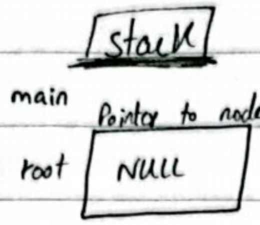
insertion Dry Run.

Day: Date:

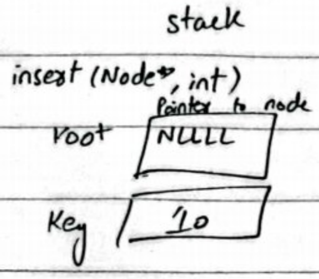
int main(){



Node* root = NULL;

root = insert (root,10)

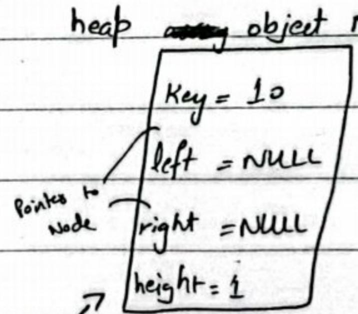
↓ function called.



if (root == NULL)   condition true

return new Node(Key);

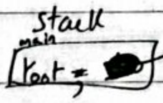
heap ~~any~~ object Node.



all condition will be skip

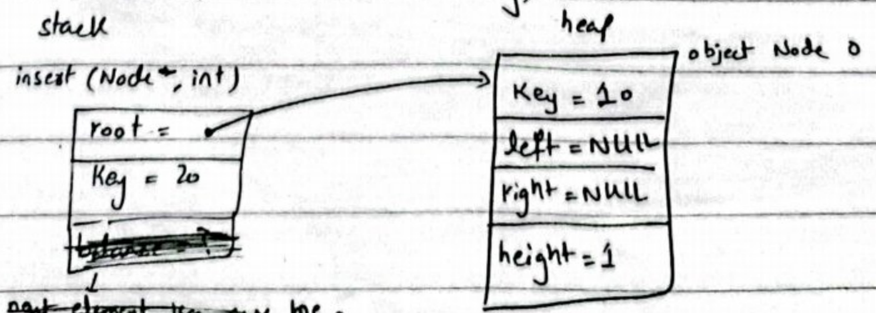
return root;

↓ of insert function.



root = insert (root , 20) function called.

Node* insert (Node* root, int Key)



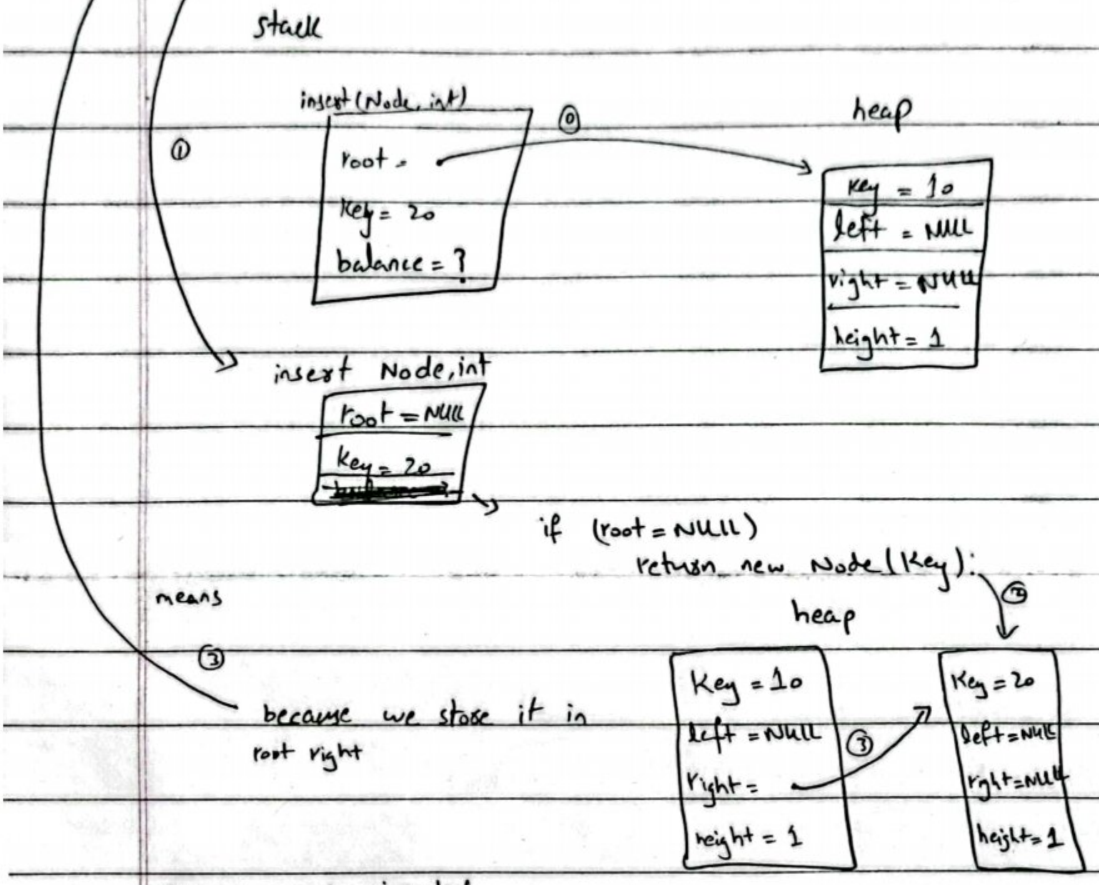
~~next element kon case~~ me figure out krogi.

