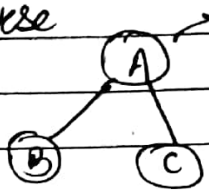
Date : Page :

UNIT - 5
TREE & GRAPH

Root
left
Right

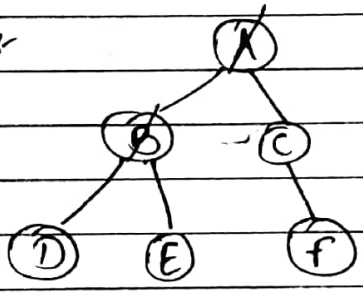
(1) Traverse → node



A	B	C

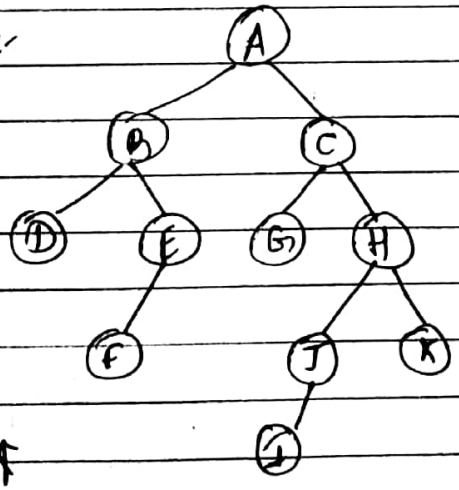
(a) Pre → Root, left, right
(b) Post → left, right, root
(c) In → left, root, right

(a) Ex:-



A, B, D, E, C, f (Pre)

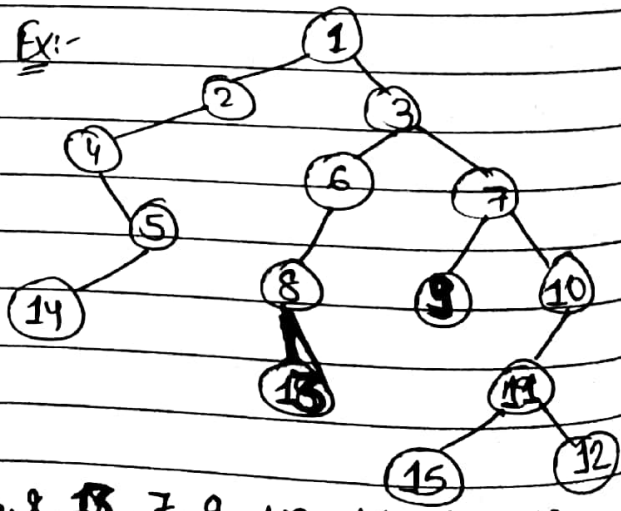
(b) Ex:-



In, → D

Pre, A, B, D, E, f, C, G, H, J, I, K

(c) Ex:-



Pre, 1, 2, 4, 5, 14, 3, 6, 8, 13, 7, 9, 10, 11, 15, 12

(d) Ex:-

e) Ex:-

Ex:-

Ex:-

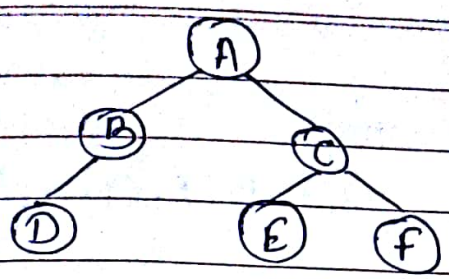
Ex:-(f

Ex:-f

Ex(

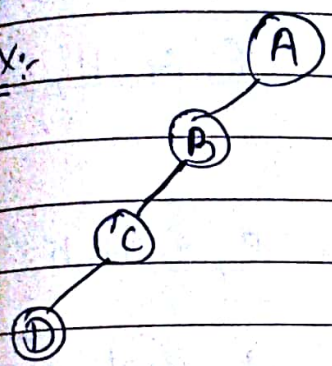
Date : Page :

