PAGE NO. :
DATE : / /

DSA

UNIT-4

Bubble sort :-

In bubble sort, each element is compared with it adjacent element. If first element is larger than the second element than the position of the element are interchanged otherwise it is not changed.

(1) 10 14 2 11 6

Path(1)

10 14 2 11 6

10 14 2 11 6

10 14 11 6

10 2 14 14 6

10 2 14 6 14

Path(2) → 10 2 11 6 14

2 10 11 6 14

2 10 11 6 14

2 10 6 11 14

Path(3) → 2 10 6 11 14

2 10 6 11 14

2 6 10 11 14 (ext)

2 6 10 11 14

2 6 10 11 14

Path-1

Ex (2) 4 1 5 6 2

(4) 1 5 6 2

4 (4) 5 6 2

1 4 (5) 6 2

1 4 (5) 6 2

1 4 5 (6) 2

1 4 5 2 6

Path (2)

(1) 4 5 2 6

1 (4) 5 2 6

1 4 (5) 2 6

1 4 2 (5) 6

1 4 2 5 6

Path (3)

(1) 4 2 5 6

1 (4) 2 5 6

1 2 4 5 6

Ex (3) 32, 51, 27, 85, 66, 23, 13, 15, 57

Path (32) 51, 27, 85, 66, 23, 13, 15, 17

32 (51) 27 85 66 23 13 75, 57

32 27 (51) 85 66 23 13 75, 57

32 27 51 (85) 66 23 13 15 57

32 27 51 66 (85) 23, 13 15 57

32 27, 51, 66 23 (85) 13 15, 57

32 27 51 66 23 13 (85) 15 57

32 27 51 66 23 13 15 (85) 57

32 27 51 66 23 13 15 57 85

PAGE NO.:
DATE: / /

Path (2) :-

(32) 27, 51, 66, 23, 13, 15, 57, 85

27 (32) 51 66 23 13 15 57 85

27 32 (51) 66 23 10 15 57 85
27 32 51 (66) 23 13 15 57 85
27 32 51 23 (66) 13 15 57 85
27 32 51 23 13 (66) 15 57 85
27 32 51 23 13 15 (66) 57 85
27 32 51 23 13 15 57 (66) 85

Path (3)

(27) 32 51 23 13 15 57 66 85
27 (32) (51) 23 13 15 57 66 85
27 32 (51) 23 13 15 57 66 85
27 32 23 (51) 13 15 57 66 85
27 32 23 13 (51) 15 57 66 85
27 32 23 13 15 (51) 57 66 85

Path (1)

(27) 32 23 13 15 51 57 66 85
27 (32) 23 13 15 51 57 66 85
27 23 (32) 13 15 51 57 66 85
27 23 13 (32) 15 51 57 66 85
27 23 13 15 (32) 51 57 66 85

Path (5)

(27) 23 13 15 32 51 57 66 85
23 (27) 13 15 32 51 57 66 85
23 13 (27) 15 32 51 57 66 85
23 13 15 27 32 51 57 66 85

Pathak

23, 13 15 27 32 51 57 66 85

13 23, 15 27 32 51 57 66 85

13 15 23 27 32 51 57 66 85

11. 77, 33, 44, 11, 88, 22, 66, 55

Pass 1

33 77, 44 11 88 22 66 55

33 44 77, 11 88 22 66 55

33 44 11 77, 88 22 66 55

33 44 11 88 77 22 66 55

33 44 11 88 22 77 66 55

33 44 11 88 22 66 77 55

33 44 11 88 22 66 55 77

Pass 2

33, 44 11 88 22 66 55 77

33 44, 11 88 22 66 55 77

33 11 44, 88 22 66 55 77

33 11 44 88, 22 66 55 77

33 11 44 22 88, 66 55 77

33 11 44 22 66 88, 55 77

33 11 44 22 66 55 88, 77

33 11 44 22 66 55 77 88

Pass 3 :- 33, 11 44 22 66 55 77 88

11 33, 44 22 66 55 77 88

11 33 44, 22 66 55 77 88

11 33 22 44, 66 55 77 88

11 33 22 44 55, 66 77 88

11 33 22 44 55 66 77 88

Pass (4)

