4-jan-2017

Array :

- Same data type
- dimension
- array name
- index / subscript
- size
- contigous memory

its stored only single data type data.

- int data type size is 2 Byte in turboc & 4 byte in codeblocks.
- Block add. + S(DT) * i
  base add. ↳ index (0 to n-1)

Matrix :

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

Row major arrangement : 1 2 3 4 5 6 7 8 9
Column major " : 1 4 7 2 5 8 3 6 9

Structure :

for ex: Student [Name , phone, Gender]

```
char name [66] [100] ;
long int phone [66] ;
char gender [66] ;
```
} difficult to handle

```
struct student
{
    char name [100],
    long int phone ,
    char gender
};
```

Structure is collection of dissimilar data but related data.

```
# define struct student stud
    stud IE_DS [66]
    struct student IE_DS [66];
        IE_DS [0] . name
        IE_DS [0] . phone
        IE_DS [0] . gender
```

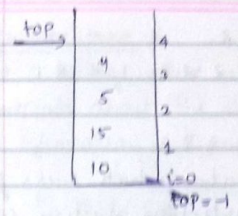
variable

Stack:

Last in first out - LIFO

```
int stack [10];
```

- every stack has the stack pointer



- top = -1

operations:

① Push ( ):
→ element → top++

② pop ( ):
→ remove element → top = top-1

③ Stack empty / Underflow (no elements)
if (top == -1)

④ Stack full / overflow
if (top == size-1)

* Undo operations (stack is used)

Example: Given set of element 3, 7, 1, 10, 15, 1, 5, 9 if Push=1 and pop=0, identify which operation is valid and formulate sequence of operations

When stack size is 7 & 8.

i) 1 1 1 2 2 1 1 1 1 2 2 2 2 2 1 2 2 1

ii) 1 2 2 1 1 1 2 2 1 1 1

iii) 1 1 1 1 1 2 2 2 1 1 2 2 2 2 2

If seek is a operation which looks for a given element at a particular position can you give the element when top value is 5, 4, 3 in the above three question.

Sol:-

Stack is empty

i) not valid because

3 7 1 → three time push

3 → two tym pop

3 7 1 10 15 → four tym push

→ five tym pop

3 → 1) " push

→ 2 tym pop → not valid

ii) not valid

iii) not valid

5-Jan

Data structure :

data → a set of value is "data." data is static by nature.

information → Data about the data is "information." By nature information is Dynamic.

Data type

- numeric: int, float, double
- non-numeric: char

primitive data type / Basic built in

non-primitive DT (string, arrays, Structure, stacks, Union, queue)

data structure → a collection of data types organised for a particular purpose.

operations on a particular purpose data type and how to do it.

- abstract data structure is only consist of necessary information - technical information
- Basic data structure - int, float, double

complex number :-

real x

imag. y

Subject .....

School / College .....

data structure

linear (controus in nature)

array stack queue link list

- consecutive in memory

non linear

graph trees tables sets

- stores in distributive in memory.

1D array :- A[i] = BA + D*(i-1)
= LB + D*(i-1-L)

2D array :-
MXN if (4x3)

a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
.
a_{41} a_{42} a_{43}

a_{11} is base addr. → Lo

A[2,3] = Lo (3-1) row (2-1) column

row major A(i,j) = Lo + w{(i-1)n + (j-1)}

w → type of data type (size of data in byte)

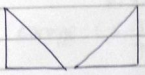
int → 2 byte
float → 4 byte
double → 8 byte
long double → 16 byte

row major (MXN)
A(i,j) = Lo + w {(i-1)N + (j-1)}

column major (MXN)
A(i,j) = Lo + w {(j-1) + (i-1)M}

M → no of rows
N → no of column

Spasse matrix:
it is same as 2-D matrix but the no of elements is half of the size of matrix.


triangular spasse matrix


diagonal sm


Cluster sm

algorithm: PUSH

Step 1: check top = size - 1

Stack size top

push (stack, size, top)

Step 1:- if top == size - 1
print stack overflow
goto end

Subject .....
School / College .....

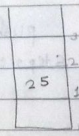
Step 2:- else
Step 3:- read x
Step 4:- increment top / top= top+1 / top++
Step 5:- stack(top) = x
end:- end

⇒ pop (stack, size, top)
Step 1: if top == -1
print stack is empty
goto end
Step 2: else
step 3: x = stack(top), print x
Step 4: decrement top by 1
end: end

peep / peek :

peek / peep → it gives the value of top element
function peep (S, Top)
Step 1: if top == -1
print no elements
Step 2: return (S[top])

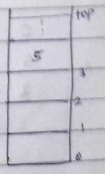
→ change (S, 25, 2, top)
S1: if top == -1
print no operation



S2: else
x = top
S3: if s[x] == 25 ;
S4: s(x) = 2 ; go to end.
else
x = x-1
S5: if x >= 0
then goto S3
S6: else
end print 25 not found
end S7: end

→ change (S, 3, 7)
↑ value
location

change (S, L, Vnew, top)
S1: if top == -1
print empty
else
if top >= L
S[L] = Vnew
end if
end if
S2: end



Subject .....
School / College .....

Applications:

2D Array:-

Solving a polynomial equations (with two variables)

for ex: $2x^2 + 5xy + y2$ —① $6xy + 5x + 7y + 8 + ex^2$ —②

A [4][4]

0 to (4-1) ⇒ 0 to 3

	y^0	y^1	y^2	y^3
x^0	0	1	1	0
x^1	0	5	0	0
x^2	2	0	0	0
x^3	0	0	0	0

	y^0			y^3
x^0	8	7	0	0
	5	6	6	0
	6	0	0	0
x^3	0	0	0	0

① + ②

8	7	1	0
5	11	6	0
8	0	0	0
0	0	0	0

$7y + 5x + 11xy + 8x^2 + y^2 + 8$

Application of stack:-

① polish notation: a polish notation is the representation of any operation expression.

infix — Prefix — Postfix

a + b * c (operand, operator, operand)

Prefix: (operator operand operand) +ab

Postfix: (operand-2 operator) ab+

a + b (operand operator operand)

* bracket > Power > Mult/Div > Add/Sub.

BODMAS left → Right

Convert in postfix or prefix: (to use the no of scans)

	Infix	Prefix	Postfix
i)	a+b*c	a+(*bc)	a+(bc*)
		= + a*bc	abc*+
ii)	(a+b)*c	(+ab)*c	(ab+)*c
		= *+abc	ab+c*
iii)	(a+b)↑c*d+c	(+ab)↑c*d+c	
		↑+abc*d+c	
		*↑+abcd+c	
		+*↑+abcdc	
		++*↑abcdc	

post: (ab+)↑c*d+c

= ~~(ab+)(d+c)↑~~ ab+c↑*d+c

= ~~abc d+c↑*~~ ab+c↑d*+c

= ~~abcd+c↑*+~~ ab+c↑d*c+

= abcdc↑*++

iv) a*b*c/d + (a+b*(c-d))

pre: a*b*c/d + (a+b*-cd)

a*b*c/d + (a+*b-cd)

a*b*c/d + (+a*b-cd)

(*ab)*c/d + +a*b-cd

**ab c/d ++ a*b-cd

/**abcd ++ a*b-cd = +/**abcd+a*b-cd

Subject .....

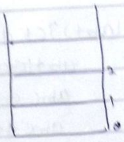
School / College .....

post: a*b*c/d + (a+b*cd-)
a*b*c/d + (a+bcd-*)
a*b*c/d + (abcd-*+)
ab**c/d + (abcd-*+)
ab*c*/d + (abcd-*+)
ab*c*d/ + (abcd-*+)
abc*d/ abcd-*++

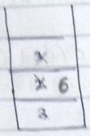
(v) 3+2*3 pre: 3+*23
= 3+6 = +3*23
= 9

post: 3+23*
323*+

For postfix by using Stack:
1) push operand
11) operator O
push(pop(2)() O pop(1))

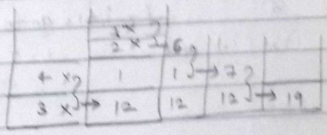


323*+
3 push 3
2 push 2
3 push 3
pop(2) O pop(1)
2 * 3 = 6
push (6)



3 6 +
3+6 = 9 PUSH(9)

Example- 34*123*++
3 PUSH(3)
4 PUSH(4)
POP(2) POP(1)
3 * 4 = 12 ; PUSH(12)
1 PUSH(1)
2 PUSH(2)
3 PUSH(3)
* POP(2) * POP(1)
2 * 3
= 6
PUSH(6)
+ 1 + 6 = 7
PUSH(7)
+ 7 + 12 = 19



- If we are doing postfix, evaluation part becomes easy.

Example: Write an algorithm to solve a given a postfix form of an expression.

Algorithm:
S1 if x = operator if a(i) = = 1ϕ ; go to end
S2: else if a(i) == operand
push [a(i)]
S3: else if
a(i) == operator

top
y = pop(top) . a(i) . pop(top)
push(y)

S4: else if
a(i) is not operator / operand

S5: print invalid operation.
S6: i = i+1 go to step 1
end: end

18 jan

② **Recursion:**
- termination limit will be there.

for example:

fact(N) = { N = 0 → 1 ; N × fact(N-1) → Otherwise }

5! → 5×4! → 4×3! → 3×2! → 2×1! → 1×0! → 0!

- Function fact(n)
if n = 0
pop (value, addr)
return 1
else
x ← n × fact(n-1)
push(n, address, x)
return x

∧ → Right to left
BODMAS → left to Right

this called as recursive nature of the function factorial.

value	address
1	
2	
3	B×3
4	A×6
5	AB1

↑value ↑address

algorithm:
FACT(N)
1: PUSH(N, top, temp addr)
2. if N = 0 — Base criteria
factorial ← 1
goto end
3. else
factorial ← N * fact(N-1)
4. pop (top, addr)
go to temp addr
5. end

for example:
[120] 5!
[5×4!] 5×4!

Conversion of Infix to Postfix:

^ →	3	
* / →	2	
+ - →	1	
(→	0	- lowest precedence

ans 1 is valid and ans 0 is invalid

for ex: a + b b
1 -1 + 1 = 1

Ex: a + b * c - d / e * b #

input symbol	stack	o/p string postfix	Rank
a	a	a	1
+	+ (a +)	a	1
b	+b	ab	2
*	+*b — pop b	ab	2
c	+*c — pop	abc	1
-	~~+*c(-) pop~~ —	abc*+	1
d	-d	abc*+d	1
/	-d-/	abc*+de	2
e	-/e	abc*+de/	2
*	-*	abc*+de/b	1
#	-*b	abc*+de/b*	2
#		abc*+de/b*-	1

acc. to precidence: 3>1 → is not chalega

1<3 → chlega

- Stack should contain elements having acending precidence.
- if precendance of I/P symbol is high then Push it.
- " " " " is low then pop it.
- all the stack elements with high or equal precedence and then push the i/p symbol.
- # means pop all the elements.