Ex:-



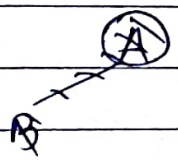
In,
D, B, A, E, C, F

Ex:-



Pre → A, B, C, D
In → D, C, B, A
Post → D, C, B, A

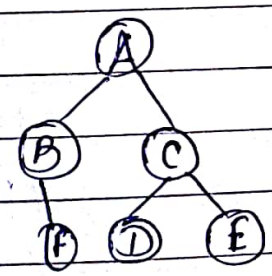
Ex:-



Ex(b) → In,
D, B, F, E, A, G, C, I, J, H, K

Ex(c) → In, ~~4, 14, 5, 2, 1, 8, 8, 6, 3, 9, 7, 15, 11~~
4, 14, 5, 2, 1, 8, 13, 6, 3, 9, 7, 15, 11, 12, 10

Ex:-(f)



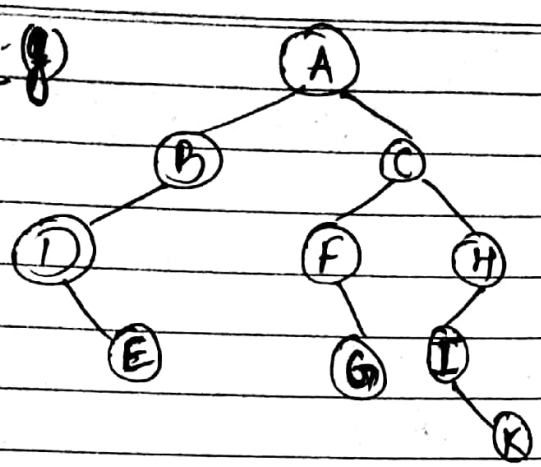
Post,
B, F, D, E, C, A

Ex:-(g) E(b) → Post,
D, F, E, B, G, I, J, K, H, C, A

Ex(c) → Post,
14, 5, 2, 8, 8, 6, 9, 15, 12, 11, 10, 7, 3, 1

Date : Page :

Ex :- (1)

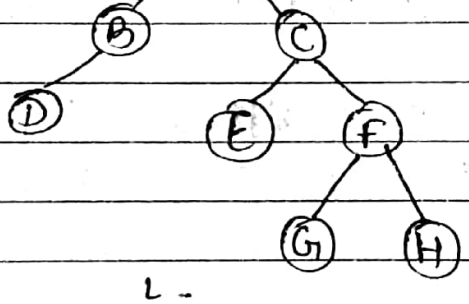


Post,
D, E, B, F, G, I, K, H, C, A

Pre,
A, D, E, B, C, F, G, H, I, K

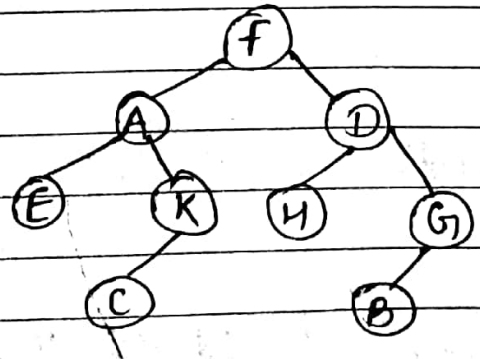
In,
D, E, B, A, F, G, C, I, H, K

Q) Inorder → D, B, A, E, C, G, F, H — L, Root, R
Ans) Post → D, B, A, E, G, H, F, C, A — L, R, Root



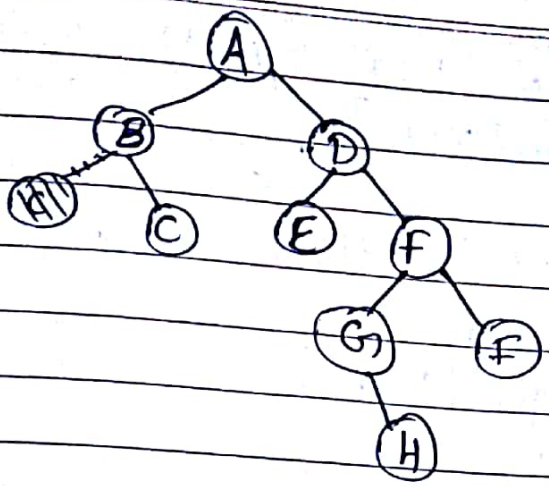
L R

Q) In → E, A, C, K, F, H, D, B, G — L, Root, R
Pre → F, A, E, K, C, D, H, G, B — Root, L, R