(11) 33 22 44 55 66 77 88

11 (33) 22 44 55 66 77 88

11 22 (33) 44 55 66 77 88

Selection sort ⇒ $O(n^2)$

(1) 77 33 44 11 88 22 66 55

Pass

11 33 44 77 88 22 66 55

11 33 44 77 88 22 66 55

11 22 33 77 88 44 66 55

11 22 33 44 88 77 66 55

11 22 33 44 77 88 66 55

11 22 33 44 66 88 77

11 22 33 44 55 88 77

(2) 66 33 44 22 55 88 11

22 33 44 66 55 88 11

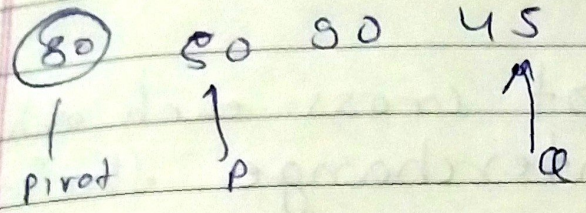
22 33 44 66 55 88 11

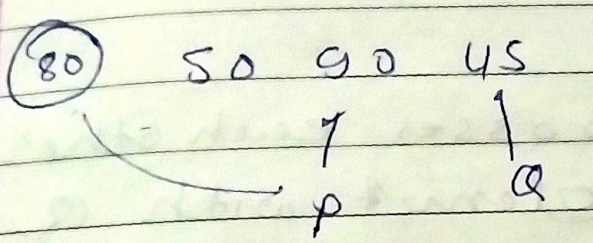
22 33 44 55 66 88 11

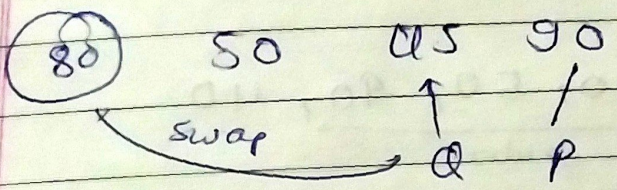
22 33 44 55 66 88 11

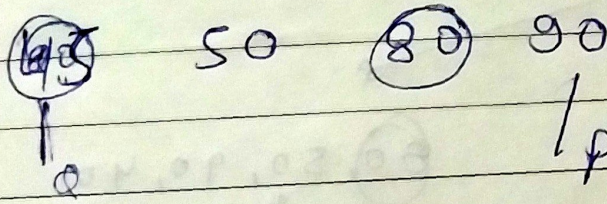
90 31 35 40 61 55 88

Quick short,









45 50 80 90

35 50 15 25 80 20 90 45

Pivot
V:35

Quick short works on devide and conquer root. In this we devide bigger problem into smaller subproblems. P increases or move toward R++ & stop only when it will get the element greater than the pivot element

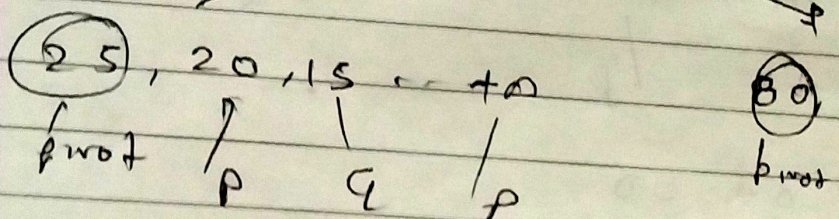
equals to 35 (pivot).
q decreases or move towards L.H.S and stops only when it will get the element lesser the pivot element equal to 35 (pivot)

If P & Q ace not cross each other then swAp or interchange their values.

check(if) P & Q crosses each others then replace pivot element with Q.

