Sr. No.	Date	Subject	Page No.	Sign.

PAGE NO.
DATE / /

DSA

→ DS - Characteristics
- Correctnes
- Minimize time & space complexity

Need
- Search data
- Handle multiple requests
- Processor speed high

Execution Time Cases
Worst, Average, Best case

→ Algo - Characteristics
Unambiguous, well defined I/O, finite, feasible, independent

Analysis - A Priori - Theoretical analysis
A Posteriori - Empirical Analysis (Post Implementation)

Space Complexity :- Amt of memory space needed by algo.
Components - Fixed :- Independent of size of problem (constants & simple variables)
- Variable - Depends on size of problems (dynamic mem. allocation, recursion stack etc)

PAGE No.
DATE / /

Time Complexity :- Time needed by algo to run to completion

→ Asymptomatic Analysis :-
Define mathematical bounds of algo's runtime performance input bound

O Notation - Worst Case (Upper Bound)
Ω Notation - Best Case (Lower Bound)
Θ Notation - Avg Case

1. ARRAYS - Holds fixed no. of items of same type.
Can be 1D or ND

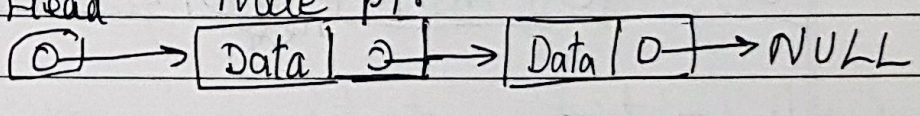
Traverse :- Simple for loop

Insert (At posn k)
Step1 Start
Step2 :- Repeat for i = n-1 to k
shift elt by one position down from i to i+1
Step3 :- set arr[k] = item
Step4 :- n ++
Step5 :- Exit

PAGE No.
DATE / /

Delete
1. Start
2. For i<k to n
   arr[i] = arr[i+1]
3. n--
4. Exit

2. LINKED LIST
linear collection of data elts (nodes)
Each node has one or more data fields and a pointer to next node
Null ptr denotes end of list
Extra space needed to store ptrs



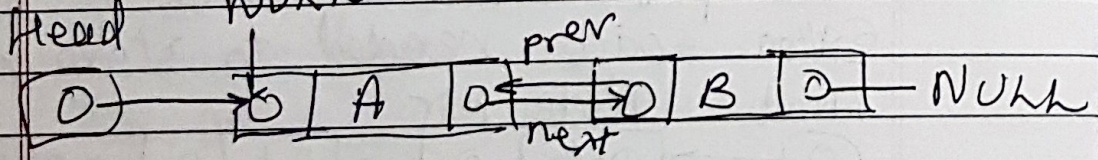
Operations :-
Insert :- Start
1. Link ptr of new node to next node
2. Link ptr of prev node to ~~next~~ new node

Delete
1. Link ptr of prev node to next node
(current node forgotten)

PAGE No.
DATE / /

Reverse Operation:-
1. Traverse to end of list, make pointer (currently null) to its previous node
2. Repeat for all nodes till first node.
3. Set ptr of first node to null
4. Make head pt to last node (from step 1)

→ Doubly linked list.
Each node has 2 pointers pointing to prev & next node



Insert:-
1. Set next ptr of current node to next node & prev of ptr to prev node
2. Make next of prev node & prev of next node point to this node

Handle empty LL, insert at start & at end accordingly

PAGE No.
DATE / /

Other operations :-
DeleteFirst, DeleteLast, Delete,
Display ~~Forward~~ Fwd, Display Backward

⇒ Circular Linked list
First element *prev points to last
& last elt (next) points to first

In single LL :- last node points to first node.

3. Stack :- LIFO Collection
Sequence of items, accessible at only one end of sequence

PUSH - Insert item into stack (top)
POP - Pop item from top of stack
TOP - current location of data in stack
PEEK - gets topmost elt (doesn't delet

PAGE No.
DATE / /

3.1 Polish Notations

Infix → a + b
Prefix → + ab
Postfix → ab +

Postfix must suitable for computers, universally accepted.
Any operation entered in a computer is converted to postfix, stored in stack and then calculated

Operator	Precedence
( )	4
exp {^}	3
* /	2
+ -	1

→ Infix to Postfix
Left paranthesis '(' → Push
Right paranthesis ')' → Pop and print till s[top] != '('
Then pop once more
Operand → Print
Operator → while(pre(in[i]) <= pre(s[top])
pop & print;
push operator

Note:- pre stands for precedence.
∴ While precedence of operator in infix expression <= Precedence of operator at top of stack