balance factus = bf.

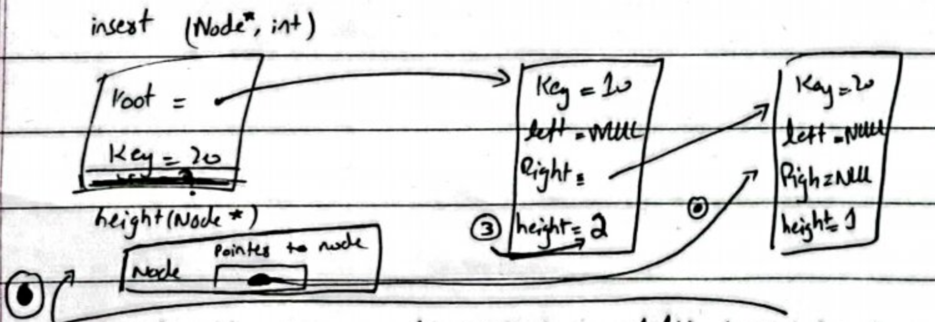
Day: condition skip   Date:

else if (Key > root->Key)   true...

root->right = insest (root -> right, Key);   recursively call itself mean.
NULL



As new node inserted
updated Stack   insert [root = NULL / Key = 20] end or remove from stack.



root->height = 1+ max (height (root->left), height (root->right));
(0, 1)
②

Node == NULL (false condition)

return ~~height~~ node->height => (1) in this case.

for height (root->left) which is NULL so return 0.

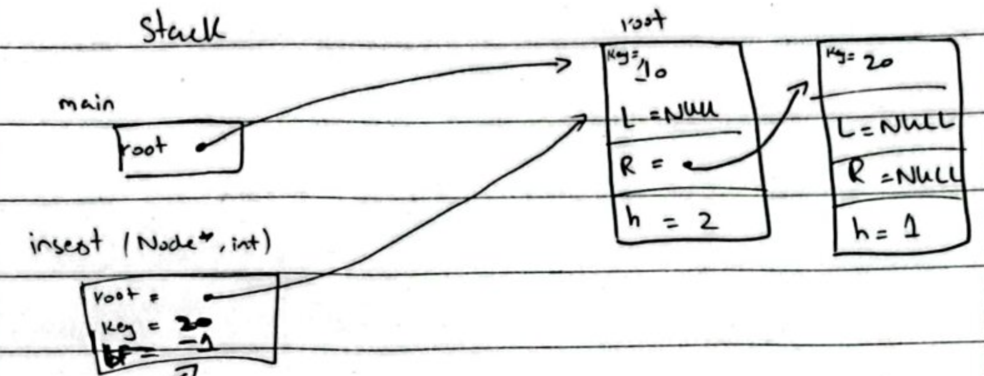
stack
max (int, int)

a = 0
b = 1

(a > b) ? a : b ;   return b = 1

root->height = 1 + 1 = 2.

