Assignment-4:

1 ans:

(a) In-order Traversal:

$a \to c \to e \to d \to q \to n \to r \to w \to s$

i.e, a c e d q n r w s

(b) Pre order traversal:

q e c a d r n s w

(c) Post order traversal:

a c d e h w s r q

2 ans:

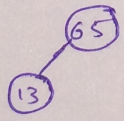
Nodes to be Inserted:

~~83, 12, 68, 55, 32, 6, 46, 57, 62~~

65, 13, 16, 52, 28, 11, 20, 14, 87, 50, 28

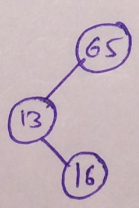
Step 1:

Step 2:



All nodes are balanced

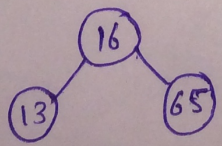
Step 3:



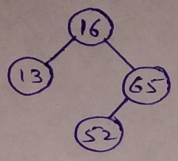
Node 65 is unbalanced

⇒ Since, 16 newly inserted node is inserted "inside" we use the double rotation.

after rotation ⇓

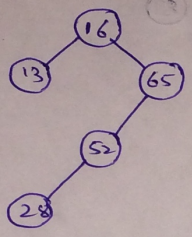


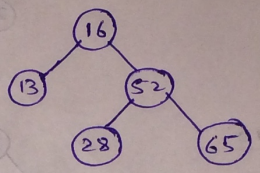
step 4:



All nodes are balanced

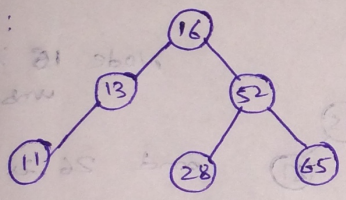
step 5:



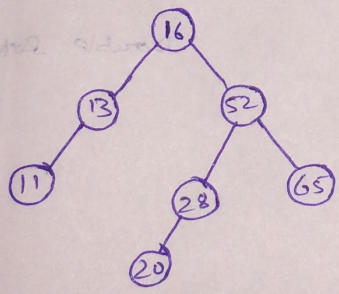


Since node 65 is unbalanced, and 28 is inserted outside we perform Single Rotation.

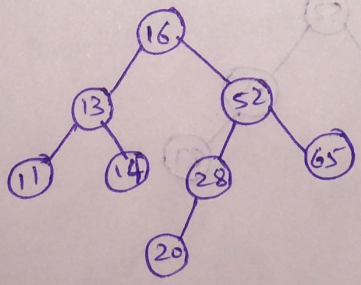
step 6:



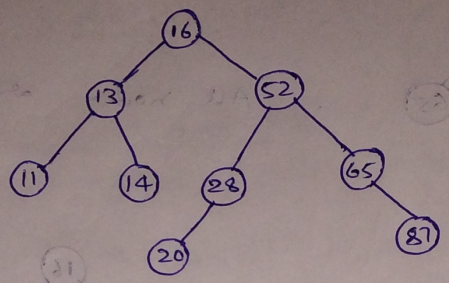
step 7:



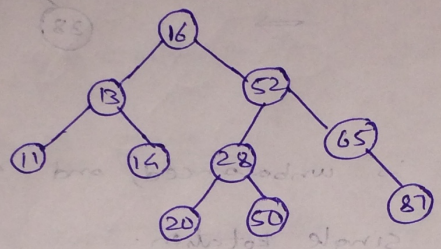
step 8:



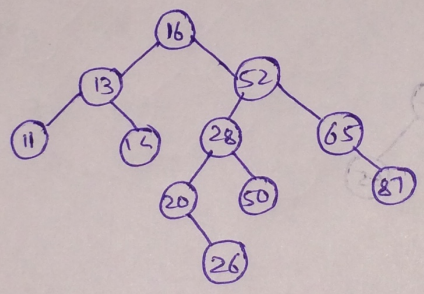
Step 9:



Step 10:

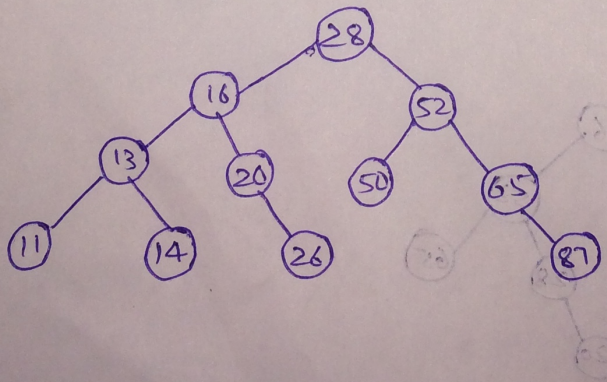


Step 11:



Node 15 is unbalanced and 26 is inserted in Inside, so perform Double Rotation.

After Double rotation.

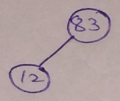


Sons: Nodes to be Inserted:

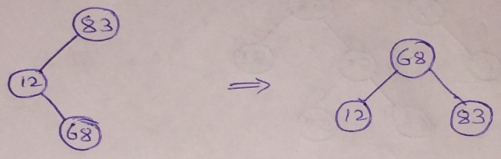
83, 12, 68, 55, 32, 6, 46, 57, 62

step 1:

step 2:

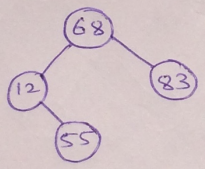


step 3:

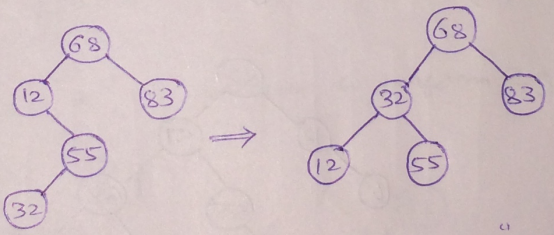


Node 83 is unbalanced after inserting 68 "inside" so perform Double rotation.

step 4:

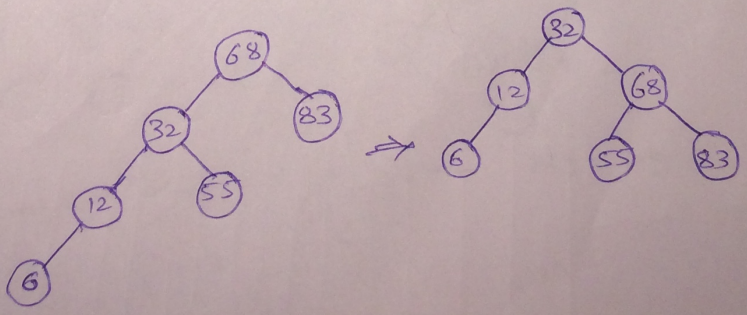


step 5:

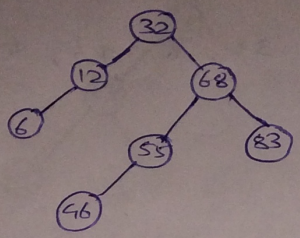


Node 12 is unbalanced after inserting 32 "inside" so perform Double rotation.

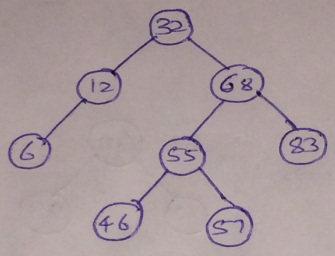
Step 6:



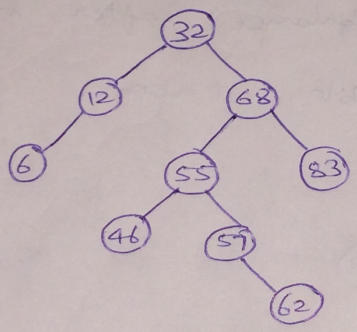
Step 7:

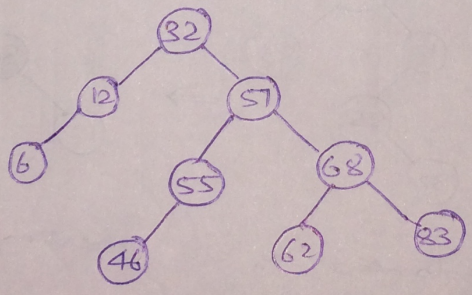


Step 8:

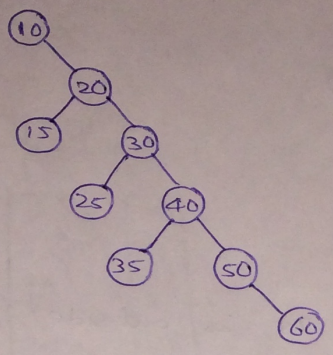


Step 9:



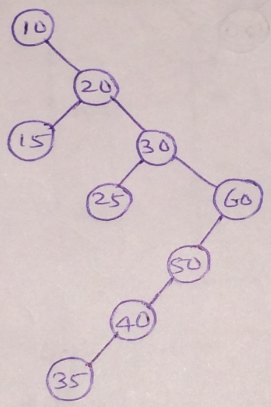


4ans: Given, Tree:

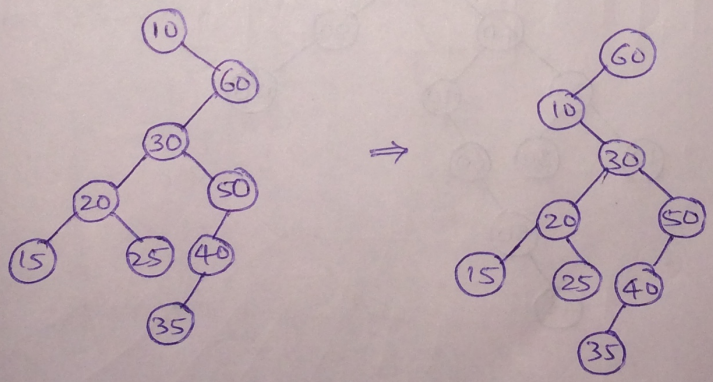


The node to be accessed is 60.

In above tree it is outside, so perform Zig-Zig.

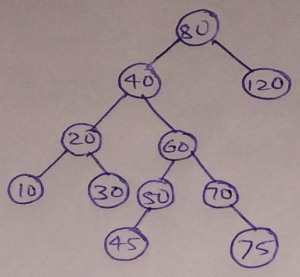


In above tree also "60" is outside, so perform Zig-Zig

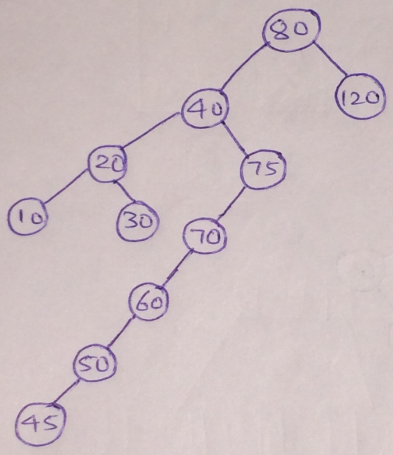


as "60" is again outside perform Zig-Zig.

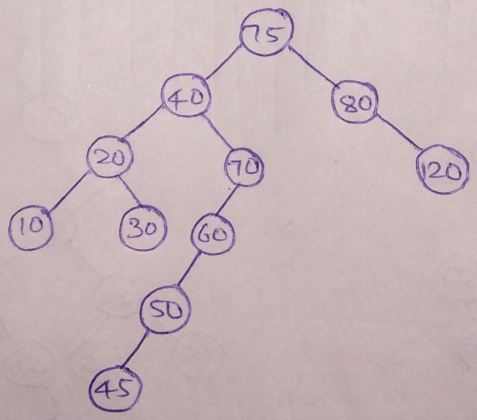
Sol^n: Given, Tree is



Node to be accessed 75 is "outside", so we perfo
the Zig-Zig operation.

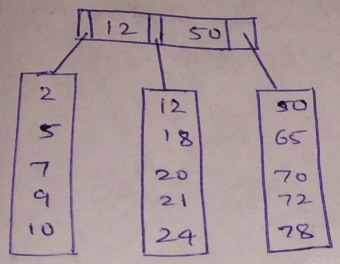


Now, since 75 is inside, we perform Zig-Zag.