PAGE No.
DATE / /

Ex: A+(B*C-(D/E^F)*G)*H

Symbol	Postfix Str	Stack
A	A	
+	A	+
(	A	+ (
B	AB	+ (
*	AB	+ (*
C	ABC	+ (*
-	ABC*	+ (-
(	ABC*	+ (- (
D	ABC*D	+ (- (
/	ABC*D	+ (- (/
E	ABC*DE	+ (- (/
^	ABC*DE	+ (- (/ ^
F	ABC*DEF	+ (- (/ ^
)	ABC*DEF^/	+ (-
*	ABC*DEF^/	+ (- *
G	ABC*DEF^/G	+ (- *
)	ABC*DEF^/G*-	+
*	ABC*DEF^/G*-	+ *
H	ABC*DEF^/G*-H	

Pop full stack

ABC*DEF^/G*-H*+

~~Postfix to Infix~~

PAGE No.
DATE / /

→ Postfix Evaluation

If symbol is operand, push to stack
If operator:- op2 = pop
op1 = pop
Perform op1 operator op2 & push to stack

Ex 5 6 2 + * 12 4 / -

Symbol	Op1	Op2	Value	Stack
5				5
6				5 | 6
2				5 | 6 | 2
+	6	2	8	5 | 8
*	5	8	40	40
12				40 | 12
4				40 | 12 | 4
/	12	4	3	40 | 3
-	40	3	37	37

→ Infix to Prefix

(1) Reverse infix expression. (Note:- While reversing '(' will become ')' and vice versa)
(2) Obtain postfix expr.
(3) Reverse postfix expr.

Ex:- $\underbrace{((a+b)*(c-d))}$
$((d-c)*(b+a))$

Symbol	Prefix Str	Stack
(		(
(		((
d	d	((
−	d	((−
c	dc	((−
)	dc −	(
*	dc −	(*
(	dc −	(* (
b	dc − b	(* (
+	dc − b	(* (+
a	dc − b a	(* (+
)	dc − b a +	(
)	dc − b a + *	

Reverse
* + ab − cd

PAGE No.
DATE / /

4 RECURSION

Technique to break large problem to sub problems and call function recursively

→ Tail Recursion :- If no operations are pending when recursive function returns to caller.

Ex :- factorial :- (Tail)

```
int fact (int n) {
    return fact1 (n,1); }
int fact1 (int n, int res) {
    if (n == 1) return res;
    else
        return fact1 ((n-1), n*res); }
```

Ex Non Tail :-

```
int fact (int n)
{   if (x == 0 || x == 1) {
        return 1;
    else return (x * fact (x-1));
}
```

→ Non Tail Recursion :- Info about pending operation must be stored in stack. Hence, stack increases wt increase in calls.

∴ Tail recursion desirable.

→ Dired Reccursion:- If fnctn calls itself. Ex:- Prev fact fnctn

→ Indirect Reccursion:-

```
int fun(int n){ if (n == 0)
                    return n;
                else
                    return fun2(n); }
int fun2 (int x) {
                    return fun (x - 1);
    }
```

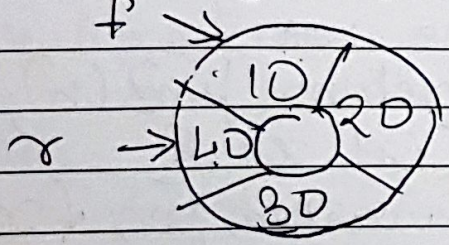
Calls to other function which ultimately calls it.

use of circular queue: Enables addition of elements in posn of deleted elts (prevents waste of space)

PAGE No.
DATE / /

5. QUEUE (FIFO)
Elt that gets in first, also gets out first. Elts added to rear & removed from front.

→ Circular Queue:



To increment front (f) or rear (r) we use
Delete: f = (f+1) % max len
Insert: r = (r+1) % max len
queue[r] = x

queue is full if f == (r+1) % max len
queue is empty if f == -1 and r == -1

→ Double Ended Queue (Dequeue)
Elts can be added and removed at either end
(Both ends must have f and r pointers) - called left and right
Dequeues can be input restricted (insert at one end only) or output restricted (remove from one end only.

PAGE No.
DATE / /

Implemented using circular array/ll

Note:- Stacks & queues can be implemented of arrays & or linked lists

6. ABSTRACT DATA TYPE (ADT)
Logically describing of how we view data and operations (data structures) without regared to how they will be implemented.

⇒ Priority Queue :- Each element is assigned a priority.
Priority determines which elt will be processed first.

ADT is like a blueprint.
It gives requirement & operations
Ex Array :-
Minimal Read Fnctionality :- Get elt at i & set an elt at pos^n & n
get(i) - get elt i
set(i, num) - set elt at pos^n n.