Symbol	Precedence	Rank
ab	3	1
*/	2	-1
+-	1	-1

Algorithm:

input

Step 1. Top ← 1
S[Top] ← '1'
2. Polish ← ' '
Rank ← 0

Step 2. next ← next symbol(input)
repeat step 4 till next = '#'

Step 3. if top < 1
print 'INVALID' stop

School / College .....

Step 4. while Pre (s[TOP]) >= Pre(next)
Temp ← pop (s[top])
~~Operand~~ append (postfix, temp)
[Postfix ← prefix O temp]
Rank ← rank + r(temp)

Step 5. push (next, top)

step 6. if rank == 1
print "invalid"
else
print "invalid"

⇒

		I(P)	S(P)	Rank
L→R	+ -	1	2	-1
L→R	* /	3	4	-1
R→L	↑	6	5	-1
	variables	7	8	1
	(	9	0	-
	)	0	-	-

Example $a_7 +_1 b_7 \uparrow_6 c_6 \uparrow_7 d$

Incoming symbol	stack	Postfix	Rank
a_7	a_8	—	0 -
$+_1$	$+_2$	a	1
b_7	$+b$ (2 8)	a	1
$\uparrow_6$	$+\uparrow_5$ (2 5)	ab	2
c_7	$+\uparrow c$ (2 5 8)	ab	2
$\uparrow_6$	$+\uparrow\uparrow$ (2 5 5)	abc	3
d_7	$+\uparrow\uparrow d$ (2 5 5 8)	abc	3
		abcd↑↑+	

example $(a + b \uparrow c \uparrow d) * (e + f/d) \#$

Incoming symbol	stack	Postfix	Rank
(	$(_0$	-	-
a	$(a_8$	a	1
+	$(+_2$	a	1
b	$(+b$	ab	
↑	$(+\uparrow_5$	ab	
d	$(+\uparrow d$	abd	
)		abd↑+	
(	(	-	-
a	(a	-	-
+	(+	a	1
b	(+b	a	1
↑	(+↑	ab	2
c	(+↑c	ab	2
↑	(+↑↑	abc	3
d	(+↑↑d	abc	3
)	-	abcd↑↑+	1
*	*	abcd↑↑+	1
(	$*(_0$	abcd↑↑+	1

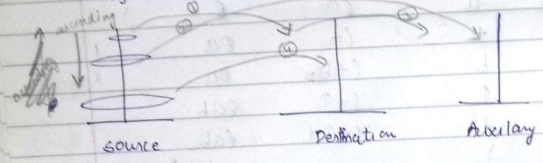
When closing) then all elements are POP and in Post fix there is no any parenthesis is showing.

School / College .....

19-Jan

e	*(e	abcd↑↑+	1
+	*(+	abcd↑↑+e	2
f	*(+f	abcd↑↑+e	2
/	*(+/	abcd↑↑+ef	3
d	*(+/d	abcd↑↑+ef	3
)		abcd↑↑+ef/+*	1

Example Tower of Hanoi → using recursion
- Lighter weight of disk on the top



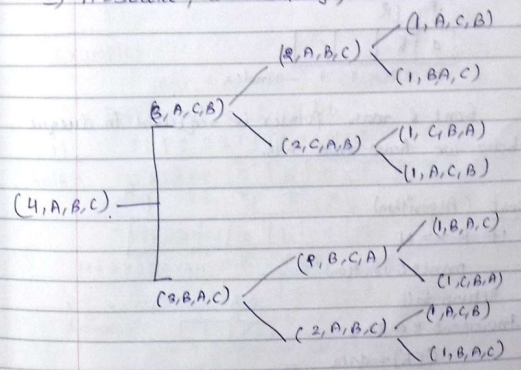
Algorithm:

```
function MoveTower (disk, source, dest, spare)
if disk == 0
    move disk from source to dest.
else
    move tower (disk-1, source, spare, dest)
    move disk from source to dest
    movetower (disk-1, spare, dest, source)
```

⇒ func tower(disk, source, A, D)
(Auxilary, Dest.)

```
If (disk == 0)
  move disk from S to D
else
  tower (disk-1, S, D, A)
  move disk from S to D
  tower (disk-1, A, S, D)
end if
```

⇒ A = source, B = Auxilary; C = destination

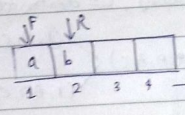


School / College .....

Queues :

- (FIFO)
- Queues have two ends.
  - max_items: max. no of items that might be on the queue.
  - item_type: data type of the items on the queue.
- enqueue & delete queue
  (insert) (delete)

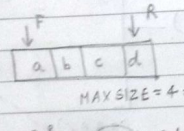
25-Jan

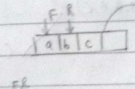


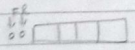
front & rear pointer is required in queue bcuz we have two ends.

Q Insert (Algorithm)

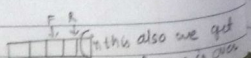
if R >= N
  print "Overflow"
  Return fail
Increment R ← R+1
  Q[R] = data
if F = 0
  F ← 1
Return







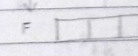
even after this are some spaces are blank.

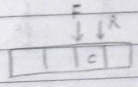


In this also we get the Queue is over.

Q Delete

if F = 0
  print (" underflow"
Return fail
  Y = Q(F)
if Q is empty
  F = R = 0 ↯ set to 0
else
  F ← F+1
Return(Y)



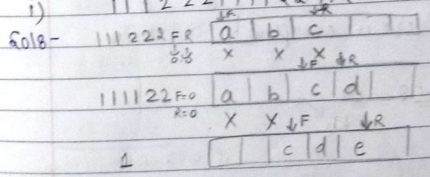


Examples

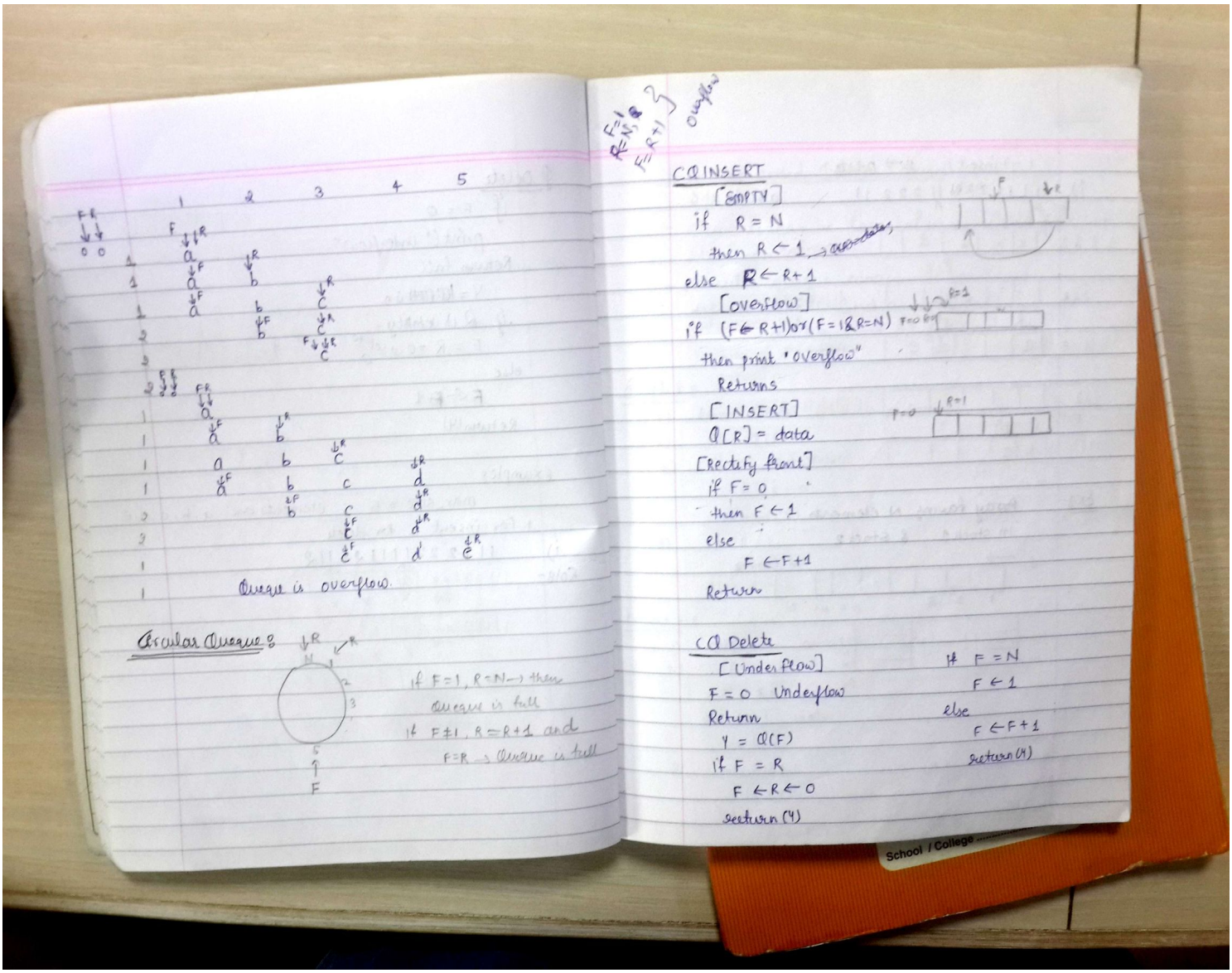
max_size = 5 ; elements are a, b, c, d, e

1 for insert 2 for delete.

i) 1 1 1 2 2 2 1 1 1 2 2 1 2

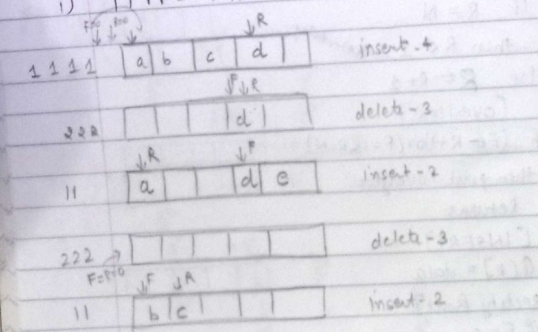


1 → overflow queue

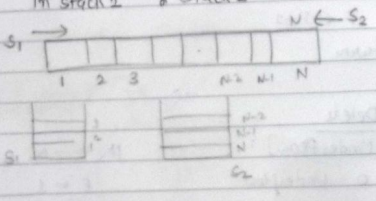


1 → insert 2 → Delete

i) 1 1 1 1 2 2 2 1 1 2 2 2 1 1 abcdeabc



Exa- Array having N elements in stack 1 & stack 2



1-feb

Double ended Queue (Dequeue):

- elements can be inserted and deleted from both the ends.
- circular queue concept is considered.
- default Insertion → at back end
  deletion → at front end

Delete R:

1) linear flow conditions (same)
2) Y = Q(R)
3) if F = R
   F = R = 0
   return (Y)
4) if R = 1
   R = N
   else R ← R - 1

Insert F:

1) overflow condition (same)
2) if F = 0
   F = R = 1
   else Q(F) = data
   return
3) if F = 1
   F ← N
   else
   F ← F - 1

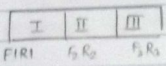
Subject .....
School / College .....