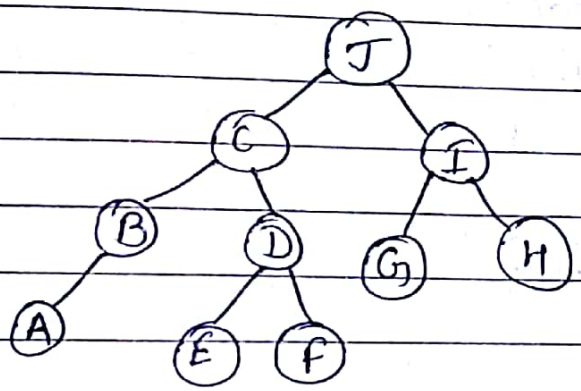
Q.) Pre → A, B, C, D, E, F, G, H, I — Root, L, R
In → B, C, A, E, D, G, H, F, I — L, Root, R

L R



Q) Post → A, B, C, E, D, F, G, H, I, J — L, R, Root

In → A, B, C, E, D, F, J, G, I, H — L, Root, R



> Algorithm :-

I Pre (node, info, left, right)

1) If node != NULL then repeat till 4).
2) Display info[node].
3) Call Pre order (left[node]).
4) Call Pre order (right[node]).
5) Exit.

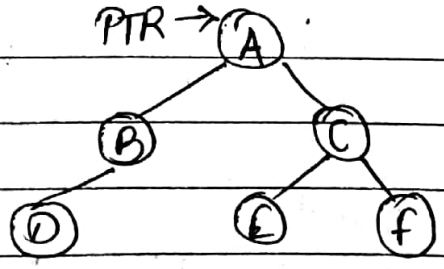
II Pre order (Info, Left, Right, Root)

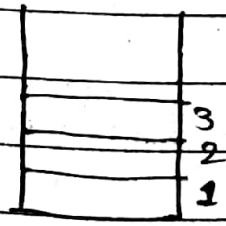
1) Initially push NULL into the stack and initialize PTR.
   Set TOP = 1 and stack[TOP] = NULL & PTR = Root

Date : Page :

2) Repeat 3) to 5) while PTR ≠ NULL.] loop
3) Apply PROCESS to Info[PTR].
4) [Right Child?]
if Right[PTR] ≠ NULL, then
Set TOP = TOP + 1
and Stack[TOP] = Right[PTR]
(end of if)
5) [Left Child?]
if Left[PTR] ≠ NULL, then
Set PTR = Left[PTR]
else (pop from Stack)
Set PTR = Stack[TOP]
and TOP = TOP - 1
(end of if)
[end of step 2 loop]
6) Exit.

Ex-



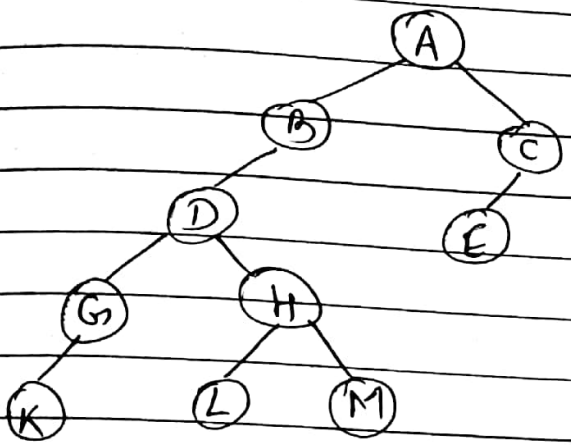


O/P → A, B, D, C, E, f

Date : Page :

UNIT - 5

Continued....



Pre → Root, L, R

A, B, D, G, K, H, L, M, C, E

Post → L, R, Root

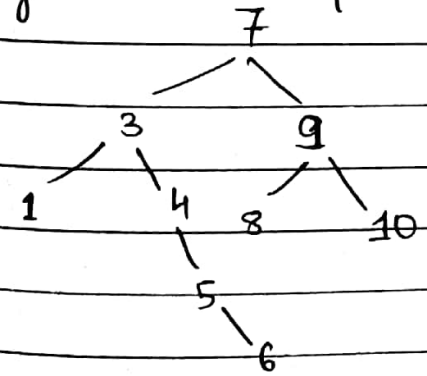
K, G

➢ Binary search Tree → A binary tree is called BST if it satisfies following rules :-

(a) The value of left child or left sub tree should be less than the value of root.