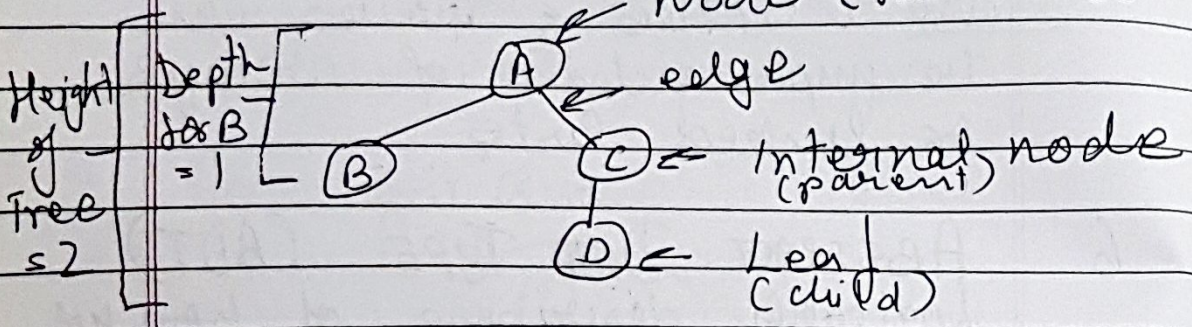
Operations :- max(), min(), search(num), insert(i, num), append(x)

Trees & Graphs →
Non Linear DS

PAGE No. | DATE / /

7. TREES

Non linear, Hierarchial DS

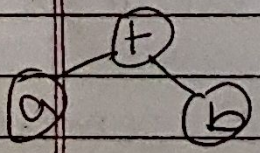


Height → Longest path to a leaf
Depth of a node : Length of path to root

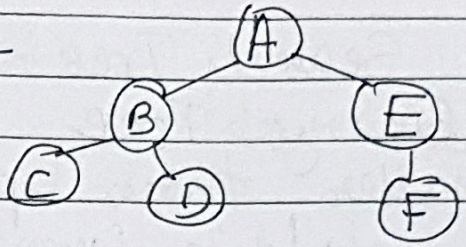
⇒ Binary Tree : Can have atmost two children

- Traversal of Binary Tree :
  L - Left Subtree
  R - Right Subtree
  D - Print Data

1. Inorder (LDR) ⇒ a + b
2. PreOrder (DLR) ⇒ + a b
3. PostOrder (LRD) ⇒ a b +



Ex :-

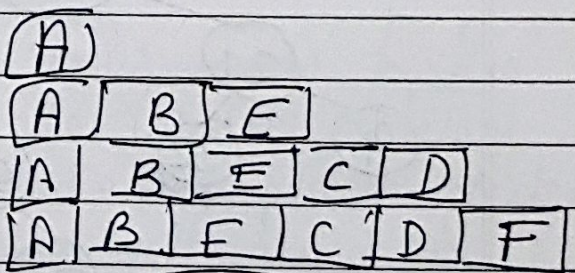


Inorder :- C B D A F E
PreOrder :- A B C D E F
PostOrder C D B F E A

⇒ PreOrder is same as Depth First Search (DFS)

→ Breadth First search (BFS) Traverses tree level by level (Use queue).
It would be like :-
A B E C D F

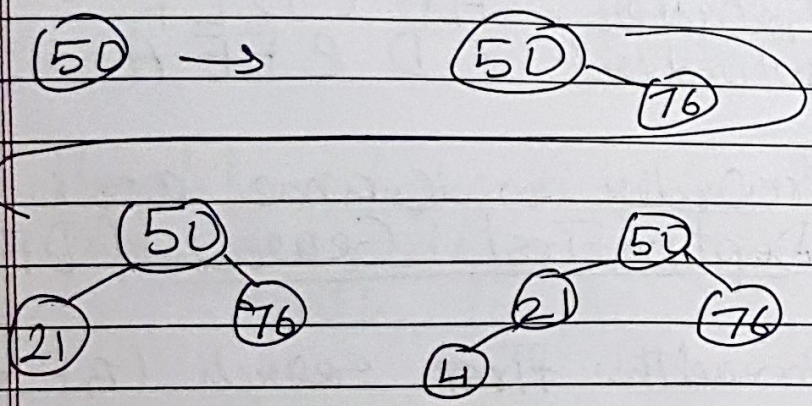
Queue :-



PAGE No.
DATE / /

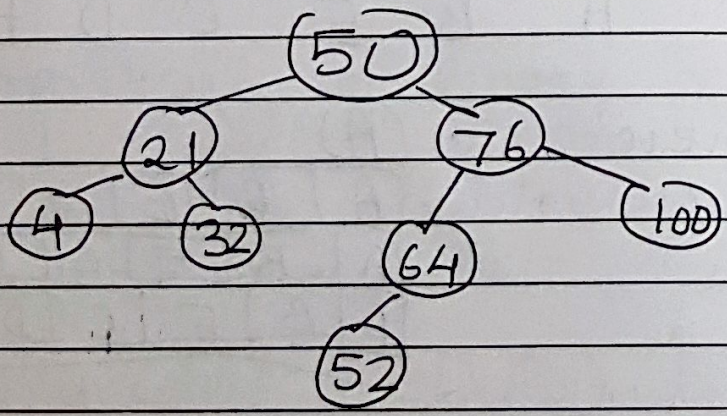
→ Binary Search Tree :-
Sorted Binary Tree. Left child is smaller than parent, right child is larger than parent

Ex :- 50, 76, 21, 4, 32, 100, 64, 52



...