- Input restricted queue → insertion at one end
  deletion at both ends
- O/P restricted queue → insertion at both ends
  deletion from both ends one

Priority Queue:

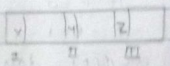
- can be implemented using
  - 1-D array
  - multi-dimensional array

1-D Array:



highest - I
lowest - III

- The priority while inserting is given while function calling
  Ex: Insert (x,1)
  Insert (z,3)
  Insert (y,2)



- While deleting an element - deletion takes place 1st in highest priority then after completing all the deletion from highest priority then 2nd priority list elements starts to deletion.

Multi-dimensional array:

	1	2	3	4
1	X	Y		
2				
3	Z			
4				
5				

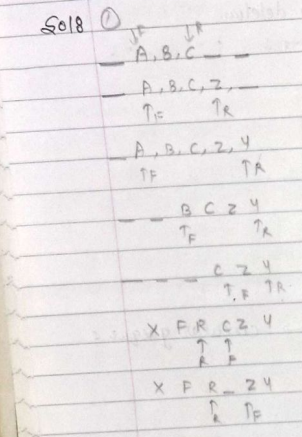
↓F ↓R

Ex: __ A, B, C __ __ → circular queue:
Insert (z)
Insert (Y)
delete delete
Insert (X)
Insert (F)
Insert (R)
Delete

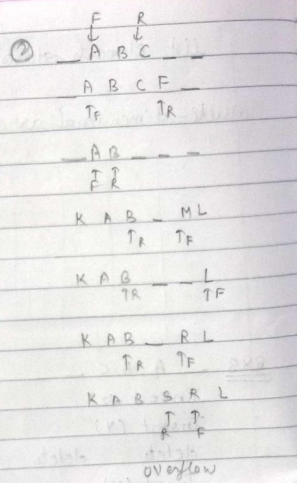
Ex:- for the same queue, as circular double ended queue, the operation are ① insert F from at the rear end
① delete, delete from rear
② K, L, M are added from front end
1 delete from front
add R from front, add S from rear, add T from

Subject .....
School / College .....

Sol8 ①

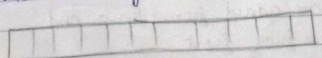


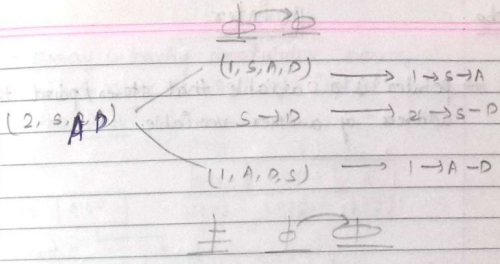
②

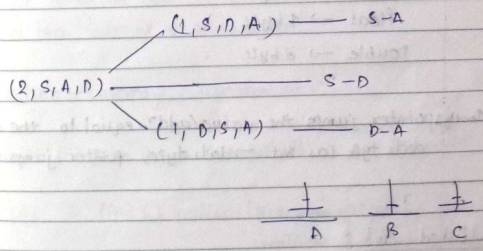


Ex: How to implement 2 stack in 1 array A[1... N] in such a way that neither stack overflows unless the total no of elements in both stacks together is N. The push & pop operations should run in a constant time:

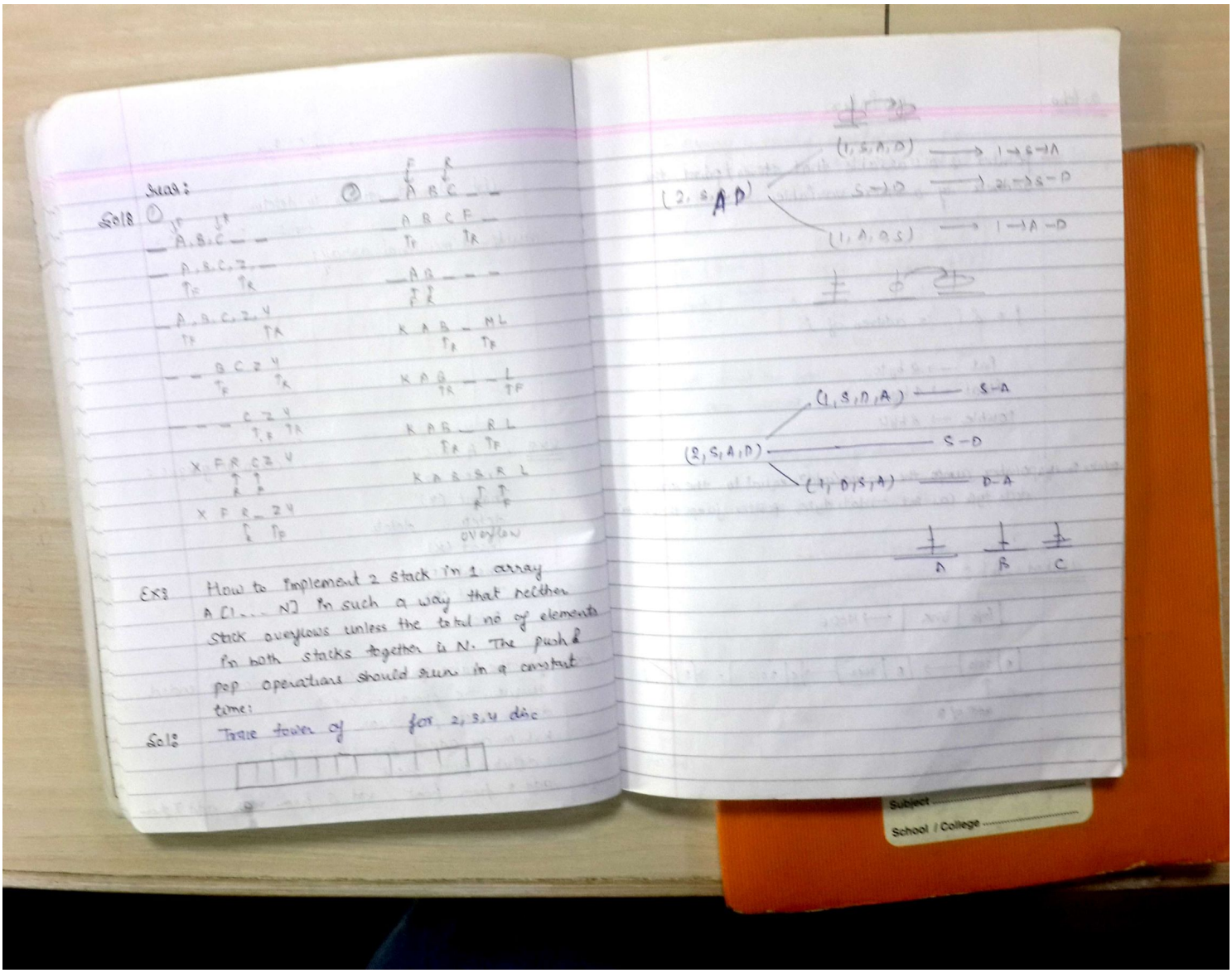
Sol: Tracе tower of for 2, 3, 4 disc







Subject .....
School / College .....

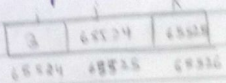


Scanned by CamScanner

9-feb

pointer

- pointer is a variable that stores / point the address of another variable.

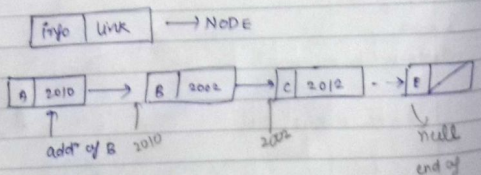


j = &i = address of i
(&i: Physical add, i: Logic add value)

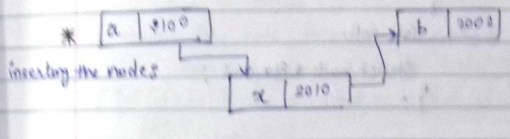
int → 2 byte
float → 4 byte
Double → 8 byte

pointer float by pointer jumps the value/addr equal to the size of data type (as int → data type, pointer jumps the 2 bytes)

Linked list:



- array is having constitutive memory.
- not to loose the track of link list, we are having the "first" pointer.



inserting the nodes:

- first stored the addr of B in x.
- and then addr of x stores in a.

malloc → memory allocation in C

- *ptr → given the value which is at ptr = addr
- ptr = (int *) malloc (n * size of (int));

link list can be used [linear/nonlinear] diff^n data structures

- Array
- Stack } - linear
- Queue

Subject .....
School / College .....

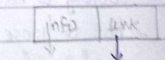
15-feb

LINKED LIST

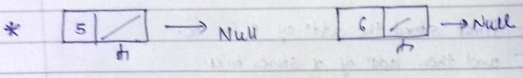
→ Singly linear link list

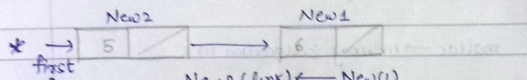
* 1 linked field
* Struct Node

```
{
    int info;
    Struct node * link;
};
```



no or name — add^{n} of next element





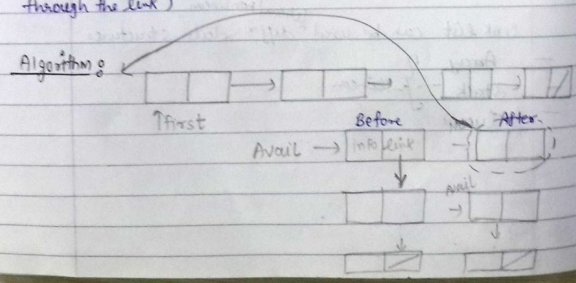
New2 (link) ← New(1)

add^{n} of New 1 link list element stored in New 2 add^{n} which was previously null.

point to 1st node in link list

(used to traverse through the link)

Algorithm :



insert (x, first) enter at first end

1. [Underflow]
   If Avail = Null
   then write ("underflow") ; Return (first)
2. [obtain add^{n} of next free node]
   New ← Avail
3. [remove free node from availability stack]
   Avail ← link (avail) (next to avail)
4. [initialize fields of new node and its link to the list]
   info (New) ← x
   link (New) ← First
5. [return add^{n} of new node]
   return (New)

link (new) ← first

- it's malloc func → memory allocation in C.