(b) The value of right child or right subtree should be greater or equal to the value of root.

Ex)

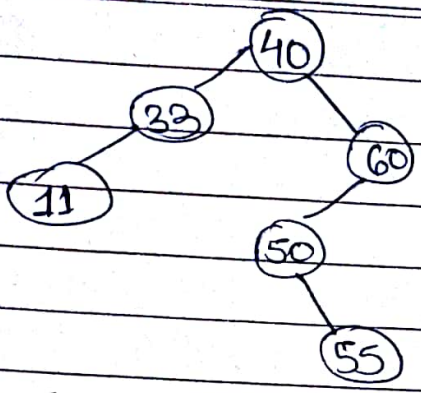


Q) Create binary search tree with following values :-

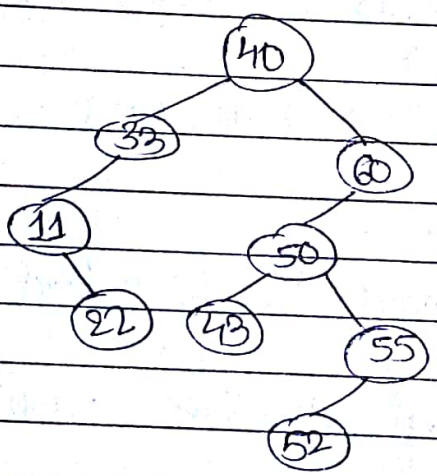
40, 60, 50, 33, 55, 11

Date : Page :

Ans.)



▷ Insert 22, 43 and 52 in existing BST.

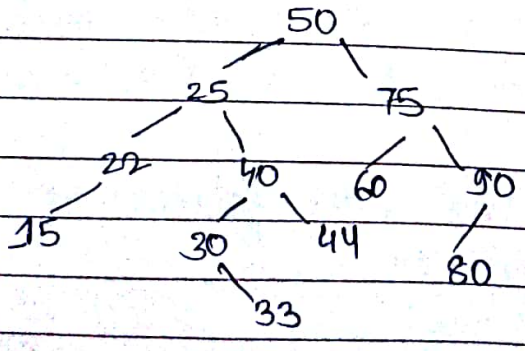


I) Q) Create BST → 50, 75, 25, 40, 22, 30, 33, 15, 44, 60, 90, 80

(a) Add 20 and 88.
(b) Delete 22. (c) Delete 25. (d) Add 77
(e) Delete 50 and 75 and give the output.

Ans.) BST →

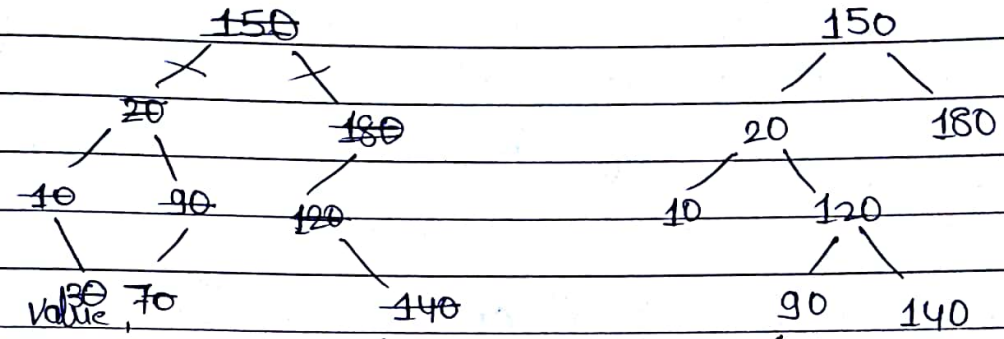
(a)



Date : Page :

Q) Create BST → 150, 180, 20, 120, 90, 140, 10, 70, 30, 50, 60, 40

Ans)



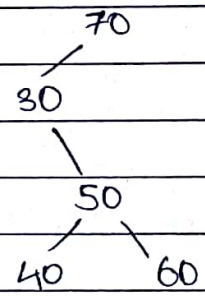
To delete a value, 70

(a) find In order traversal.