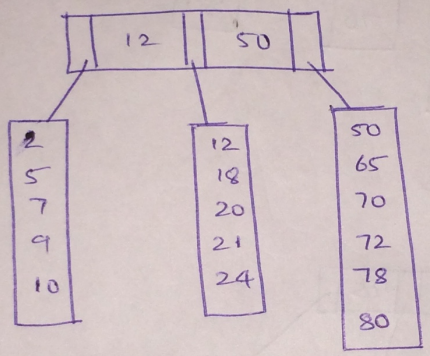


Ans:

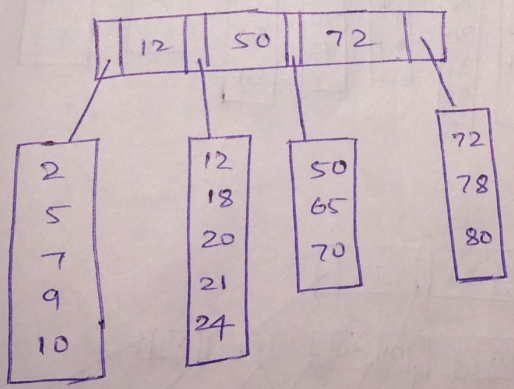
Given B+ Tree:



step 1: Insert 80:

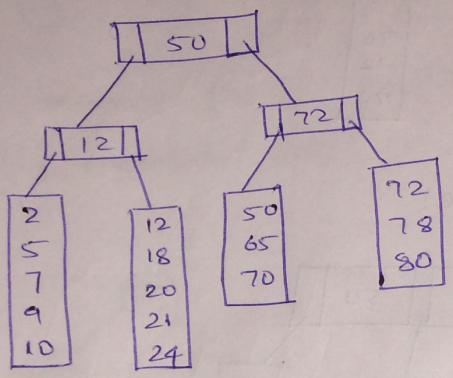


step 2: Since, L=5 one leaf can have a maximum of 5 only 5..

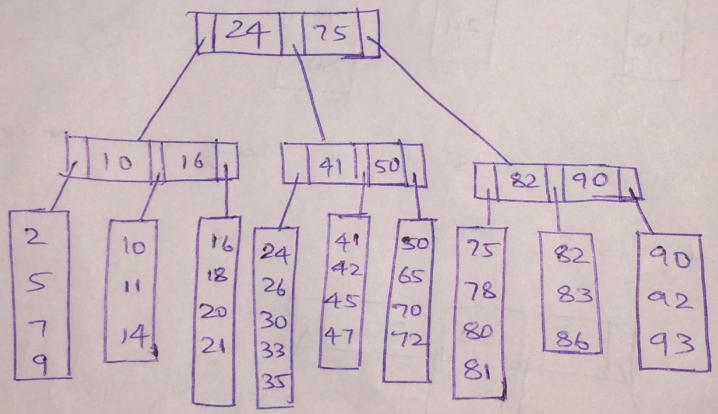


Step 3:

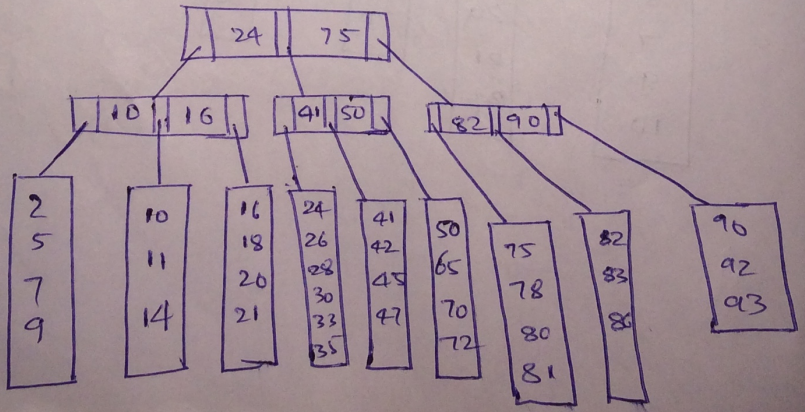
Since M=3, every non-leaf node can have only M-1 i.e 2 keys. but here are shown 3 keys so we split them into nodes as below,