Insert (x, first) enter at last end

1. If Avail = Null
   " Underflow"
2. New ← Avail
3. Avail ← link (avail)
4. info (New) ← x
   link (new) ← null
5. if first = Null
   then return (new)
6. save ← first

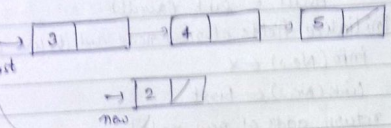
Subject .....

School / College .....

7. Repeat while link(save) ≠ Null
save ← link(save) (for next step)
8. link(save) ← New
9. return (first)

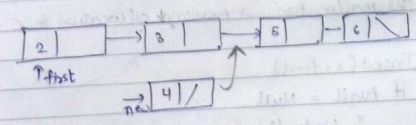
Insert ord(x, first)

I₀



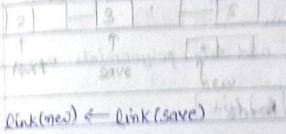
new(info) ≤ first(info)
link(new) ← first
first ← new

II₀



if info(new) > info(first)
save ← first
while link(save) ≠ Null &
info(save) < info(new)
info[link(save)] (it means value is 3)

1. save ← first
2. while link(save) ≠ Null &
info(new) > info(link save)

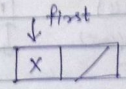


link(new) ← link(save)
link(save) ← new

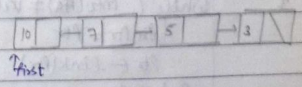
Delete:

Delete(x)

1. if info(first) = X &
link(first) = Null
then first ← null



2.



- from front end

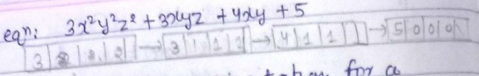
1. if(start == null)
"underflow"
2. Ptr = start
3. while start = link(Ptr)
4. free Ptr
exit

- from last node

2. Ptr = start, PrePtr = Ptr
3. while(link(Ptr) ≠ null)
PrePtr = Ptr
Ptr = link(Ptr)
endwhile
4. link(PrePtr) = null
free(Ptr)
5. exit

Subject .....
School / College .....

Polynomial eqn: $3x^2y^2z^2 + 3xyz + 4xy + 5$



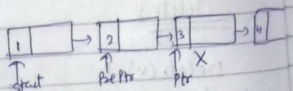
① Create a linked list representation for a polynomial equation

② Write an algo to add two polynomials using link list

Coff.	Px	Py	Pz	Link

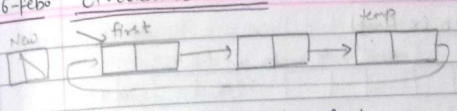
(For addition we compare the power of x, y, z and add)

Delete after a given node



1. If (Start == null)
   print ("Underflow")
2. Ptr = Start
   PrePtr = Ptr
3. While (info(Ptr) ≠ value)
   PrePtr = Ptr
   Ptr ← Link(Ptr)
   End while
4. temp = Link(Ptr)
   Link(PrePtr) → temp
   Free Ptr
5. exit

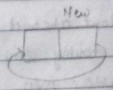
16-Feb Circular link list: insert(x, first)



5) Link (new) ← first
   temp ← first
   Repeat while ~~Link(temp)~~ Link(temp) ≠ first
   temp ← Link(temp)
   Link(temp) ← new
   first ← new → when we insert at the end point this statement is not true

Algorithm:

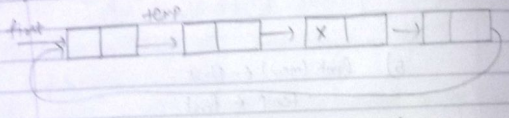
1) Get new node
   If Avail = Null
   print "No free space"
   return (first)
2) New ← Avail
   Avail ← Link(Avail)
3) info(new) = x
4) If first = Null
   Link(new) ← new
   Return (New)



Subject .....
School / College .....

Circular link list

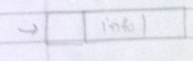
⇒ Delete (x, first)



link (temp) ← link (link (temp))

22-Feb

Doubly link list :

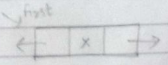


Equivalent structure in C:

```
Struct node
{
  int info;
  struct node * left * right
};
```

Equivalent abstract data struc:
Node doubly link list
variable : info
Link : right, left

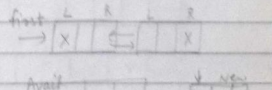
⇒ Insert (first, x)

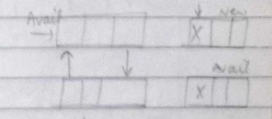


Step 1: if ~~first~~ avail == Null
~~print " DLL empty"~~

Step 1: if avail == Null
print (" No more free nodes)
Return (first)

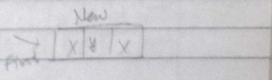
Step 2: new ← avail
avail ← Right(avail)
left(avail) ← Null



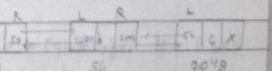


Step 3: info(new) ← x
right(new) ← null
left(new) ← null

Step 4: if First = Null
First ← new
return (First)

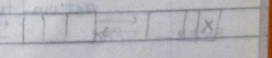


- X to be inserted after Y ;



insert (first, x, y) after that we have to insert x

Step 5: temp ← first
Step 6: repeat info(temp) # y and right(temp) # Null
temp ← Right (temp)

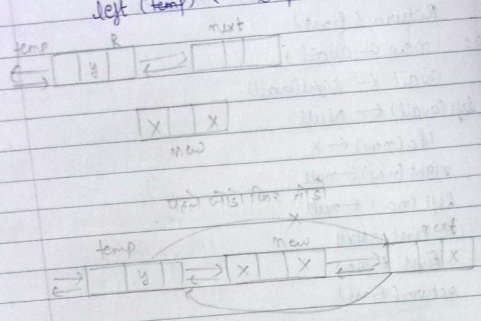


Step 7: If Right (temp) == Null and info(temp) # y
print " Y not found"
return (first)

Step 8: if info(temp) == y and temp(Right) == Null

Subject .....
School / College .....

right (temp) ← new
left (new) ← temp



Step 9: if info(temp) = = y & right (temp) ≠ null
next ← right (temp)
left(new) ← temp
right(temp) ← next
left (next) ← new
right (temp) ← new
return (first)

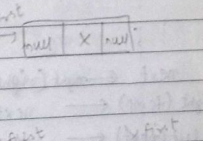
Step 10: End

⇒ Delete in doubly link list :

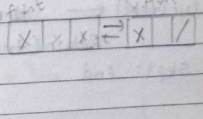
Delete (first, x)

Step 1: if first = Null
print "list empty"
return (false)

Step 2: if info(first) = x
if Right (first) = = Null
first ← null
return (True)

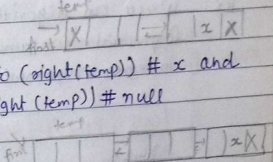


else
first ← right (first)
left (first) ← Null
return (True)



Step 3: temp ← first

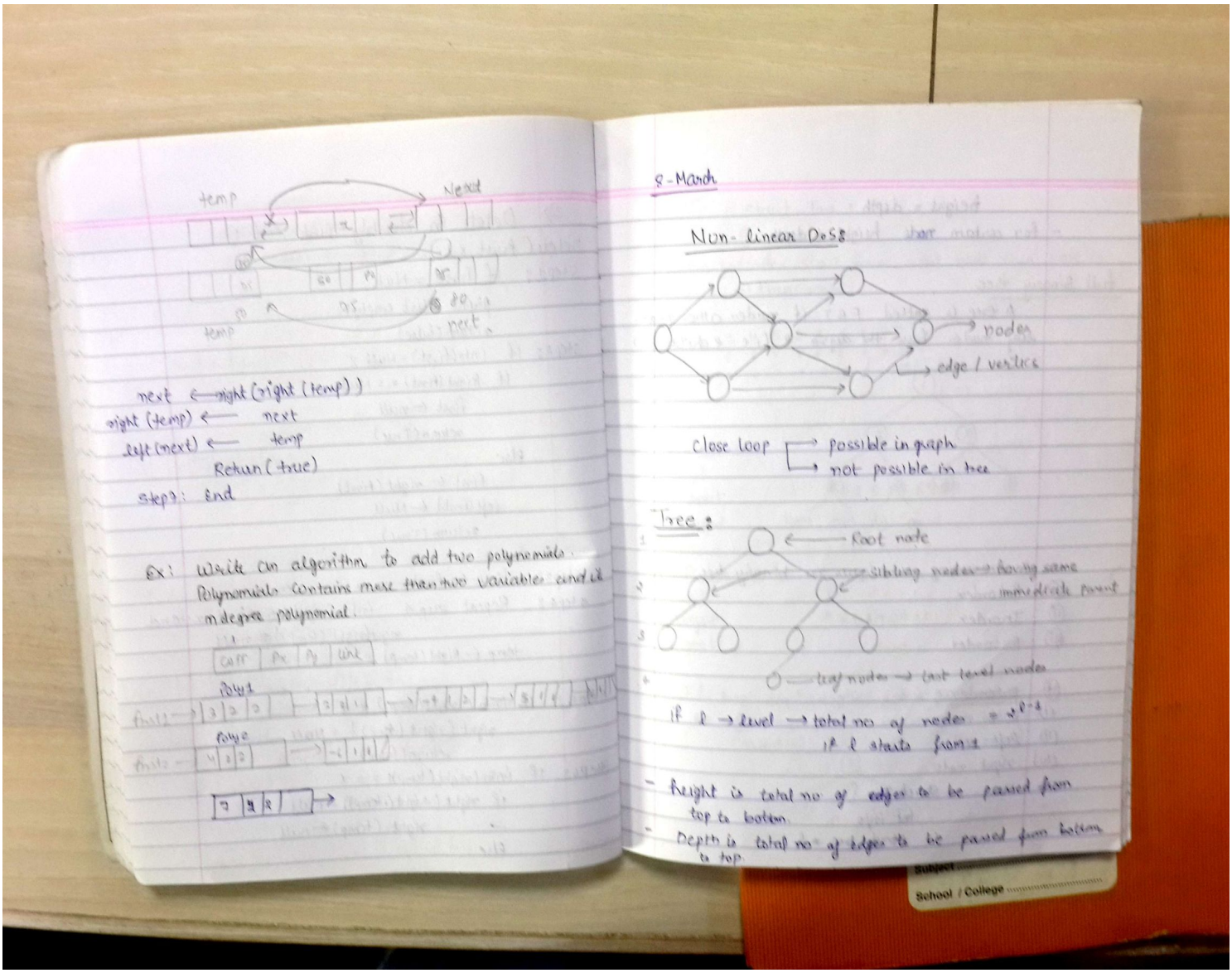
Step 4: Repeat step 4 info (right(temp)) ≠ x and right(right (temp)) ≠ null
temp ← Right (temp)