get balance(root);



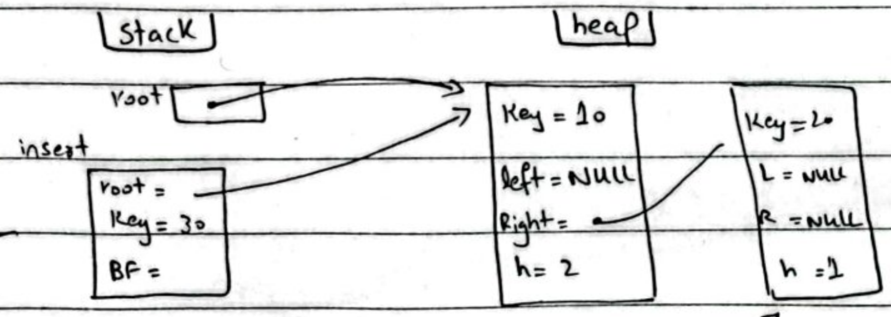
(0 - 1)
return height (node->left) - height (node->right)

same as before calculating height;

next condition will be skip.

Third insert

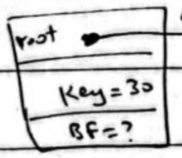
root = insert (root, 30);



else if (key > root->key)

~~return~~ root->right = insert ($\overset{20}{\text{root->right}}$, $\overset{30}{\text{key}}$)

for this recursive function

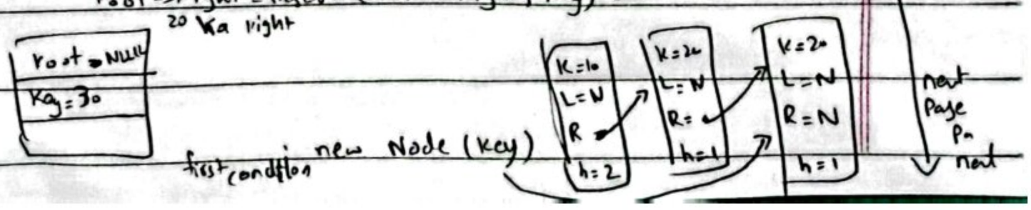


other condition skip

else if (key > root->key)

root->right = insert ($\overset{\text{NULL}}{\text{root->right}}$, $\overset{30}{\text{key}}$)

20 ka right



Day: Now for Node 20 Date:

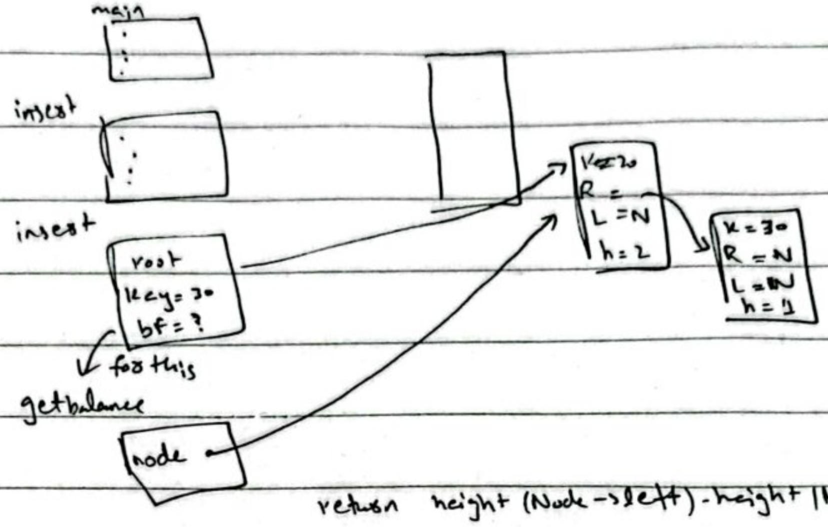
we check it height.

root->height = 1 + max(height(root ....)