Final Tree :-

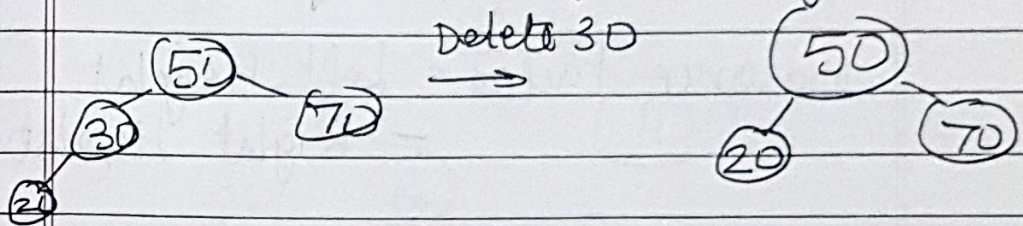


To find a value (x) :-
x == node → return true
x < node → traverse left
x > node → traverse right

Delete :-

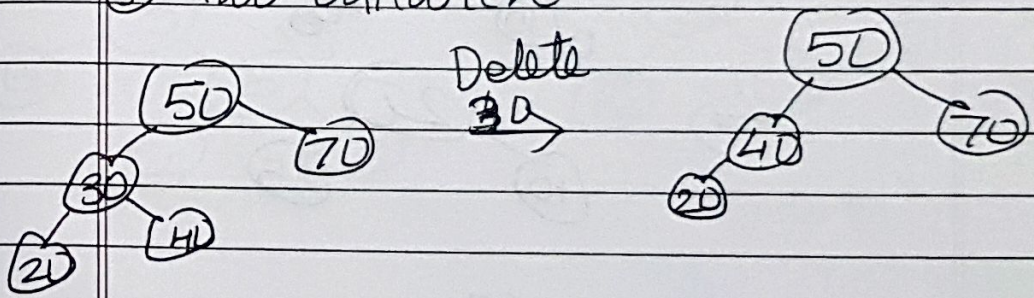
(1) No children (leaf node)
Just delete

(2) One child (Left or Right)



Make parent of node to be deleted point to child.

(3) Two children



Find the min value node to the right of node to be deleted. Add this node in place of deleted node & make left child of deleted node, left child of the new node and right child of deleted node, right child of new node (if applicable)

PAGE No.
DATE / /

→ AVL Tree

Height balanced BST.

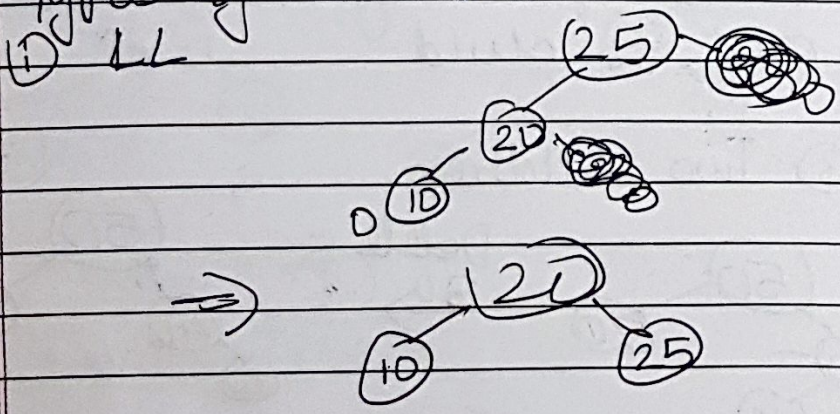
Height of the node kids is at most 1.