In → L, Root, R

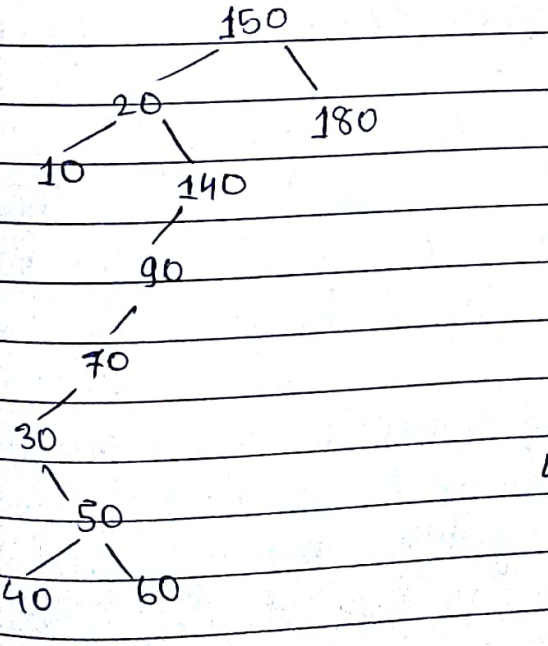
10, 20, 30, 40, 50, 60, 70, 90, 120, 140, 150, 180

*(all the values are sorted) PC

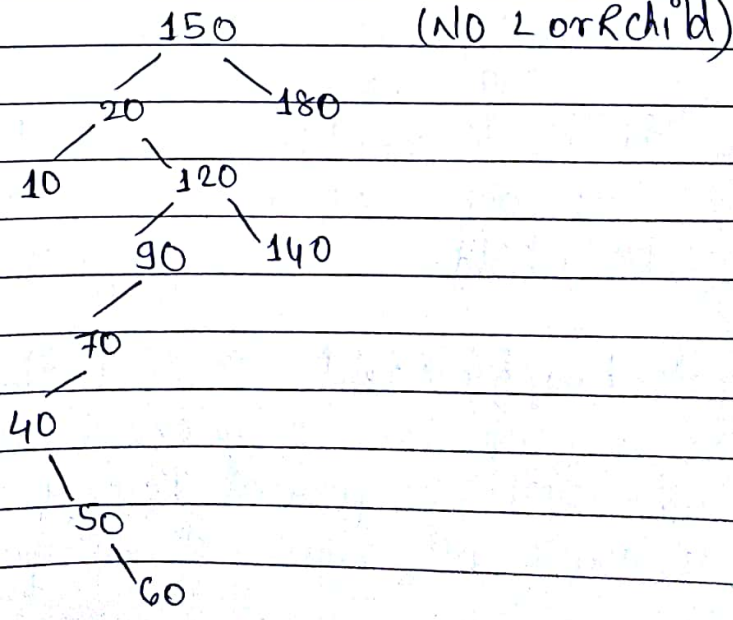


(b) Delete the element and place the In-order successor of the deleted element at its place.

Delete 120, In-order successor = 140 (NO L or R child)



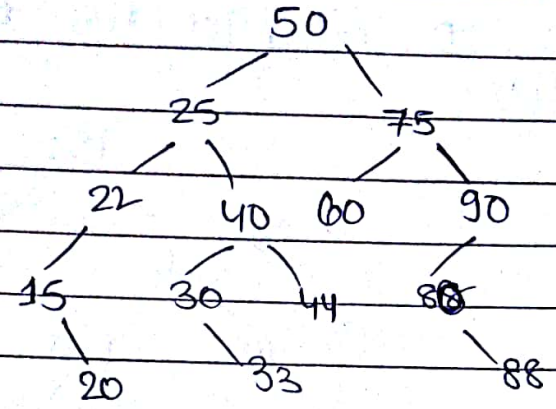
Delete 30, IOS = 40 (NO L or R child)



Date : Page :