Step 5: if info (right(temp)) ≠ x
right (right (temp)) = Null
return (false)

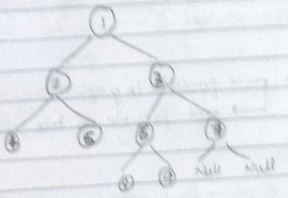
Step 6: if info (right (temp)) = = x
if right (right (temp)) = null
right (temp) ← null
else

Subject .....
School / College .....



height = depth
- for certain node height = depth

Full Binary Tree
A tree is called FBT if nodes other than leaf nodes have the degree 2 (i.e. 2 children)



3 traversing techniques in binary tree
(I) Preorder
(II) Inorder
(III) Postorder

(1) preorder :
(i) Root node
(ii) Left node
(iii) right node

struct Tree {
int info
struct tree *l ;

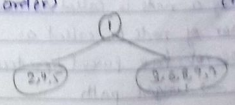
struct Tree * r;
} *node;

In preorder start from root node then move in left → then right

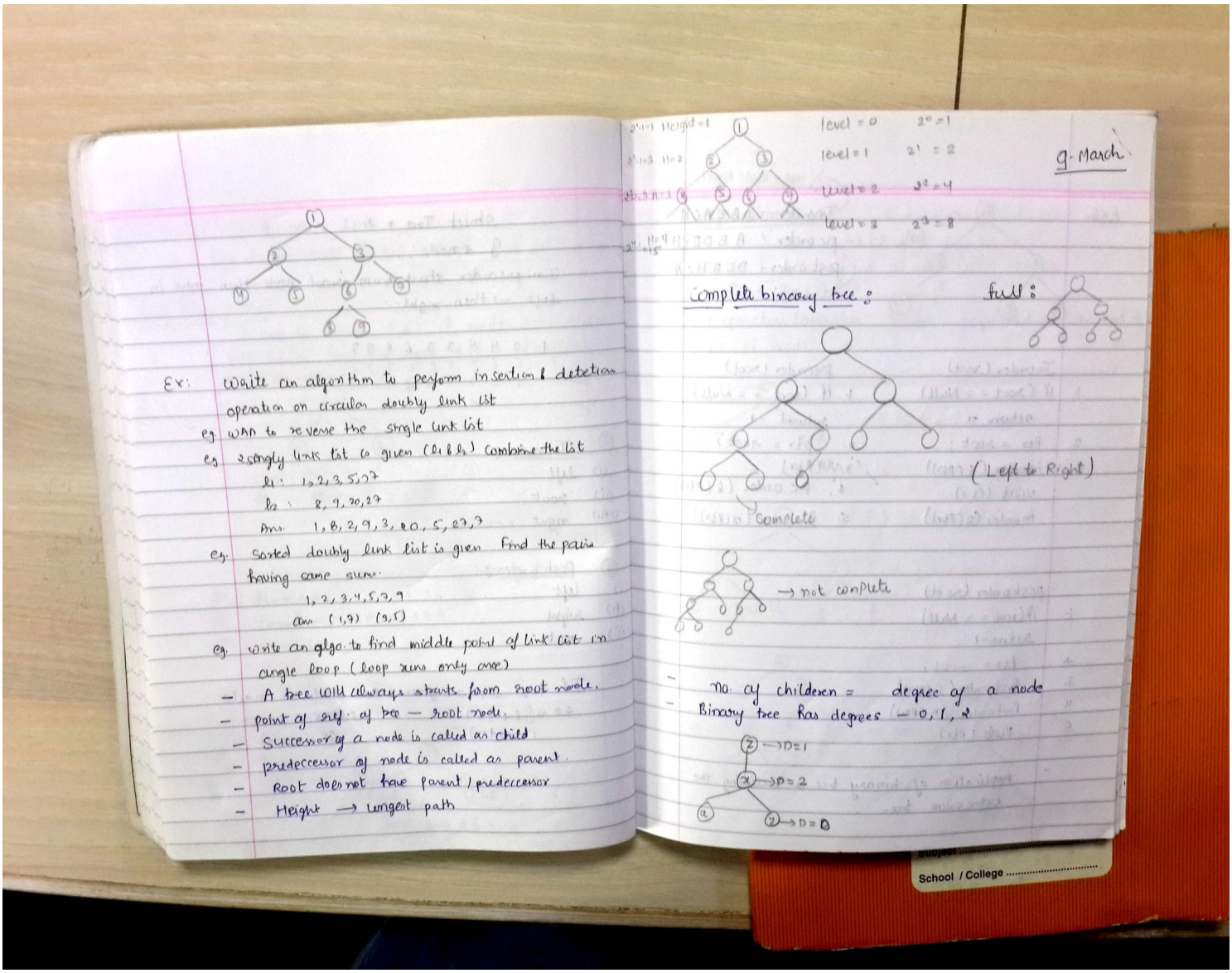
1 2 4 5 3 6 8 9 7
(2) In order :
(i) left
(ii) root
(iii) right

(3) Post order :
(i) left
(ii) Right
(iii) root

If two seq are given & tree is to be made
42 5 1 8 6 9 3 7
(Inorder)
1 2 4 5 3 6 8 9 7
(Pre order)



Subject .....
School / College .....



Ex: Write an algorithm to perform insertion & detetion operation on circular doubly link list

eg WAP to reverse the single link list

eg 2 singly link list is given (l_1 & l_2) combine the list

l_1 : 1, 2, 3, 5, 7

l_2 : 8, 9, 20, 27

Ans. 1, 8, 2, 9, 3, 20, 5, 27, 7

eg. Sorted doubly link list is given Find the pairs having same sum.

1, 2, 3, 4, 5, 7, 9

Ans (1,9) (3,5)

eg. write an algo. to find middle point of link list in single loop (loop runs only once)

- A tree will always starts from root node.
- point of ref. of tree — root node.
- Successor of a node is called as child
- predeccessor of node is called as parent.
- Root does not have parent / predeccessor
- Height → longest path

level = 0 $2^0 = 1$

level = 1 $2^1 = 2$

level = 2 $2^2 = 4$

level = 3 $2^3 = 8$

9 March

complete binary tree :

full :

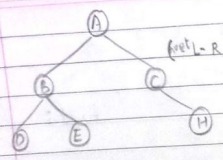
(Left to Right)

Complete

→ not complete

- no. of childeren = degree of a node
- Binary tree has degrees — 0, 1, 2

Ex:



(left - Root - Right)

Inorder : D B E A C H

preorder : A B D E C H ← (root L-R)

postorder : D E B H C A

(L-R-root)

Inorder (root)
1. if (root == Null)
   return
2. Ptr = root;
3. inorder (L(Ptr))
   visit (Ptr)
   inorder (R(Ptr))

preorder (root)
1. if (root == Null)
   return -1
2. ptr = root;
3. visit(Ptr)
4. Pre order (L(Ptr))
5. Pre order (R(Ptr))

postorder (root)
1. if(root == Null)
   return-1
2. Ptr = root;
3. Postorder(L(Ptr))
4. Postorder (R(Ptr))
5. Visit (Ptr)

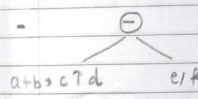
- Application of binary tree is building the expression tree.

Expression tree:

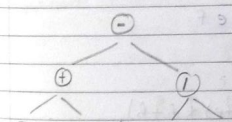
a + b * c ↑ d - e / f
4 3 2 1 5 3

The operation/operator to be performed at the end → root of node.

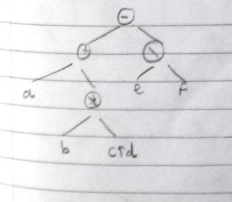
-



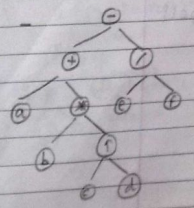
-



-



-



School / College .....

- If this expression tree is traversed in post order then we get the postfix expression and same for the inorder ⟶ infix
pre order ⟶ Prefix

postorder : (left - right - root)

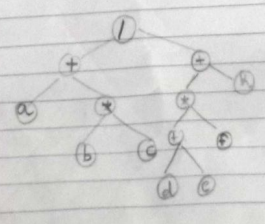
abcd↑*+ef/-

pre order : (root - left - right)

-+a*b↑cd/ef

Ex: create expression tree?
(a↑b*c)/(d/e)*f↑h)
do pre and post order traversal?

Sol:



15-March

(Root-Left-R) Preorder : /+a*bc+de*+*+defh

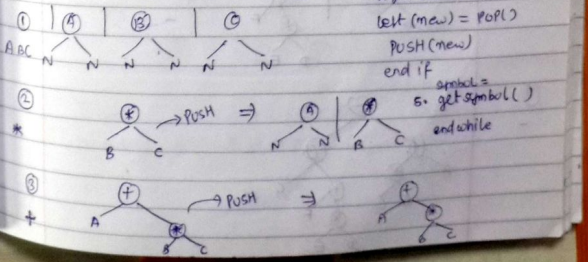
(Left-R-Root) Postorder : abc*+de+f*h+/

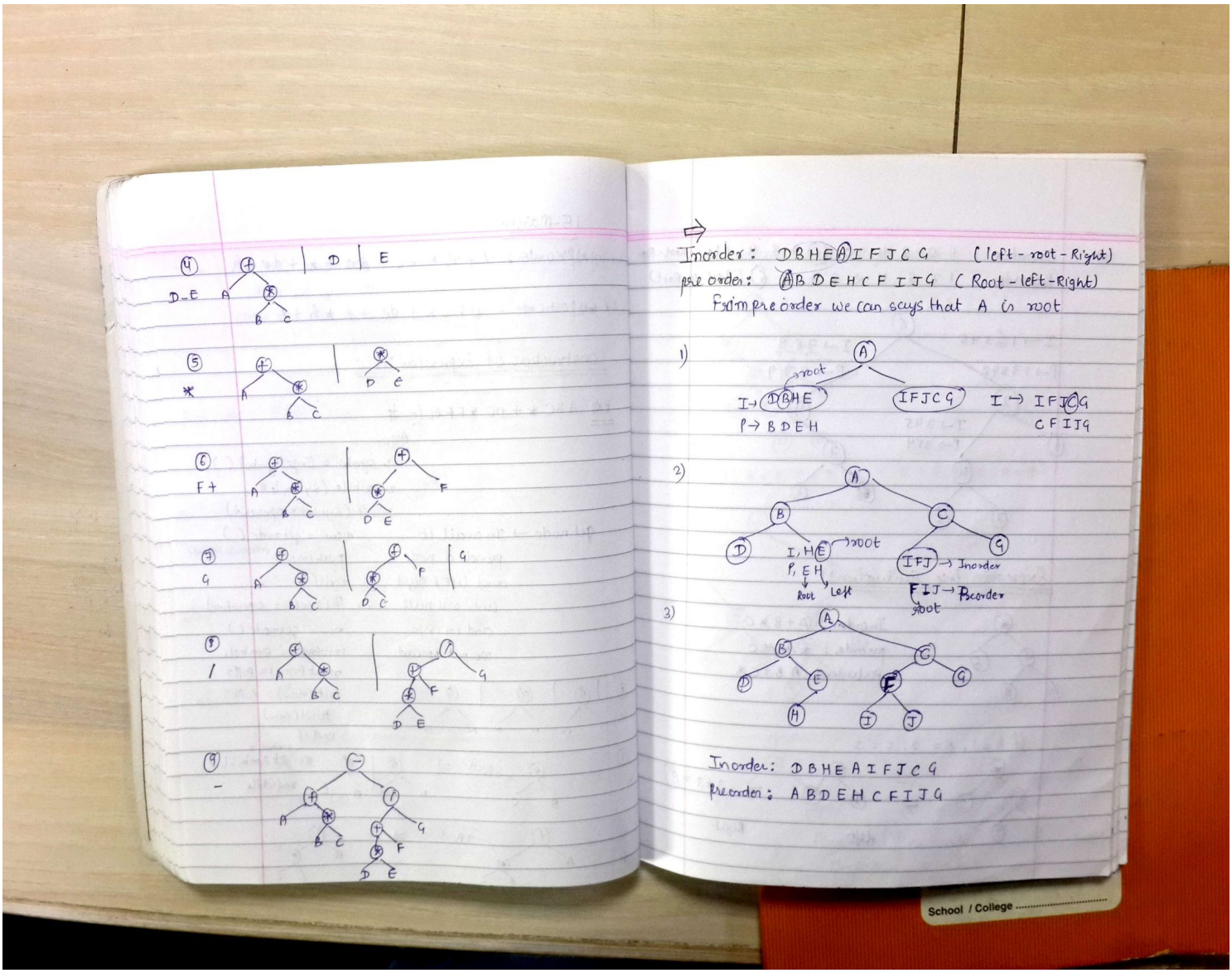
Construction of Expression Tree

Ex: ABC*+DE*FF G1/-#

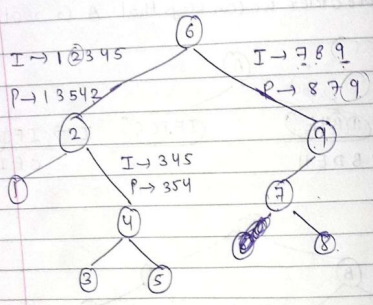
Algo:
1. symbol = E.getsymbol();
2. while (symbol ≠ '#')
3. if (symbol = operand)
new = getnode()
info(new) = symbol,
push(new)
endif
4. if (symbol = Operator)
new = getnode()
info(new) = symbol,
right(new) = POP()
left(new) = POP()
PUSH(new)
endif
5. symbol = getsymbol()
endwhile

getnode → In avail it provide a node and left & Right link will null and in info we put operand.

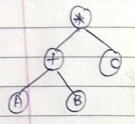




⇒ Inorder : 1 2 3 4 5 (6) 7 8 9 (Left-Root-R
Postorder : 1 3 5 4 2 8 7 9 (6) (Left-R-Root)

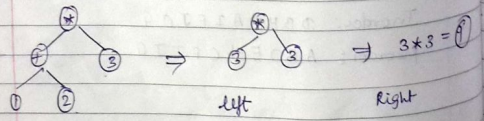


Expression tree Evaluation:



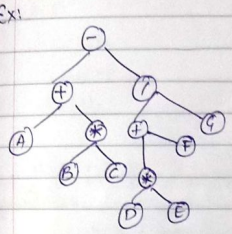
Inorder : (A + B) * C
preorder : * + A B C
postorder : A B + C *

if A = 1, B = 2, C = 3

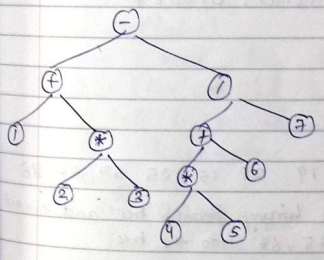


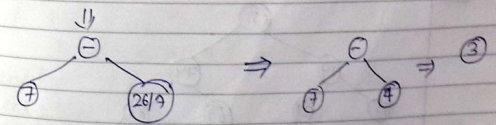
Evaluation: Left - Right - Root

Ex:



A = 1, B = 2, C = 3, D = 4, E = 5, F = 6, G = 7



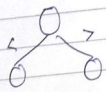


floatvalue / ceilvalue
0 - 0.5 0.5 - 1

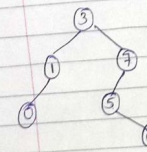
School / College .....

Binary search tree:

→ all left are less
→ all left are greater



Ex: 3 1 7 5 0 6

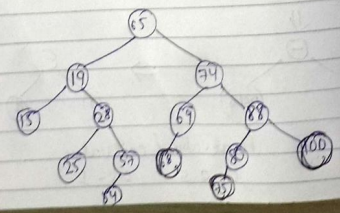


inorder: 0 1 3 5 6 7

Ex:

a) 65 74 69 19 28 15 25 57 54 88 80 Construct a binary search tree and insert element 100, 75, 68 into the tree.

Sol:



Binary search tree search operations:

Create:

i) Search
ii) Insert
iii) Delete

- root pointer is very imp in BST tree.

Algorithm:

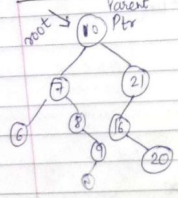
```
Search (root, key)
if (root == null)
  print "Empty BST"
return (false)
ptr = root, flag = false
while (Ptr ≠ null && flag = false)
 if info(Ptr) == key
      flag = true;
 endif
if info(Ptr) > key
   ptr ← left(Ptr)
endif
if key > info(Ptr)
   Ptr ← right(Ptr)
endif
end while
if flag = True
   print found
```

Subject .....
School / College .....

16-March

else
print not found
end if
return

BST insertion Algorithm



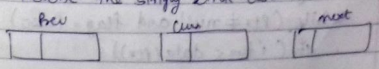
We need a parent pointer and traverse pointer also.

Insert (root, key)
1. if (root = = null)
new = get node → 

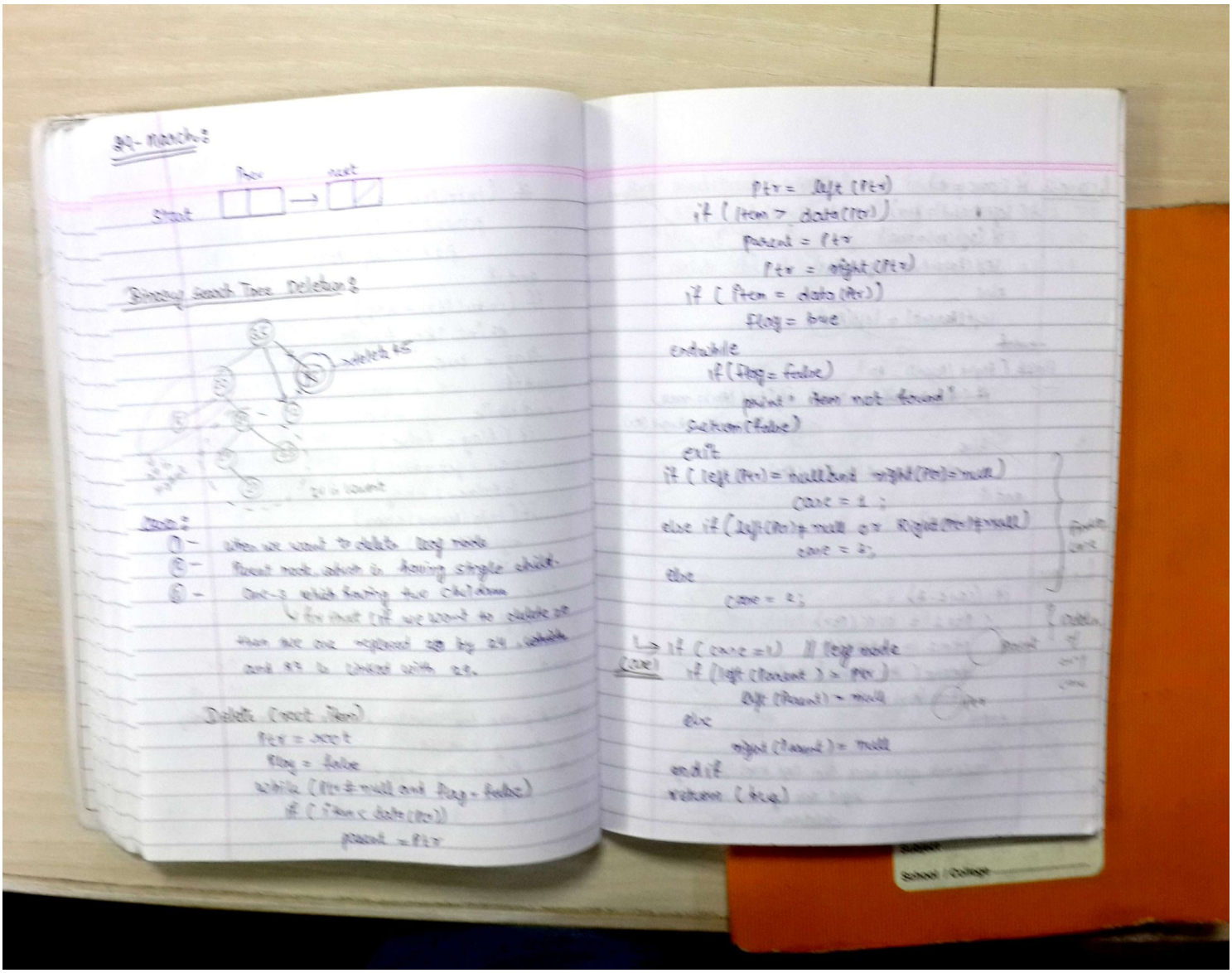
info(new) = key
root ← new
return (tree)
2. Ptr ← root
Parent ← root
while (Ptr ≠ null & flag = false)
if (info(Ptr) < key)
parent ← root ptr
ptr ← right (Ptr)
end if

if (info(Ptr) > key)
parent ← ptr
ptr ← left(Ptr)
end if
if (info(Ptr) = key)
print "key found"
flag = true
end if
end while
if (flag = false) → Yhn Pr Ptr = NULL, so we have to check whether the new value (key) is going to left or right to the Parent node.
new = get node( )
info(new) = key
if (Info (Parent) < Key)
right (Parent) = new
end if
else
left (Parent) = new
end if
End

Ex- ① Algorithm for insertion & deletion for circularly doubly link list.
② Reverse the singly link list.