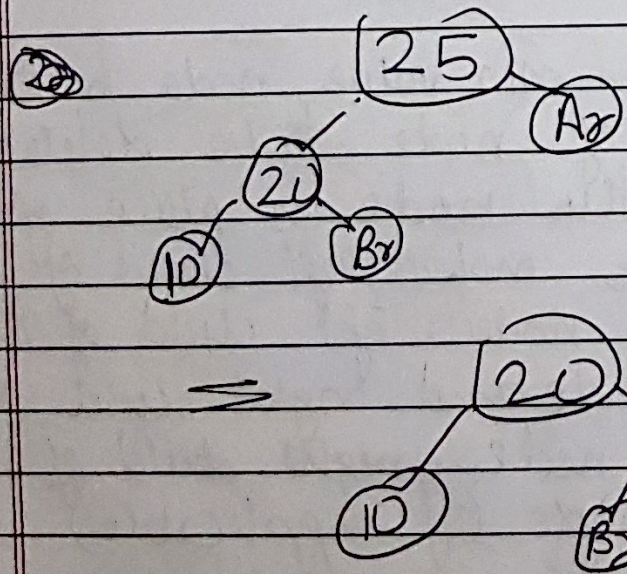
Correct balance factor is 1, 0, -1

Balance Factor = Left_Height − Right_Height

Types of Rotations :-

1) LL

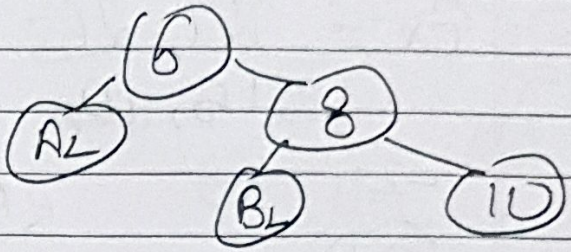




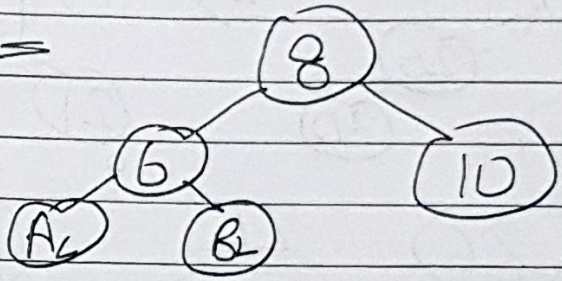
Viewing, Insertion, Deletion
takes O(log n)