25, 20, 15 (35) 80, 50, 90, 40

less — greater



(80) 50 90 40
P Q

15 20 25

(80) 50 40 90
(80) Q P

40 50 80 90

combined as

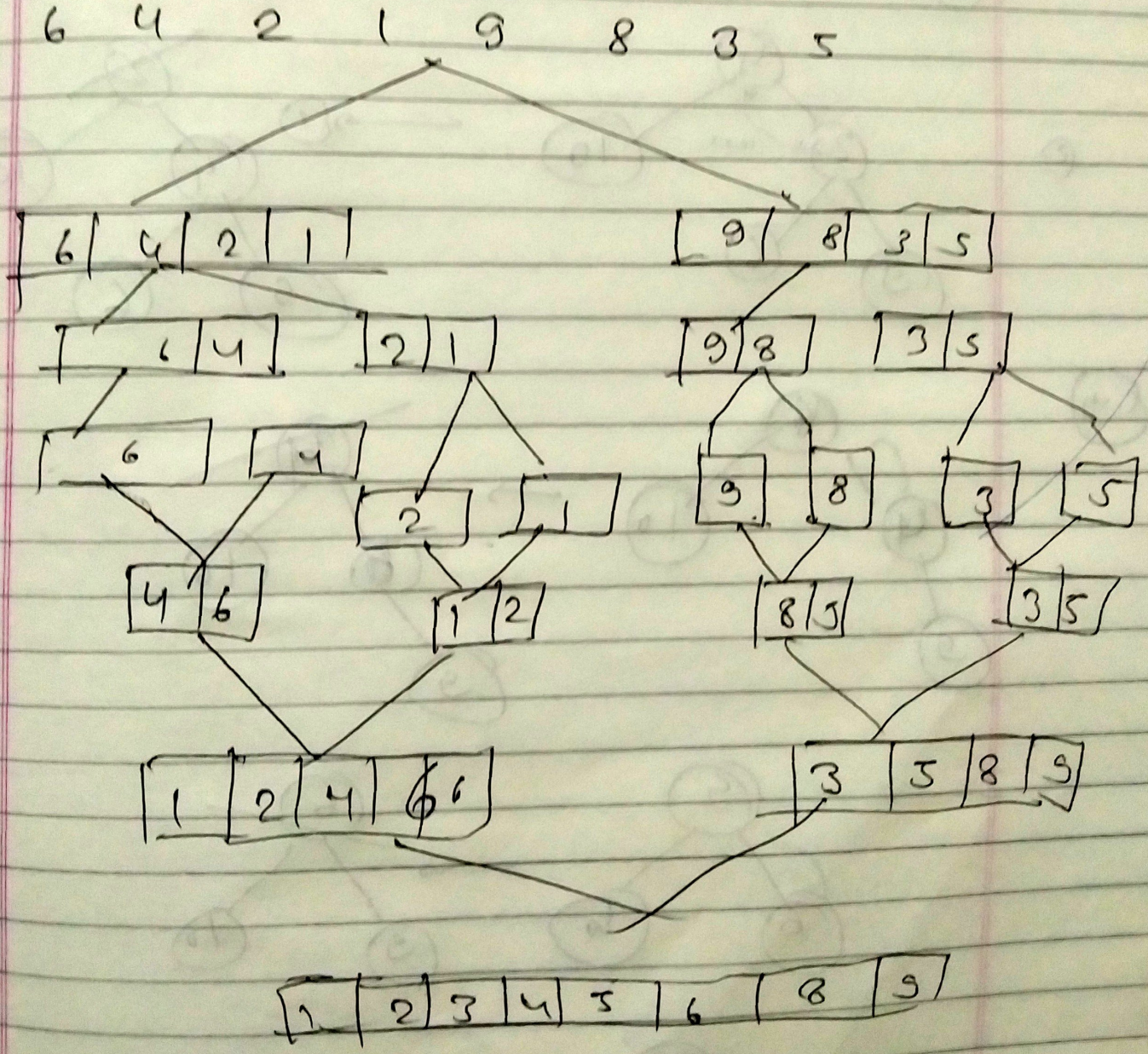
15 20 25 40 50 80 90

Best case $n \log n$
Worst case $O(n^2)$

(Time complexity $n \log(n)$)

Merge sort :-

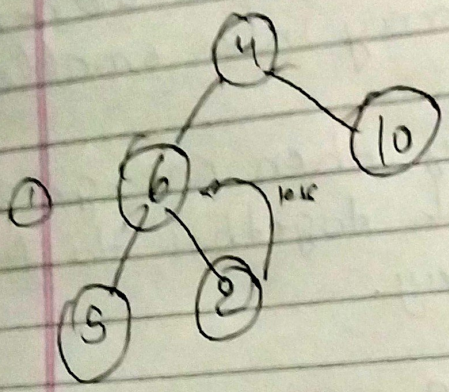
It is a sorting algorithm that works by dividing any array into smaller subarrays.
Sorting each subarray then merge the sorted sub arrays back together to form the final sorted array.