I

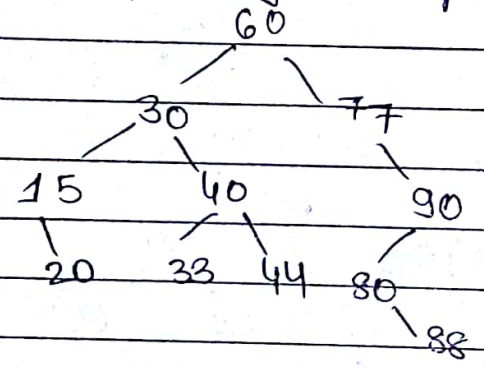
Ans. (a)



Ans. (b) In → L, Root, R

~~15~~, ~~22~~, ~~30~~, ~~40~~

After performing all operations, output =>



A, M, P, B, S, D, Z, C, T, N, G, Q, W, E, X

Heap sort – Heap is the DS in which values are arranged either in increasing or decreasing order. There are two types :-

(i) Max heap (ii) Min heap

They are the It is a complete binary tree with the prop. that value at each node is either large or small to the value of its children. This prop. is called heap prop.

➢ Heapify / delete → It is the processor for manipulating heap DS. It picks the largest child and compare it with the parent key, if parent key is larger than it quits otherwise it swap the parent key with

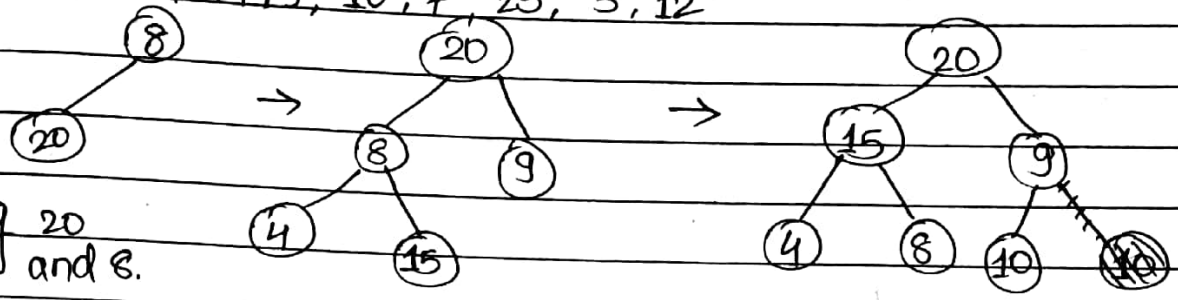
Date : Page :

largest child key.

Q) Create max heap and apply heapify.

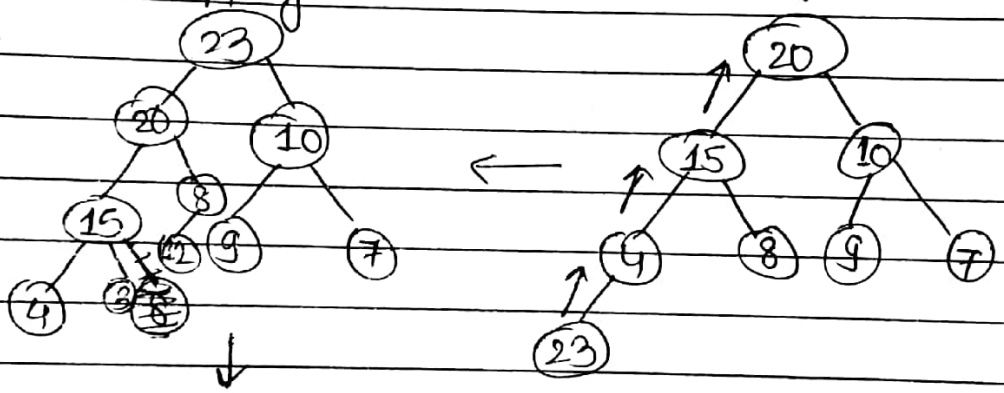
8, 20, 9, 4, 15, 10, 7, 23, 3, 12

Ans)



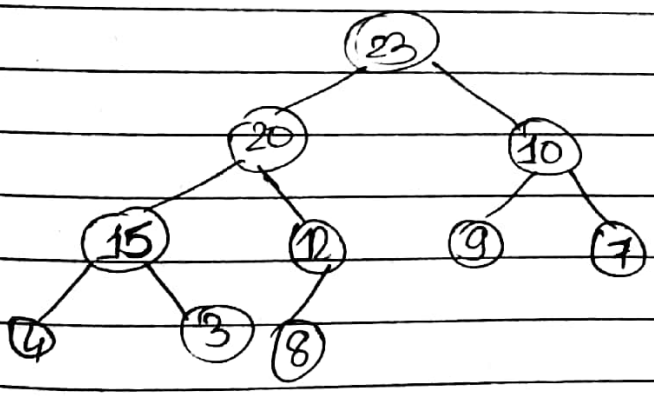
swapping 20 and 8.