PAGE NO
DATE / /

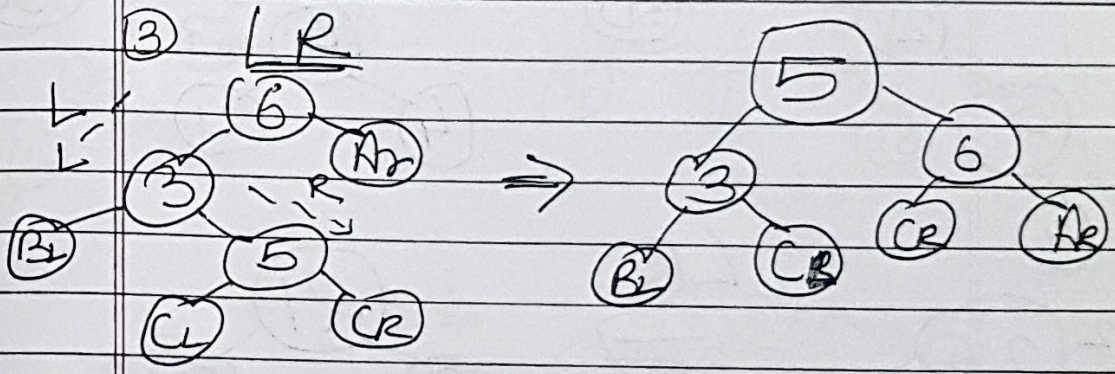
② RR



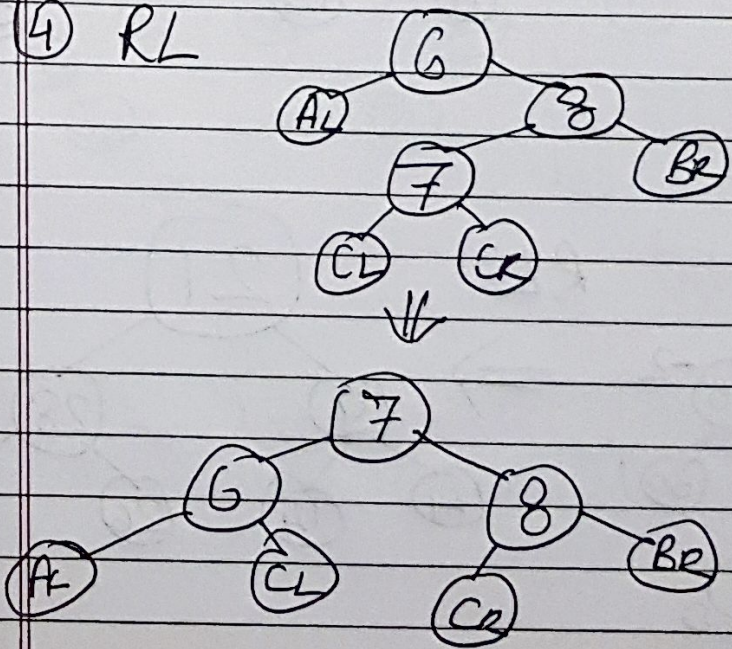
=



③ LR

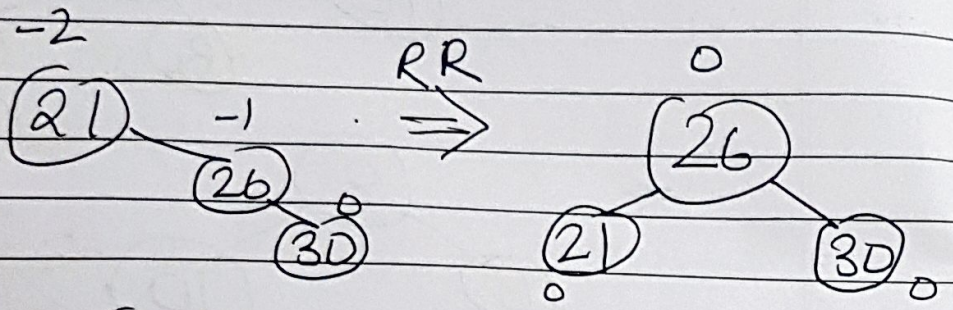


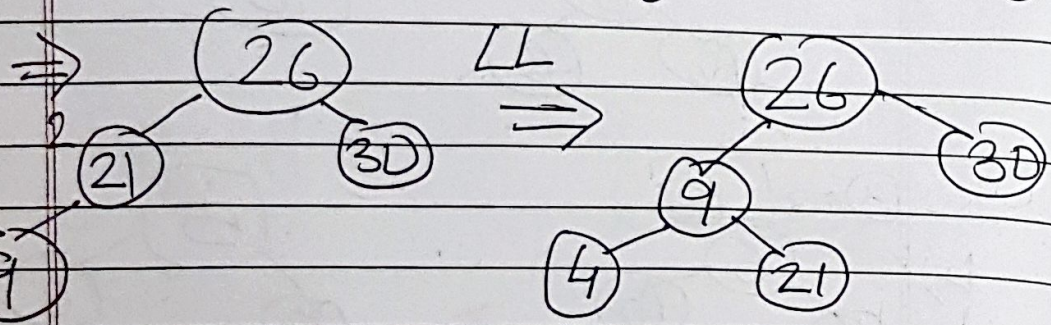
④ RL