School / College .....



↳ Case-2 if (case = 2)

if (left(Parent) = Ptr)

if (left(Ptr) = null)

left(Parent) = right(Ptr)

else

left(Parent) = left(Ptr)

~~endif~~

else if (Right (Parent) = Ptr)

~~if (Right (Parent) = Null)~~ if (left (Ptr) = null)

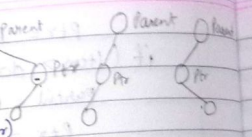
Right(Parent) = Right (Ptr)

else

right (Parent) = Left (Ptr)

endif

return (true)



↳ Case-3

if (case-3)

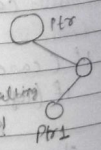
Ptr1 = succ (Ptr)

item1 = data (Ptr1)

delete (Ptr, item1) → from calling of case1 or case2

data (Ptr) = item2

endif



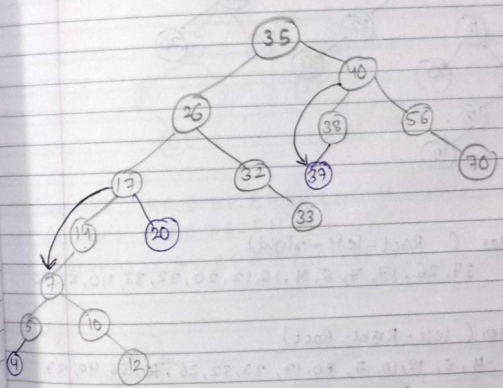
succ → gives back the left most child of right side of the node to be deletion.

Ex8 construct a BST for the following step

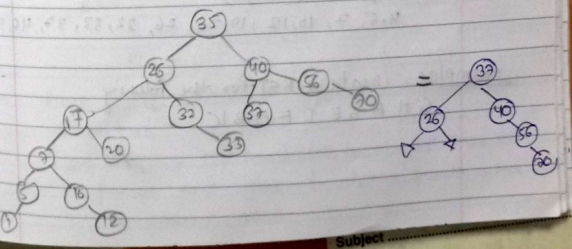
35, 26, 32, 70, 14, 40, 38, 56, 33, 7, 5, 10, 12, 70

insert the following element for : 4, 37, 20

delete the following elements : 14, 38, 35



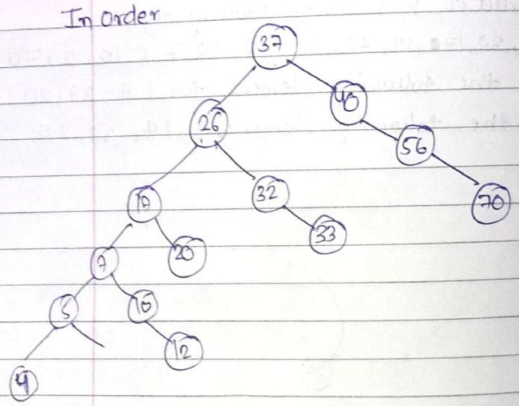
after deletion of 14 & 38



Subject .....

School / College .....

4! In order



Preorder (Root - left - right)

37, 26, 17, 7, 5, 4, 10, 12, 20, 32, 33, 40, 56, 70

postorder (left - Right - Root)

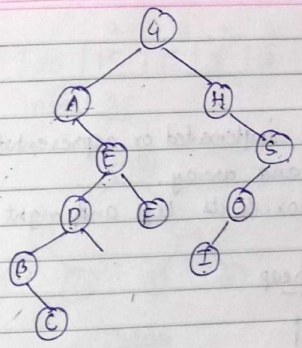
4, 5, 12, 10, 7, 20, 17, 33, 32, 26, 70, 56, 40, 37

Inorder (left - Root - Right)

4, 5, 7, 10, 12, 17, 20, 26, 32, 33, 37, 40, 56, 70

Example: Create a BST for the following

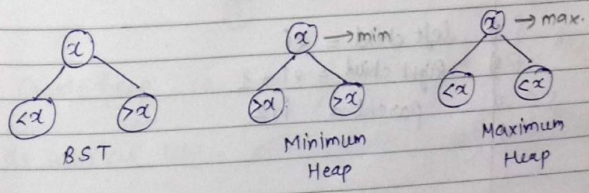
G H A S E D F O B I C



Inorder: ~~B C D E F~~ A B C D E F G H I O S

Preorder: G A E D B C F H S O I

Postorder: C B D F E A I O S H G



- heap is a different type of data structure
- In BST, left hand side is always less to parent and RHS is always greater value to parent root.

Subject .....

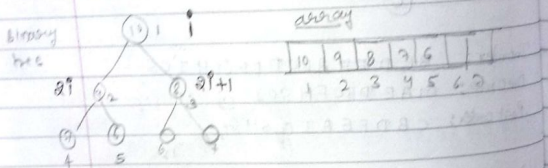
School / College .....

30-March

Max Heap:

- heap can be implemented or represented in the linked list and array.
- root is max. to its left and right children.

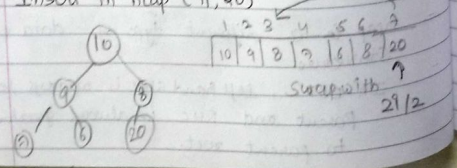
Array to store heap



Child { 1) Left child = 2i
2) Right child = 2i + 1

Parent { 3) Parent = i/2

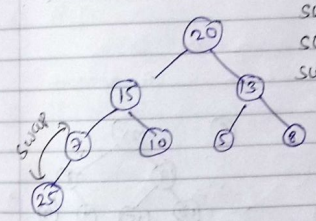
Insert in heap (H, 20)

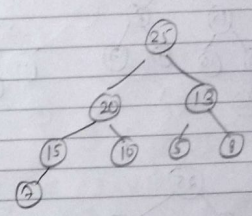


Ex:- 1 2 3 4 5 6 7 8

20	15	13	7	10	5	8	25

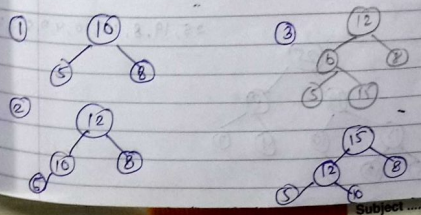
insert 25





Ex:- Create heap 10, 5, 8, 12, 15, 4, 7, 19, 25, 2

- as soon as large element comes, start comparing



Subject .....

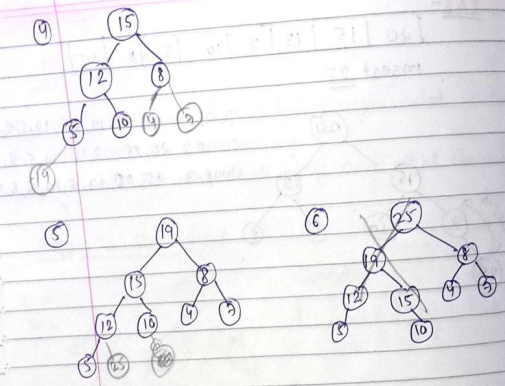
School / College .....

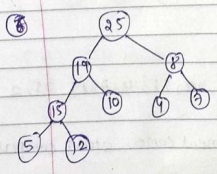
5-April

Delete in heap

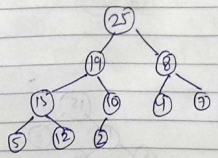
- Extract max (H)

① if (N = 0)
    print ("No element")
    return
② if (N = 1)
    N = 0
    return
③ Swap (A(1), A[N])
    N = N-1
④ i = 1
⑤ l = 2
    r = 2i + 1
⑥ ~~if A[l] > Af~~ if (l ≤ N)
        if (r ≤ N)
            if (A[l] > A[r])
                greatest = l
            else
                greatest = r
        else
            greatest = l
⑦ if (A[i] < A(greatest])
    Swap (A[i], A[greatest])
    i = greatest
    repeat from Skp ⑤





25, 19, 8, 15, 10, 4, 2, 5, 12

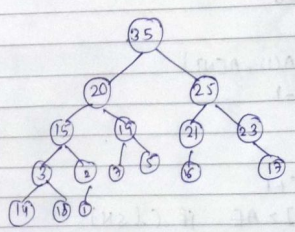


Subject .....
School / College .....

(5) else
print (" Out of bound)
Stop

Example: elements of the array are 35, 20, 25, 15, 19, 21, 23, 3, 2, 7, 5, 16, 17, 14, 18, 1 trace the algo for extract max. func

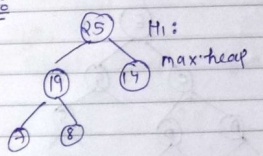
Sol:

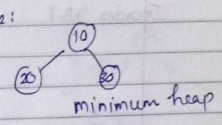


i=	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	35	20	25	15	19	21	23	3	2	7	5	16	17	14	18	1
		i		l		r										
	1	20	25	15	19	21	23	3	2	7	5	16	17	14	18	
	25	20	1	15	19	21	23	3	2	7	5	16	17	14	18	
			i			l	r							l	r	
	25	20	23	15	19	21	1	3	2	7	5	16	17	14	18	
							i							l	r	
Ans	25	20	23	15	19	21	18	3	2	7	5	16	17	14	1	

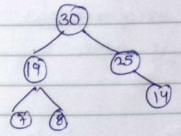
Ex: Assume two heaps h_1 with 5 nodes and h_2 with 3 nodes. Perform merge operation on h_1 and h_2. h_1 is max heap and h_2 is min heap. The resultant heap is same as which heap is retain.

Sol:

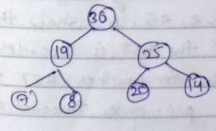




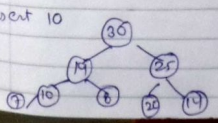
- insert 30



- insert 20



- insert 10



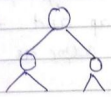
max. heap

School / College .....

6-April

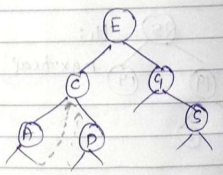
Neha Mothe

Threaded Binary Trees



nodes	null
3	4
1	2

Inorder TBT



N(?) → joined to predecessors of node
N → joined to successors of node

Ex: Create a BST for the following nodes 15, 17, 7, 5, 20, 13, 2, 25, 6. Show the thread structure if the BST is inorder TBST. (ii) also show in another BST having the same elements a pre-order threaded BST

Graphs

6-April

Graph is the set of nodes connected at the edges.

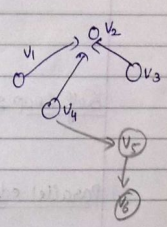
vertex set V = { $V_1, V_2, \ldots V_n$ }

connected by

edge set E = { $(V_1, V_2), (V_2, V_4), (V_2, V_3), \ldots$ }

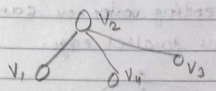
Types

① Directed graph / Digraph



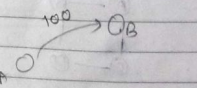
② Undirected graph

edges does not have direction.