swapping 15 and 8

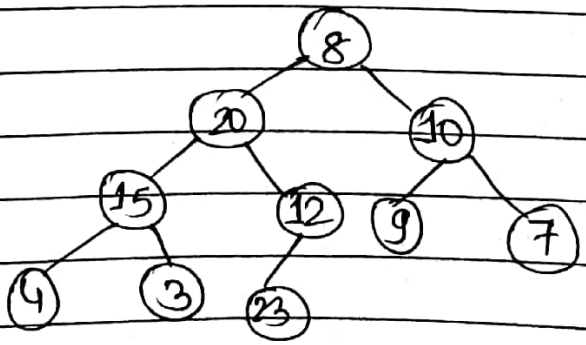


swapping 8 and 12.

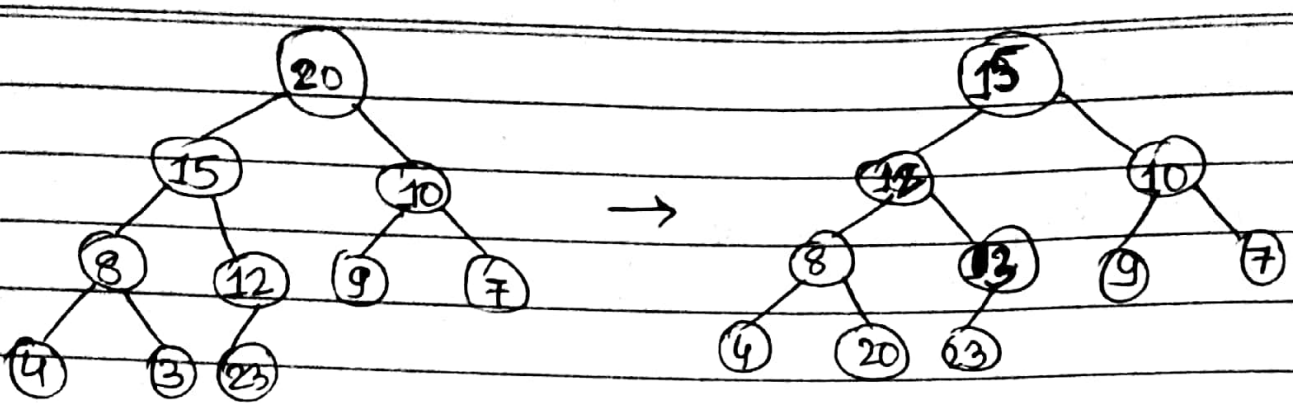
swapping 23 and 20



- Create minheap.

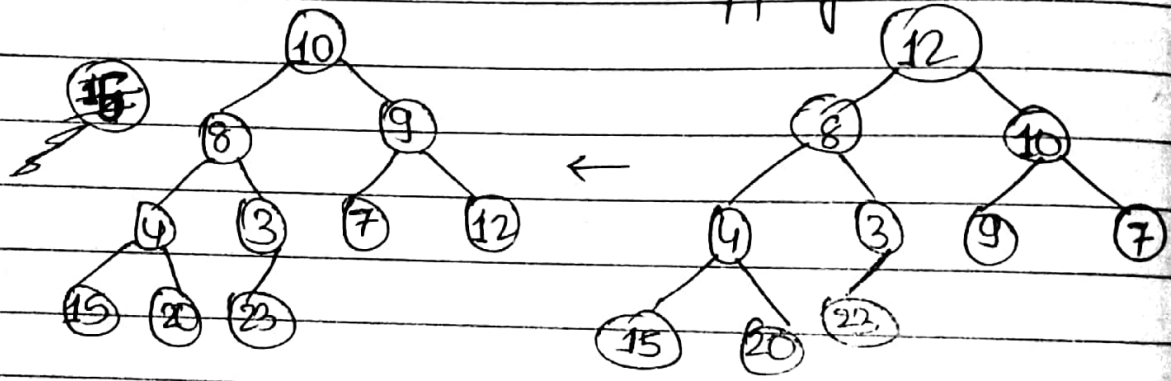


Date : Page :