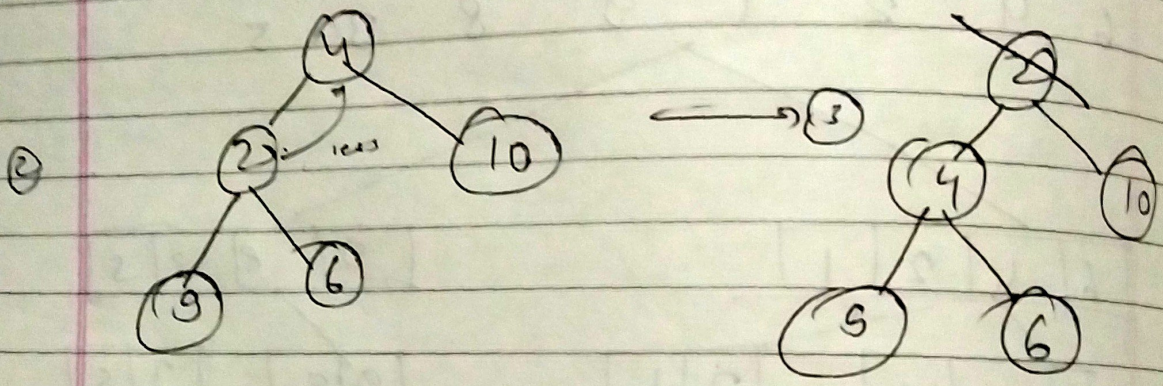


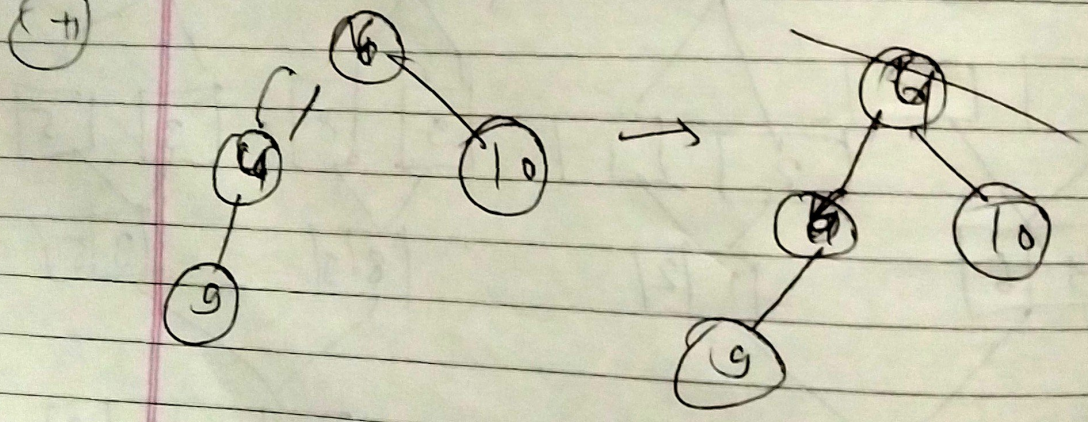
Ex ① 42, 84, 75, 20, 60, 10, 5, 30

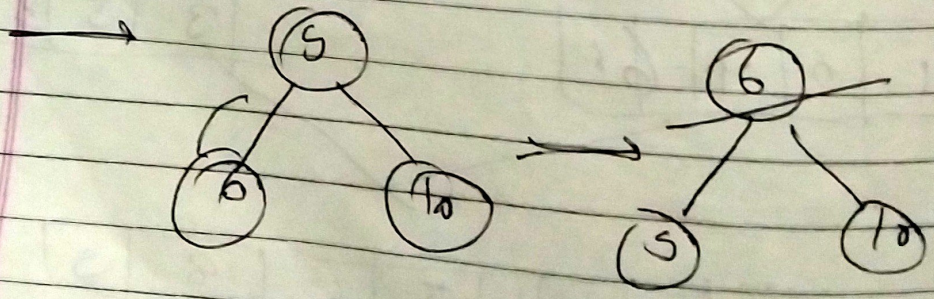
Heap sort ⇒ [Complexity n*log(n)]

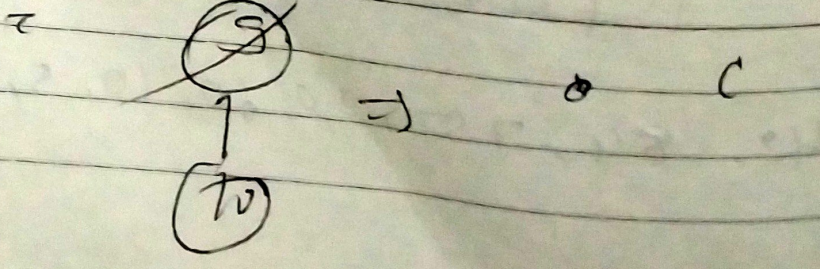
4 6 10, 9, 2











DATE: / /

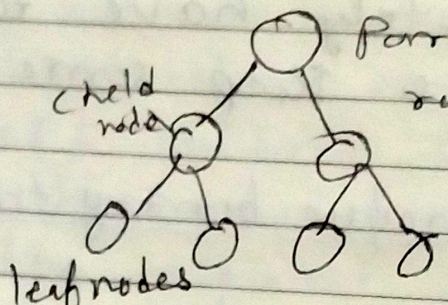
non

Tree → Tree is a linear data structure use to store data element.

Types → ① Binary tree
② An Array tree

① Binary tree -

A tree in which every node can contain almost two child node is called a Binary tree.

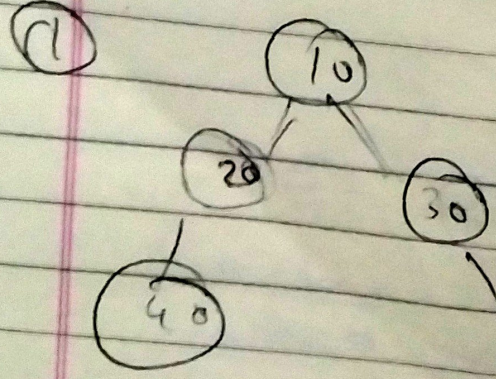


level	height
1	0
2	1
3	2

height - The longest path from the root node to leaf node is called height of tree.

height = # level - 1

Tree Traversal ⇒ ① Preorder
② Inorder
③ postorder



Inorder - 40, 20, 10, 50, 30

Preorder = 10, 20, 40, 30, 50

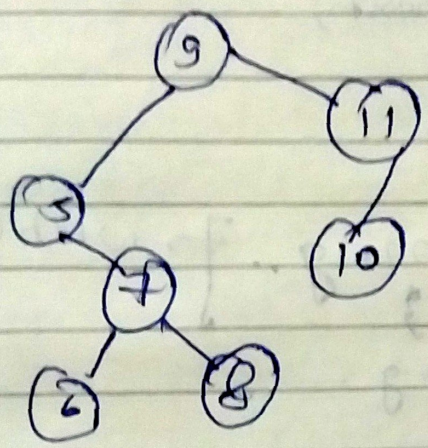
Post = 40, 20, 50, 30, 10

(sorted element)

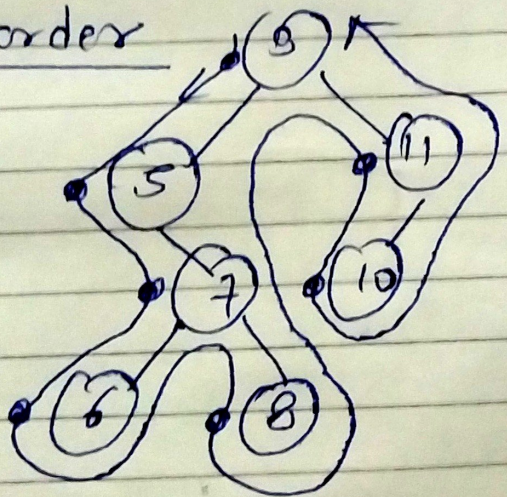
Binary search tree :

