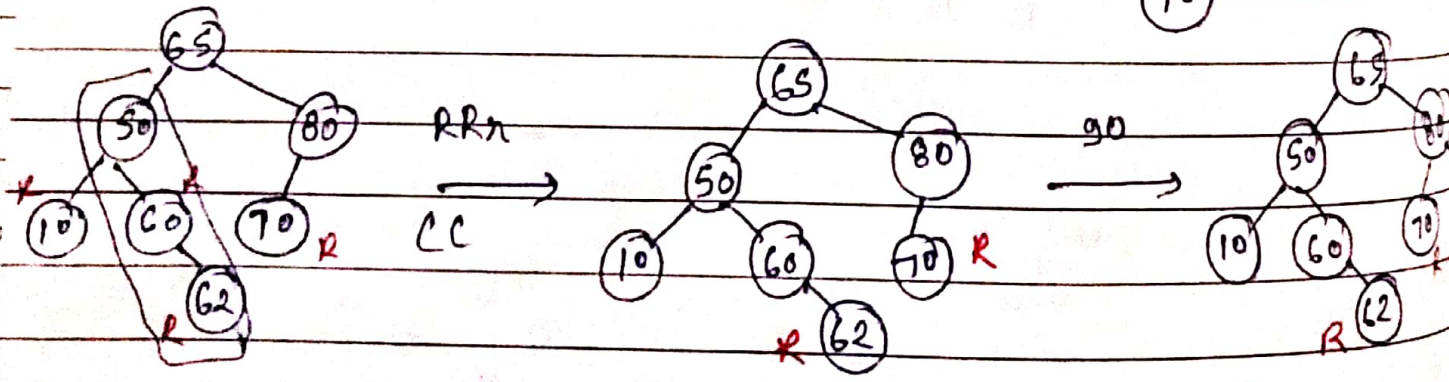
Sol: 65 → 50 → 80 → 10



LL



RRR



90

63