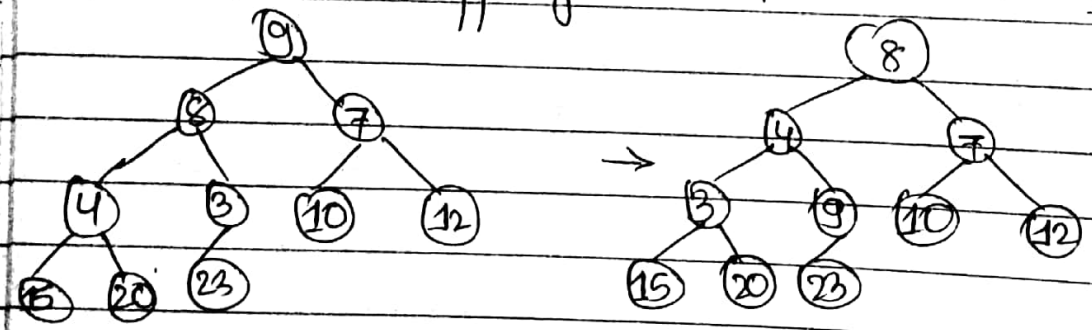
Swapping 3 and 20

Swapping 4 and 15



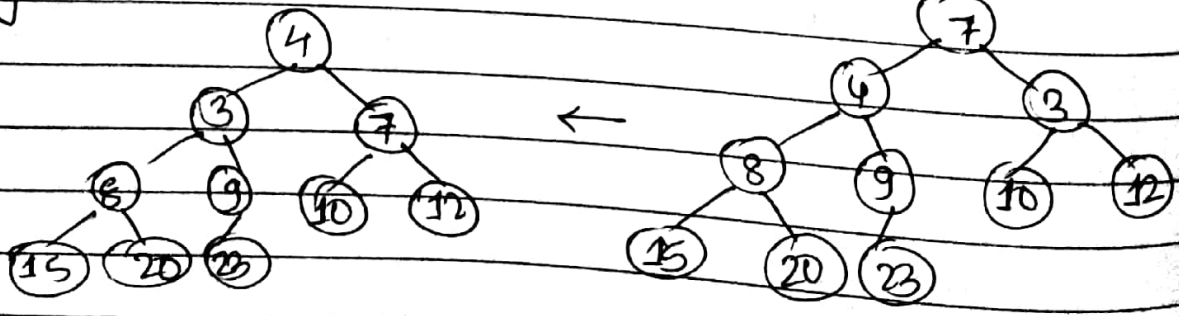
Swapping 10 and 7

Swapping 7 and 12



swapping 3 and 9

Swapping 8 and 3



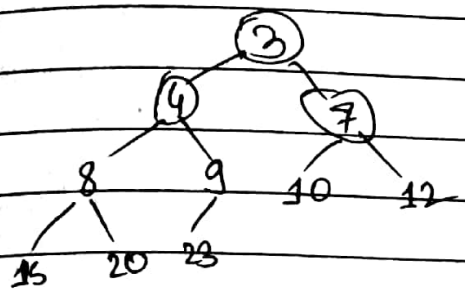
swapping 4 and 3

Swapping 7 and 3

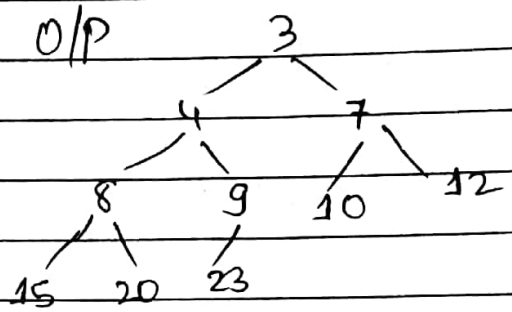
Date : Page :

25, 6, 87, 22, 47, 10, 35, 7, 63, 52, 41, 2

Q) Create min heap and apply heapify.



O/P



3	4	7	8	9	10	12	15	20	23