(1) Preorder = Root, left, right
(2) Inorder = left, Root, right
(3) Postorder = left, Right, Root

9, 5, 7, 8, 6, 11, 10

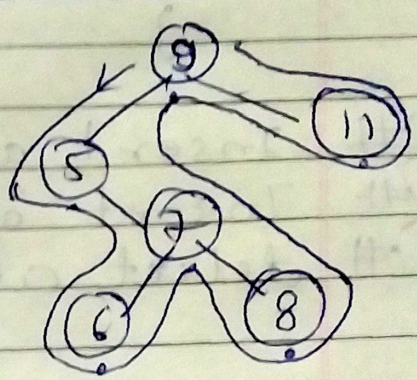


(II) Preorder



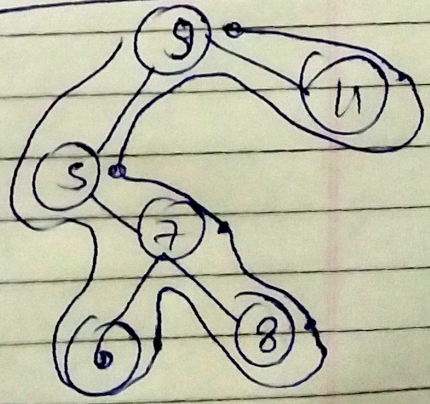
9, 5, 7, 6, 8, 11, 10

(III) Inorder

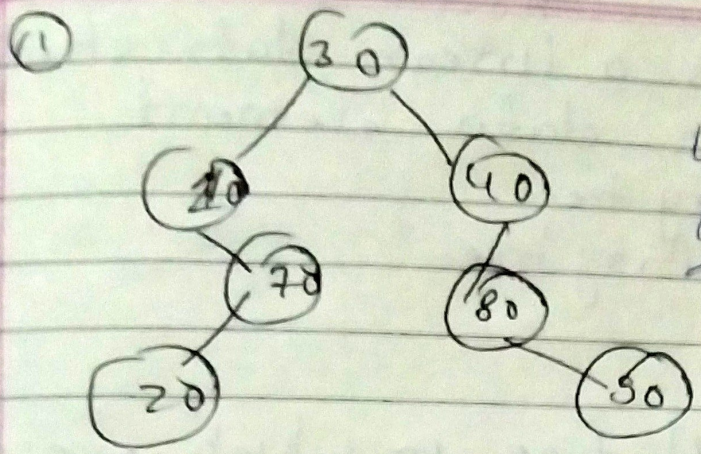


5, 6, 7, 8, 9, 11

Postorder



6, 8, 7, 5, 11, 9



(1st visit)
Preorder = 30, 10, 70, 20, 40, 80, 50
(2nd visit) Inorder = 10, 20, 70, 30, 80, 50, 40
(3rd visit) Postorder = 20, 70, 10, 90, 80, 40, 30

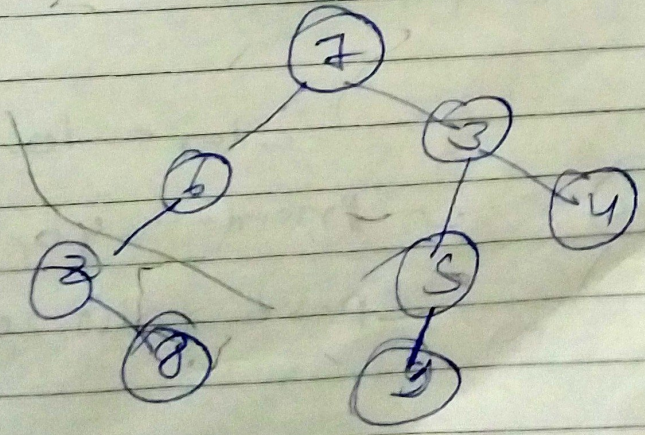
To apply the about technics every node should definitely have two child node otherwise keep some dumy node

Constructing unique binary tree :

A unique binary tree can be constructed if any two traversal are given and one of them should in order.

Ex If any two traversal are given

1) postorder : 8, 2, 6, 9, 5, 4, 3, 7
inorder : 2, 8, 6, 7, 9, 5, 3, 4



DATE: / /

Construct the unique binary with the two tree

Post: 10, 8, 23, 22, 27, 25, 15, 50, 95, 60, 40, 29

In: 8, 10, 15, 22, 23, 25, 27, 29, 40, 50, 60, 95

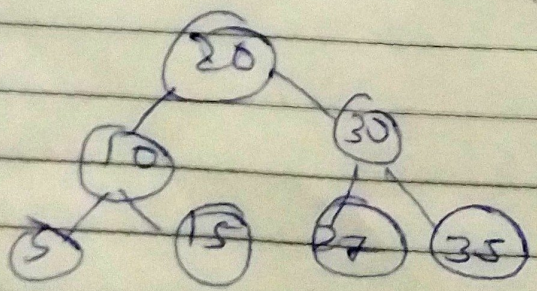
Binary search tree (BST):-

BST is also a binary tree but with the restriction of all the element of left subtree should be smaller than the root node.
All the element of right subtree should greater than root node.