③ Weighted graph

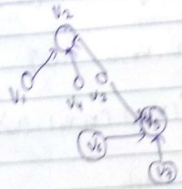
- A numerical value is associated b/w 2 nodes or the edge.
- numerical value can be — distance



School / College .....

Adjacent vertices :

$V_2 \rightarrow V_1, V_4, V_3$

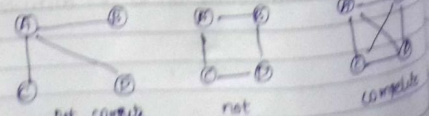


Self loop : Starting and ending vertex are the same

Parallel edge : Starting vertex are same and ending vertex are same, but starting & ending, traits parallel edge.

$e_1, e_2, e_3 \rightarrow$ parallel
$e_4 \rightarrow$ self

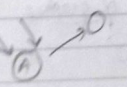
complete graph :



not complete | not comp. | complete

directed Edge { indegree / out degree

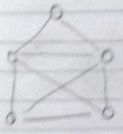
indegree = 2
no of edge emerging on the vertices



Outdegree = 1

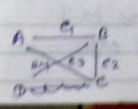
- degree for Undirected  = 3 = degree

- degree = N-1 for complete graph while N → no. of nodes



no self loops are present in complete loop graph.

Disconnected graph :
which are not connected



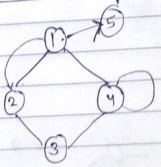
E and F are can't reached

12-April

it can be connected if FE C B A B D → connected graph

Graph traversals

Representation of graph:



i) Sets V = { 1, 2, 3, 4, 5 }

E = {(1,2) (2,1)} ① → ② (1,2)
① ← ② (2,1)

ii) Adjancy matrix

	1	2	3	4	5
1	0	1	0	1	1
2	1	0	1	0	0
3	0	1	0	1	0
4	1	0	1	0	0
5	1	0	0	0	0

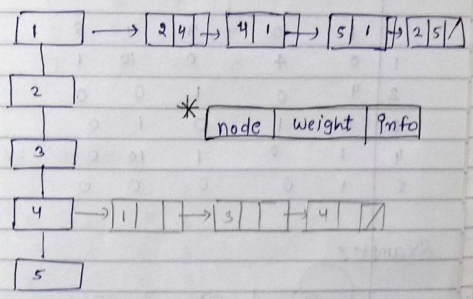
[1 row, 5th columns] → ① → ⑤

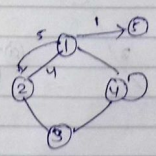
self loops

By adjancy matrix we can't represent the parallel edges

— Yes and No, how many yes or no but we can't shown their ~~wait~~ weight or efficient By this matrix.

iii) link list

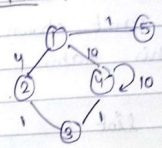




— directive and non-directive should not be at a same time on graph. Either It will be Directed or non-directed.

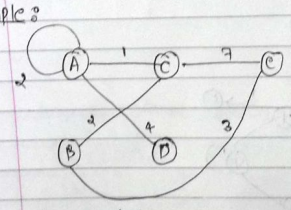
School / College .....

Weighted adjacency matrix



	1	2	3	4	5
1	0	4	0	10	1
2	4	0	1	0	0
3	0	1	0	1	0
4	1	0	1	10	0
5	1	0	0	0	0

Example:



(a,b,4)

(a) Represent in set

V = { A, C, E, B, C, D }

= { (A,C,1), (A,D,4), (A,A,2), (C,B,2), (C,E,7), (E,B,3) }

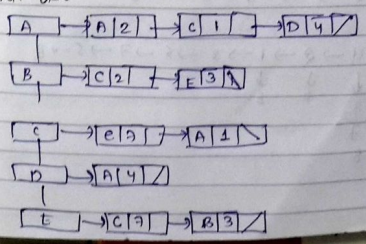
(b) Adjacency

	A	B	C	D	E
A	1	0	1	1	0
B	0	0	1	0	1
C	1	1	0	0	1
D	1	0	0	0	0
E	0	1	1	0	0

(c) Weighted adjacency

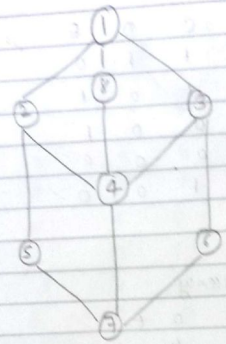
	A	B	C	D	E
A	2	0	1	4	0
B	0	0	2	0	3
C	1	2	0	0	7
D	2	0	0	0	0
E	0	3	7	0	0

(d) Link list

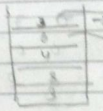


School / College .....

Depth first Search (DFS):



1 → 2 → 5 → 7 → 4 → 8 → 3 → 6



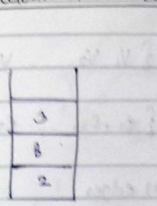
1 - 2 - 5 - 7 - 6 - 3 - 4 - 8

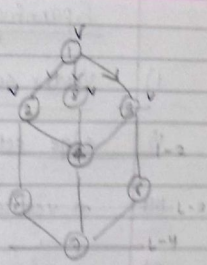
Algo:

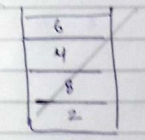
4 → 2 → 1 → 3 → 6 → 7 → 5 → 8

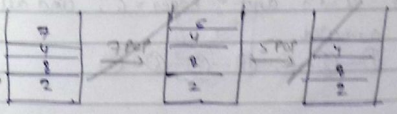
3
7
8

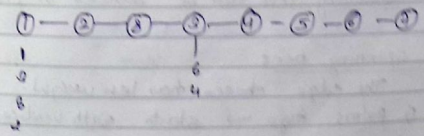
Breadth first Search:

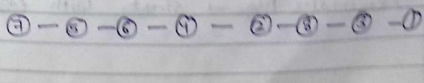












Spanning Tree:

i) Vertex set $V = \{V_1, V_2 \dots V_n\}$

ii) edge set $E = \{e_1, e_2, \dots e_{n-1}\}$

iii) n vertices and (n-1) edges

iv) no cycle

v) no parallel edges

vi) connected graph

Ex: if $V \to 7$ and edges = 6, and all are connected

① — ② — ③ — ④ — ⑤ — ⑥ — ⑦

- spanning tree can be directed or undirected.

Min. spanning tree:

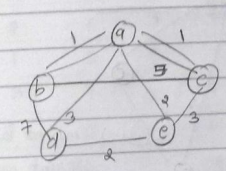
The edge chosen has less weight.

i) prims Algo → starts with vertices

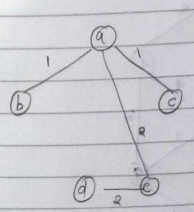
ii) krushkal's Algo → starts with edge

13-April

Krushkal:



a-c	1	b-c	5
a-b	1	b-d	7
a-e	2	d-e	2
a-d	3	c-e	3



5 → vertex
4 → edges

PRIMS:

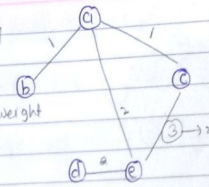
Best way to choose and use vertex is by Adjacency matrix.

	a	b	c	d	e
a	0	1	1	3	2
b	1	0	5	7	0
c	1	5	0	0	3
d	3	7	0	0	2
e	2	0	3	2	0

do not choose as it forms self-loop

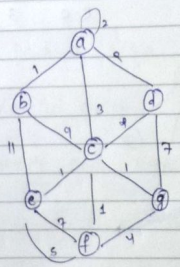
School / College .....

Start from any vertex eg:-b
then choose an edge with less weight
visit the vertex chosen there

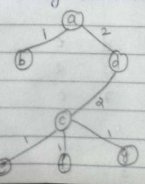


3 → removed as a-e has less weight compared to c-e

total weight = 1+1+2+2 = 6



PRIMS : Starting vertex : b



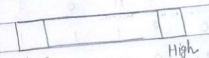
	a	b	c	d	e	f	g
a	2	1	3	2	0	0	0
b	1	0	9	0	11	0	3
c	3	9	0	2	1	1	1
d	2	0	2	0	0	0	7
e	0	11	1	0	0	5	0
f	0	0	1	0	5	0	4
g	0	3	1	7	0	4	0

Subject .....

School / College .....

19-April

① Quick Sort



→ Binary Search
⇒ (. it should be sorted)

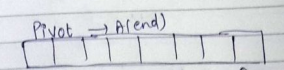
2	4	16	25	30

$x = 5$, $n = 5$

Algo:

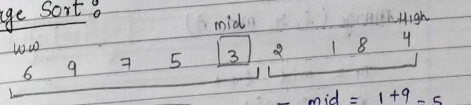
```
search( A, n, x )
for i = 0 to n-1, i++
  if A(i) = x
    return i
  else
    return -1
```

-



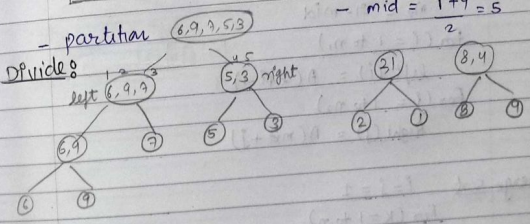
- Partition

② Merge Sort :



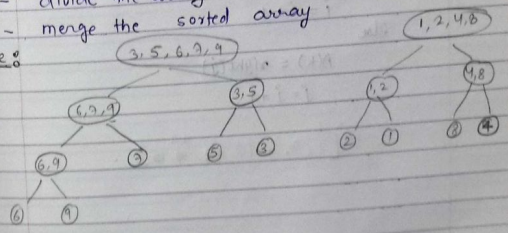
- partition
- mid = $\frac{1+9}{2} = 5$

Divide :



- divide the array
- merge the sorted array

Merge :



Subject .....
School / College .....

ALGO:

Merge (l, r, mid, A)

$n_1 = mid - l + 1$

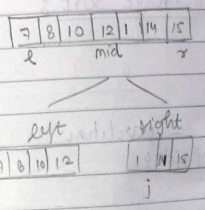
$n_2 = r - mid$

For ($i = 1$ to n_1)

  left (i) = A(l + i - 1)

For ($j = 1$ to n_2)

  Right (j) = A(mid + j)



Merge sort i = j = 1

For (k = 1 to n)

  if left(i) <= right(j)

    A(k) = left(i)

    i = i + 1

  else

    A(k) = right(j)

    j = j + 1

A[1, 7, 8, 10, 12, 14, 15] (k = 1 ... 12)

③ Heap Sort:

Subject .....

School / College .....