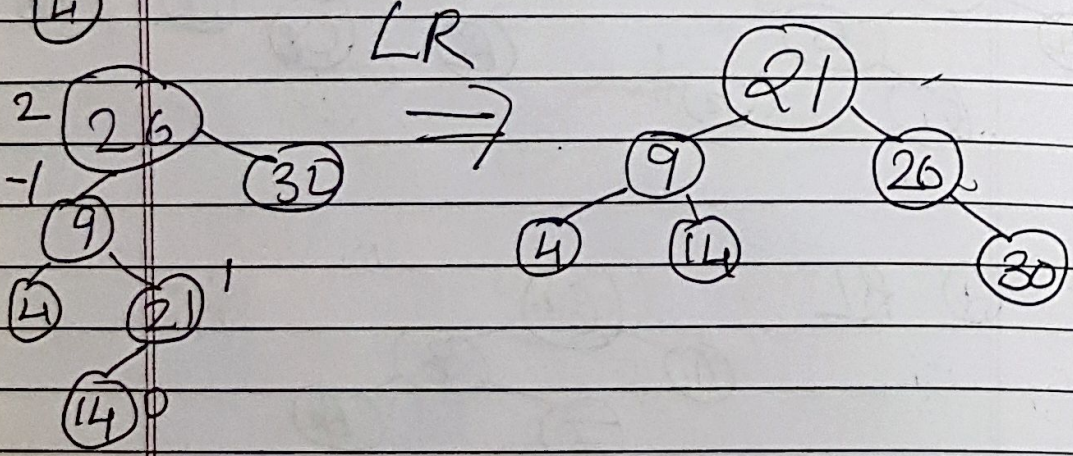


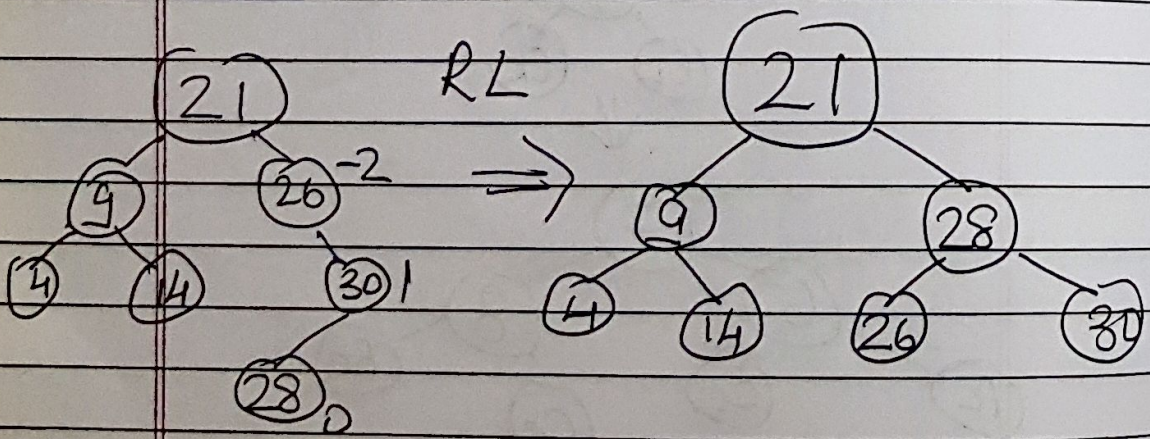
PAGE No.
DATE / /

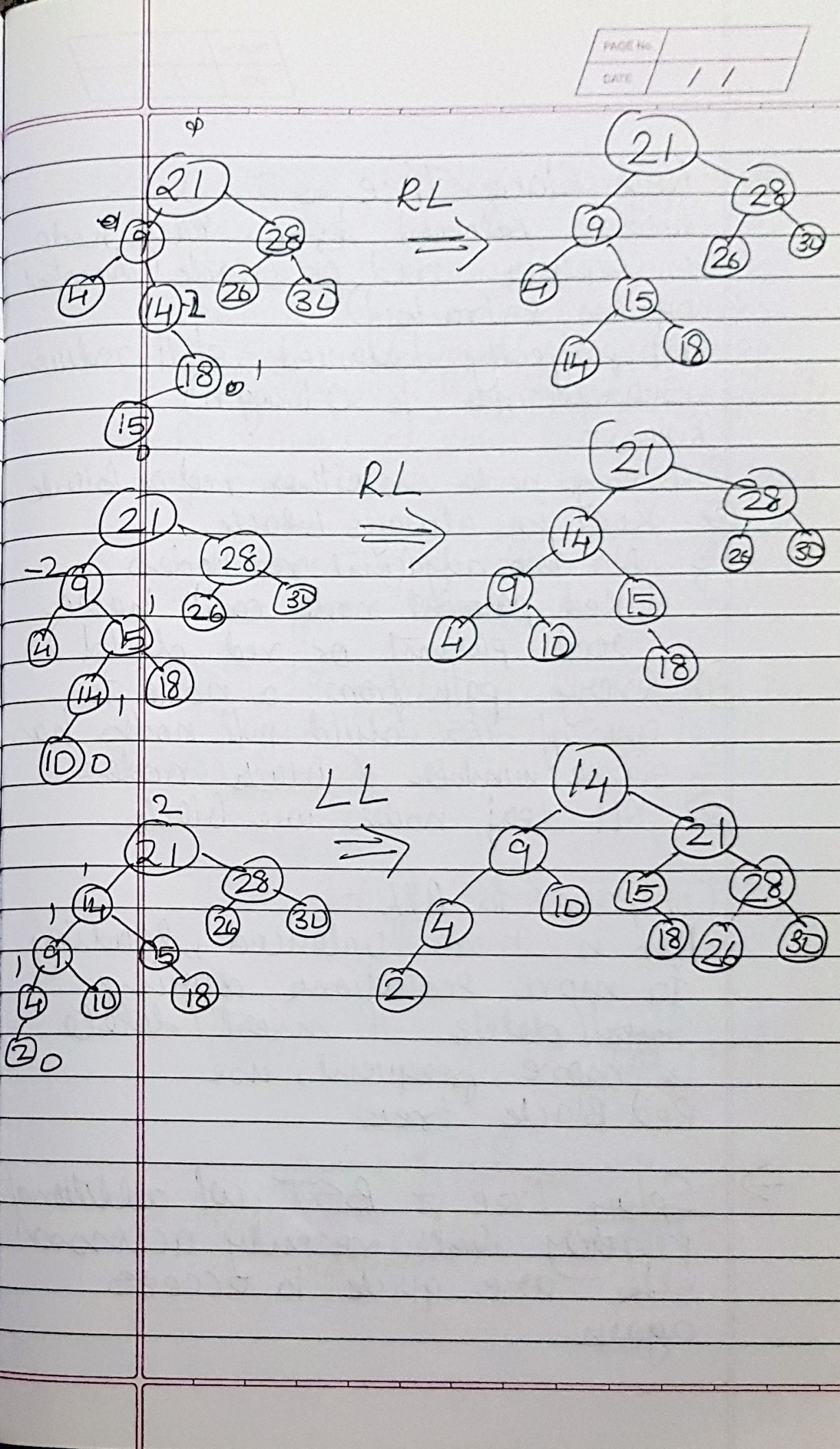
Ex:- 21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2,