All the element in BST should be unique

BST is used for searching an element.

(a) Construct the BST with the given element
→ 20, 10, 15, 30, 35, 27, 5



Preorder: 20, 10, 5, 15, 30, 27, 35
Inorder ⇒ 5, 10, 15, 20, 27, 30, 35
Postorder ⇒ 5, 15, 10, 27, 35, 30, 20

PAGE NO.:
DATE: / /

NOTE :-

Inorder Traversal of a BST is always sorted order

last element in the preorder and last element in the inorder are same. (This is true for only complete binary tree)

Time complexity for a BST ⇒

Best case $O(\log n)$

worst ⇒ $O(h)$
height

Stack data structure.

It is linear data structure

It work on LIFO

insertion & deletion operations from one end of stack data structure.

① register stack
② memory stack

Conversion of Infix into postfix, prefix :-

Q.1 # Infix ⇒ a + b * c
op1 op2

sol^n ⇒ a + (* b c)

✓ prefix ⇒ + a * bc

✓ postfix ⇒ a + b * c
⇒ a b * c * +
⇒ a + [b c *]
⇒ a [b c *] +
⇒ a b c * +

Q.2 # (A + B) * c / D convert into prefix.

sol^n Prefix -

(A + B) * / c D
(+ A B) * c / D
* (+ A B) c D
/ * + A B c D

[+ A B) * c / D
left to right
[* + A B c] / D
/ * + A B c D)

postfix ⇒

(A B +) * c / D
op1 op2
[(A B +) c *] / D
op1 op2
(A B +) c * D /

Ans:-

(ab+)

Postfix ⇒ <operand 1><operand 2><operator>

a + b * c / D ↑ e ↑ f * a - b / c

⇒ a + b * c / D ↑ (e f ↑) * a - b / c

⇒ a + b * c / (D e f ↑ ↑) * a - b / c

⇒ a + (b c *) / (D e f ↑ ↑) * a - b / c

⇒ a + (b c * D e f ↑ ↑ /) * a - b / c

⇒ a + (b c * D e f ↑ ↑ / a *) - b / c

⇒ a + (b c * D e f ↑ ↑ / a *) - (b c /)

⇒ [(a b c * D e f ↑ ↑ / a *)] - (b c /)

⇒ (a b c * D e f ↑ ↑ / a * (b c /)

⇒ a b c * D e f ↑ ↑ / a * + b c / - [Answer]

Homeworks ⇒

Q.1 (A + B) * (C + D)

Q.2 A * B + C * D

Q.3 A + B + C + D

Q.4 K + L - M * N + (O ↑ P) * W / U / V * T + Q

(Postfix)

① $(A+B) * (C+D)$

Prefix:

$* (A+B)(C+D)$

$* (+AB)(C+D)$

$* (+AB)(+CD)$

$\Rightarrow * + AB + CD$

$(AB+) * (CD+)$

$(AB+)(CD+) *$

$AB+CD+*$

② $A * B + C * D$

Prefix

$(*AB) + C * D$

$(*AB) + (*CD)$

$+ (*AB)(*CD)$

$+ * AB * CD$

Postfix

$(AB*) + C * D$

$(AB*) + (CD*)$

$(AB*)(CD*) +$

$AB*CD*+$

③ $A + B + C + D$

Prefix

$(+AB) + C + D$

$[+(+AB)(C)] + D$

$[+(+(+AB)(C)D]$

$+ + + ABCD$

Postfix

$= (AB+) + C + D$

$\Rightarrow (AB+C+) + D$

$\Rightarrow (AB+C+D)+$

$AB+C+D+$

(4) Pond

K + L − M * N + (O ↑ P) * W / U / V * T + Q

K + L − M * N + (O P ↑) * W / U / V * T + Q

K + L − (M N *) + (O P ↑) * W / U / V * T + Q

K + L − (M N *) + (O P ↑ W *) / U / V * T + Q

K + L − (M N *) + (O P ↑ W * U /) / V * T + Q

K + L − (M N *) + (O P ↑ W * U / V /) * T + Q

K + L − (M N *) + (O P ↑ W * U / V / T *) + Q

(K L +) − (M N *) + (O P ↑ W * U / V / T *) + Q

(K L + M N * −) + (O P ↑ W * U / V / T *) + Q

(K L + M N * − O P ↑ W * U / V / T * +) + Q

K L + M N * − O P ↑ W * U / V / T * + Q + Ans

Q.
(5) $a * (b + c)$

(a) Prefix:

$a * (+bc)$

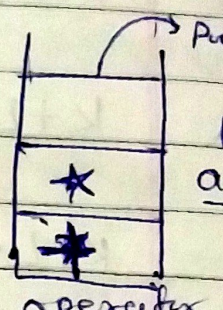
$(*a+bc)$

(1) Postfix

$a * (bc+)$

$(abc+) *$

$abc+*$



Algorithm in form of table

I/P	TOS	
H	L	PUSH
L	H	POP
L	H	PUSH(R-L)
E	E	POP(L-R)
E	E	PUSH
C	op+	PUSH
op+	C	PUSH
C	C	

operand A, B, C, X, Y, P, Q ...