Sol: Given, B+ Tree:

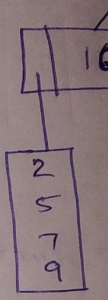


step 1: Insert 28:



step 2: Sin

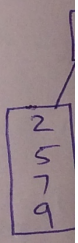
maxium of



step 3: since

have only

below:

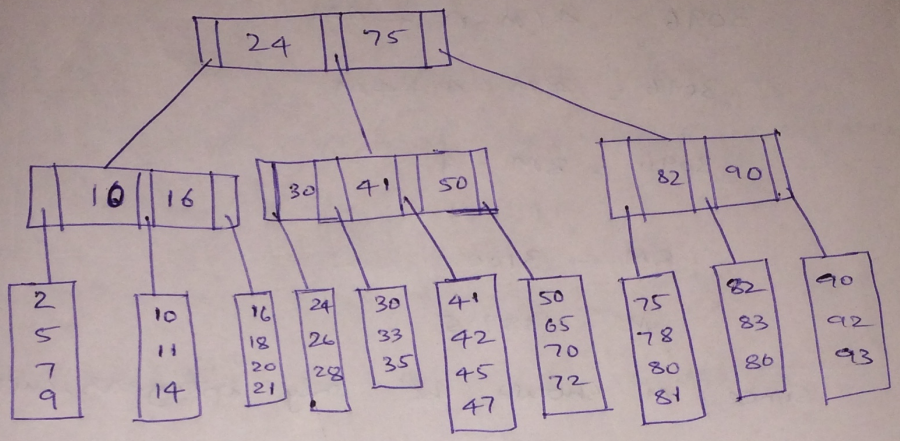


step 4:

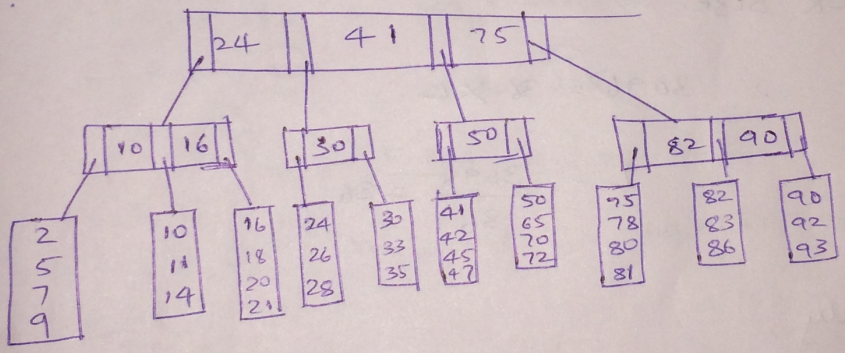
con

are
shown
below,

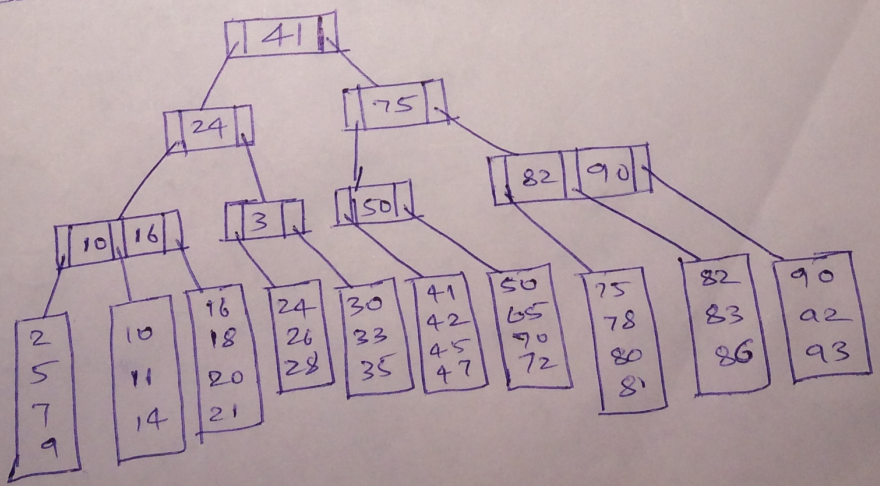
step 2: since, L=5 a leaf can have only a maximum of 5 values



step 3: since m=3, each non-leaf node can have only m-1 i.e 3-1=2 keys. so do as shown below:



step 4: Same as reason,



8ans:

each non-leaf node can have maximum of M-1 keys and M pointers

$$3096 = 4(M-1) + 4M$$

$$3096 = 4M - 4 + 4M$$

$$3096 = 8M - 4$$

$$\therefore 8M = 3100$$

$$M = 387.5$$

since, M should be only integers, we truncate

$$\therefore M = 387$$

As the leaf node is stored only in a single block size.