RL
RL
LL

PAGE No.
DATE / /

⇒ Red Black Tree

Another balanced tree. Each node is either red or black (denoted by an extra bit)

Not perfectly balanced, still reduces search time to O(log n)

Rules :-

1. Every node is either red or black
2. Root is always black
3. No two adjacent red nodes (Red ~~parent~~ node can't have red parent or red child)
4. Every path from a node to any of its child null nodes has same number of black nodes
5. All leaf nodes are black

Compare to AVL :-

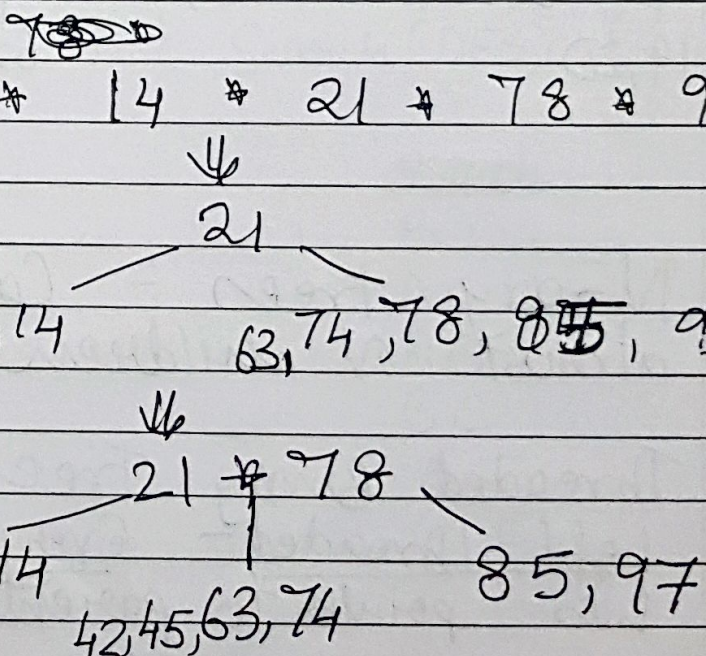
AVL is more balanced, leading to more rotations during insert/delete. If insert/delete is more frequent, use Red Black tree

⇒ Splay Tree :- BST w/ additional property that recently accessed elts are quick to access again

PAGE No.
DATE / /

⇒ B-Tree :- Self balancing tree.
When data is too large to be stored in main memory, it is read of from disk in chunks.
B-Tree designed to minimize disk access O(logn) complexity.
- All leaves at same level
- Each node can hold m-1 keys where m is disk size.
- Number of children = no. of key + 1
- Tree grows & shrinks at root

Ex:- 78, 21, 14, 11, 97, 85, 74, 63, 45, 42, 57, 20, 16, 19, 30 32, 30, 31 m = 5



Root will split

⇒ N-ary trees :- Can have atmost N childrens

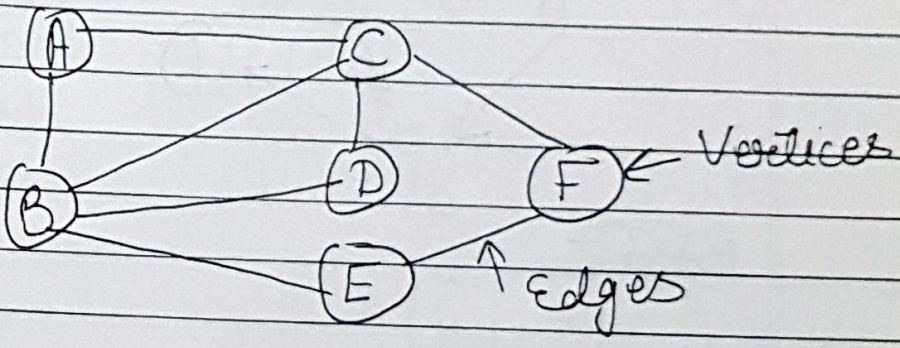
⇒ Threaded Binary Tree

Left Threaded :- Every right child has pointer to parent

Right Threaded :- Left child has pointers to parent

~~8.~~ ~~GRAPH~~
- Two way threaded - Both children point to parent.

8. GRAPH
Non linear DS with vertices/nodes and edges. G(V,E)



Breadth First Search (BFS) (Queue)
Depth First Search (DFS) (Stack)

BFS :-

	~~Stack~~
A	B C
B	A C D F
C	A B D F
D	B C
E	B F
F	C E

Visited

A	B	C	D	E	F
1	1	1	1	1	1

Queue
O/P

A	B	C	D	E	F
A	B	C	D	E	F