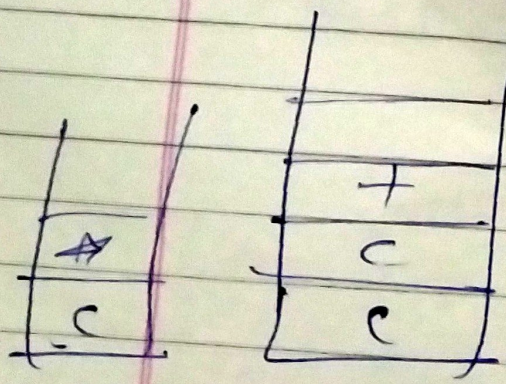
operator +, -, *, /, ^

Q.1 $((a+b) * c)$

Sol: Postfix

$((ab+) * c)$

$= (ab+c*)$



$ab+c*$

Q.2 # $a+b*(c/d1ef1)*(a-b)/c$

Post-

$\Rightarrow a+b*(c/D1ef1)*(a-b)/c$

$\Rightarrow a+b*(c/(Def11)]*(a-b)/c$

$\Rightarrow a+b*(c/(Def11)]*(a-b)/c$

$\Rightarrow a+b*(cDef11/)*(a-b)/c$

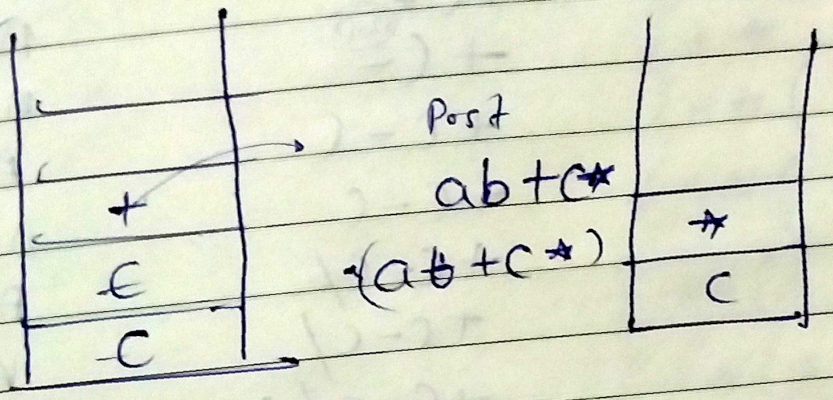
(6) (1) a + b - c

(2) a * b + c

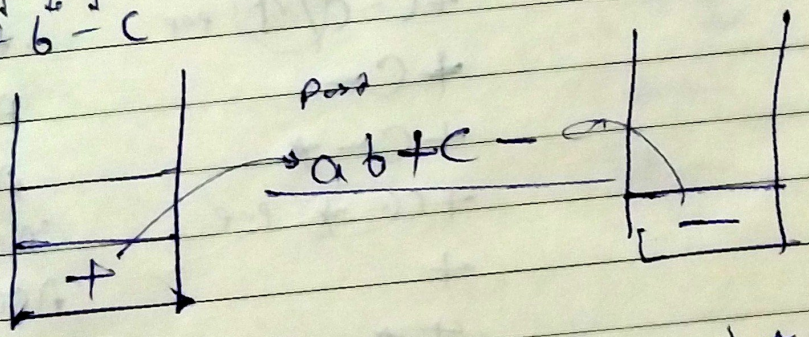
(3) a ↑ b ↑ c

(4) a * (b + c

(5) ((a + b) * c)



(1) a + b - c



Q.1 A + (B * C - (D / E ↑ F) * G) * H

A + B * C − (D / E ↑ F) * G) * H

PAGE NO.:
DATE: / /

Q.1

SYMBOL	STACK	POSTFIX
A		A
+	+	A
C	+ C	A
B	+ C	AB
*	+ C *	AB
C	+ C * pop	ABC
−	+ C −	ABC *
C	+ C − (	ABC *
D	+ C − C	ABC * D
/	+ C − C /	ABC * D
E	+ C − C /	ABC * DE
↑	+ C − C / ↑	ABC * DE
F	+ C − C / ↑ pop	ABC * DEF
)	+ C −	ABC * DEF ↑ /
*	+ C − *	ABC * DEF ↑ /
G	+ C − * pop	ABC * DEF ↑ / G
)	+	ABC * DEF ↑ / G * −
*	+ *	ABC * DEF ↑ / G * −
H	+ *	ABC * DEF ↑ / G * − H
		ABC * DEF ↑ / G * − H *

2 imp

(A + B / C * (D + E) - F)

Q.2

SYMBOL	STACK	POSTFIX
(	(	
A	(	A
+	(+	A
B	(+	AB
/	(+ /	AB
C	(+ /	ABC
*	(+ *	ABC/
(	(+ * (	ABC/
D	(+ * (	ABC/D
+	(+ * (+	ABC/D
E	(+ * (+	ABC/DE
)	(+ *	ABC/DE+
-	(-	ABC/DE+*
F	(-	ABC/DE+*F
)		ABC/DE+*F-