$$\therefore 3096 = 36 \times L$$

$$L = \frac{3096}{36} = 86$$

Finally,

$M = 387$ and $L = 86$

9ans:

Let

(i)

(ii)

(i) work

and a

for work

No. of

so,

so,

(ii)

Ans:

Let us consider 2 cases:

(i) worst - case

(ii) Best - case

(i) worst - case:

The B+ tree can have maximum of n children and a minimum of $\left\lceil \frac{n}{2} \right\rceil$ children for worst case,

No. of records in a single block is $\frac{L}{2} = \frac{86}{2} = 43$

So, $43 \times \left(\frac{n}{2}\right)^N = 86\,00\,000$ (N is depth)

$$\left(\frac{387}{2}\right)^N = 2,00,000$$

$$N \log\left(\frac{387}{2}\right) = \log 200000$$

$$N \times 2.286 = \log_{10} 2 + \log 100000$$

$$N \times 2.286 = 0.301 + 5$$

$$\therefore N = \frac{5.301}{2.286} = 2.31 \approx 3$$

So, the No. of level required is (depth+1) i.e 4.

(ii) Best - case:

$$86 \times (387)^N = 8,600,000$$

$$(387)^N = 1,00,000$$

$$N \log 387 = \log 100000$$

$$\therefore N = \frac{\log 100000}{\log 387} = \frac{5}{2.587} = 1.932 \approx 2$$

No of levels required is (depth+1) i.e, 3.

10 ans:

In a Binary tree, each node can have only upto 2 children as maximum.

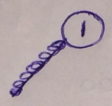
so, in other words, a node can

(i) 0 children (or)

(ii) 1 child (or)

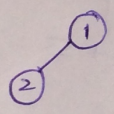
(iii) 2 children.

Case 1: 1 node

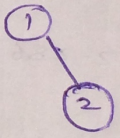


It has 0 children i.e, 2 null child pointers

Case 2: 2 nodes



(or)

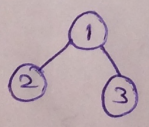


It has 3 null child pointers

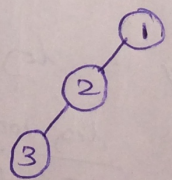
(By replacing one of the above null child pointer with 2 new null child pointers)

i.e, $2 - 1 + 2 = 3$

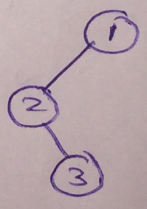
Case 3:



(or)



(or)



All above trees have exactly 4 null child pointers.

Some above explanation,

$3 - 1 + 2 = 4$

have

So, by seeing above observation we can say that as no. of nodes are increasing by 1 the no. of null child pointers are increasing by 1.

also, from case 1: no. of nodes (N) = 1
no. of null child pointer = 2 (i.e, N+1)

Similarly, case 2: N = 2
no. of null child pointer = 3 (i.e, N+1)

Thus we can say that if there 'N' nodes in a tree, we will have N+1 null child pointer

Child
ointers

11 ans:

As shown in the previous question,

⇒ N nodes will have N+1 null child pointers

Adding an another level in a perfect binary tree means adding nodes to all null pointers.

ter

Let us assume a perfect binary tree has N nodes, so no. of null child pointers are N+1

To add new nodes and make the resulting tree to be perfect binary tree, all null pointers are added with new nodes

∴ Thus resulting number is,

Total = N + (N+1)

[N → old nodes
N+1 → new nodes]

= 2N+1

rs.

∴ Adding a new level to a perfect binary tree of N nodes will makes 2N+1 nodes in total.