A + (/ B

① (A + B / C * (D + E) - F)

Prefix ⇒

⇒ (A + B / C * (+ DE) - F)

⇒ (A + (/ BC) * (+ DE) - F)

⇒ (A + [* (/ BC + DE)] - F)

⇒ (+ A * / BC + DE) - F

⇒ - + A * / BC + DEF

(A + B / C * (+ DE) - F)

(A + B / * (C + DE) - F)

(A + (/ B * (C + DE) - F)

A + (- / B * C + DEF)

+ A - / B * C + DEF

1) (A+B/C*(D+E)-F)

Refix ⇒ (F-(E+D)*C/B+A) ⇒

SYMBOL	STACK	POSTFIX
C	C	
F	C	F
−	C−	F
(	(−(	F
E	(−(	FE
+	(−(+	FE
D	(−(+	FED
)	(−	FED+
*	(−*	FED+
C	(−*	FED+C
/	(−/	FED+C*
B	(−/	FED+C*B
+	(+	FED+C*B/−
A	(+	FED+C*B/−A
)	pop	FED+C*B/−A+

⇒ FED+C*B/−A+

Reverse:

+A−/B*C+DEF

⇒ A+(B*C−(D/E↑F)*G)*H

A+(B*C−(D/[↑Ef])*G)*H

⇒ A+(*BC)−(D/[↑Ef]*G)*H

2, A+[(*BC)−(/D↑Ef)*G]*H

⇒ A+*[−*BC*/D↑EfG]*H]

5) A+(A−*BC*/D↑EfGH)

⇒ A*−*BC*/D↑EfGH Proved that

H*(G*(F↑E/D)-C*B)+A

H	*	H
*	*(	H
C	*(	H
G	*(*	HG
*	*(*(	HG
(	*(*(	HG
F	*(*(	HGF
↑	*(*(↑	HGF
E	*(*(↑	HGFE
/	*(*(/	HGFE↑
D	*(*(/	HGFE↑D
)	*(*	HGFE↑D/
-	*(-	HGFE↑D/*
C	*(-	HGFE↑D/*C
*	*(-*	HGFE↑D/*CB
B	*	HGFE↑D/*CB**
)	+	HGFE↑D/*CB**+
+	+	HGFE↑D/*CB**+A
A		
.		

~~+A*-BC*/~~

+A*-*BC*/D↑EFGH

PAGE NO. :
DATE : / /

a+b*c/D↑e↑f*a-b/c

c/b-a*f↑e↑D/C*b+a

c	-	c
/	/	c
b	/	eb
-	+	cb/
a	-	cb/a
*	-*	eb/a
f	-*	cb/af
↑	-*↑	cb/af
e	-*↑	cb/afe
↑	-*↑↑	cb/afe↑
D	-*↑↑	cb/afe↑D
/	-/	cb/afe↑D↑*
C	-/	cb/afe↑D↑*C
*	-*	cb/afe↑D↑*C/
b	-*	cb/afe↑D↑*C/b
+	-+	cb/afe↑D↑*C/b*-
a	+	cb/afe↑D↑*C/b*-a
-	o	cb/afe↑D↑*C/b*-a+

⇒ +a-*b/C*↑↑Defa/bC

⇒ +a-*b/C*↑↑Defa/bC

79

Key = 67

0	1	2	3	4	5	6
4	9	3	6	11	15	19

Linear search ⇒

linear search is a very simple search algorithm in this type of search a sequential search is made over all item one by one.

every item is checked & if a match is found then that particular item is returned otherwise search continue till the end of data collection.

① best case O(1)
② worst case O(n)

Binary search ⇒

0	1	2	3	4	5	6	7	8	9
10	15	18	25	29	32	37	40	54	60

l r target 37

(sorted assending order)

element are arranged in sorted form.

Time complexity O(log₂n)

$$mid = \frac{l+r}{2}$$

$$mid = \frac{0+9}{2} = 4.5 = 4$$

condetions occurs

① A[mid] == Target
return;

② A(mid] < Target
l = mid + 1;

③ A[mid] > Target
r = mid - 1;

⇒

	l	r	mid
①	0	9	4
②	5	9	7
③	5	6	5
④	6	6	6

⇒

0	1	2	3	4	5	6	7	8	9
10	15	18	25	29	32	37	40	54	60

l mid r

0	1	2	3	4	5	6	7	8	9
10	15	18	25	29	32	37	40	54	60

l mid r

stack :-

A stack is a container of a object that are inserted & removed according to last in first out (LIFO). Principle.

Object can be inserted at any time. Inserting an item known as "pushing".

It allows insertion & deletion operation from one end of the stack data structure, that is top.

A stack is a logical concept that consist of a set of similar element.

It is an abstract data type that hold an ordered linear sequence of items.

It is used for evaluting expression with operands & operation (Infix to postfix conversion)

Queues :-

A queue is defter from a stack that its insertion & removal operation according to first in first out (FIFO) principle.
Elements may be inserted at any time. but only element which has been in the queue the longest may be removed.

Element are inserted at the rear (enqueued) & removed from the from (dequeued)

It is a linear data structure that is open at both end
It is used to manage threads in multithreading & implementing priority queue