↳ (20)

Same as we done with in 10 last insertion

ma....



return height (Node->left) - height (Node->Right)

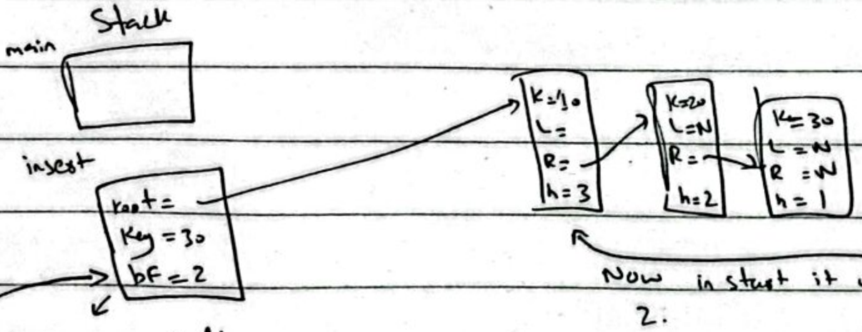
NULL - 1

0 - 1

-1

next condition for 20 will skip

return to first time we call insert.



Now in start it was 2.

where we left

root->height = 1+ max (height (root->left), height (root->right));

Same procedure

1 + max(0, 2);

= 1+2 = 3

get balance height (root left) - height (root Right)

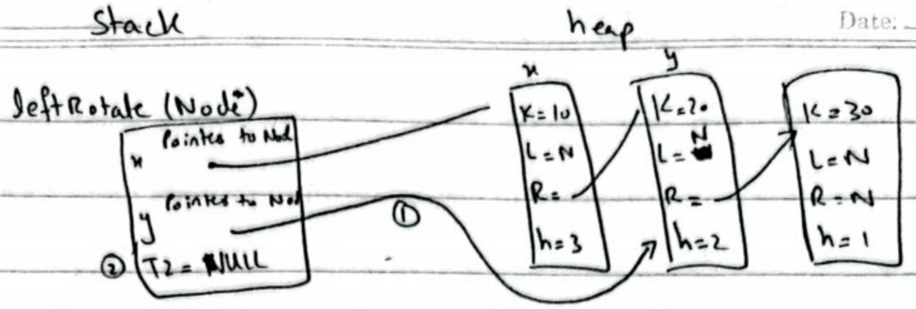
0 - 2 = 2

Now rotation condition will True.

it Right Right case.

if (balance > 1 && key > root->Right->key)

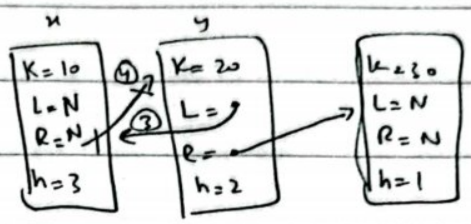
return leftRotate(root); function call.



① Node* y = x->Right;

② Node* T2 = y->left;

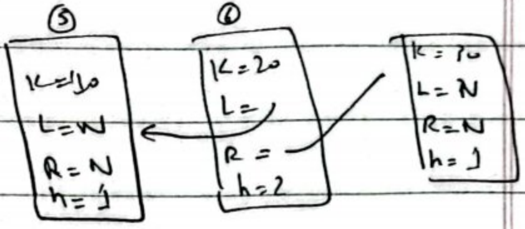
③ y->left = x



④ x->Right = T2

⑤ x->height= .... calculating height

⑥ y->height=.... calculating height

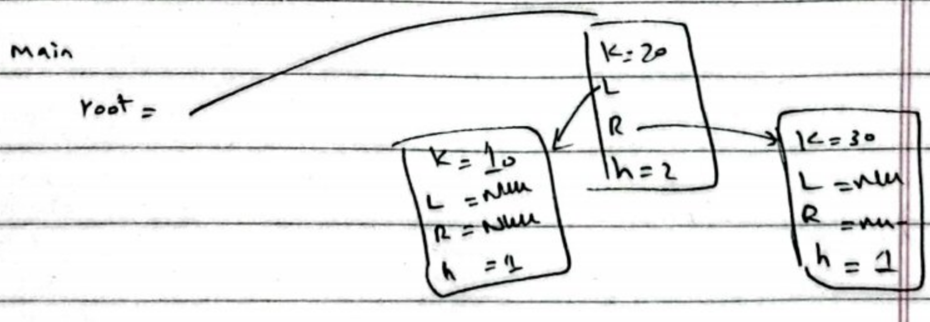


return y.

return root;

in main function

root = 20



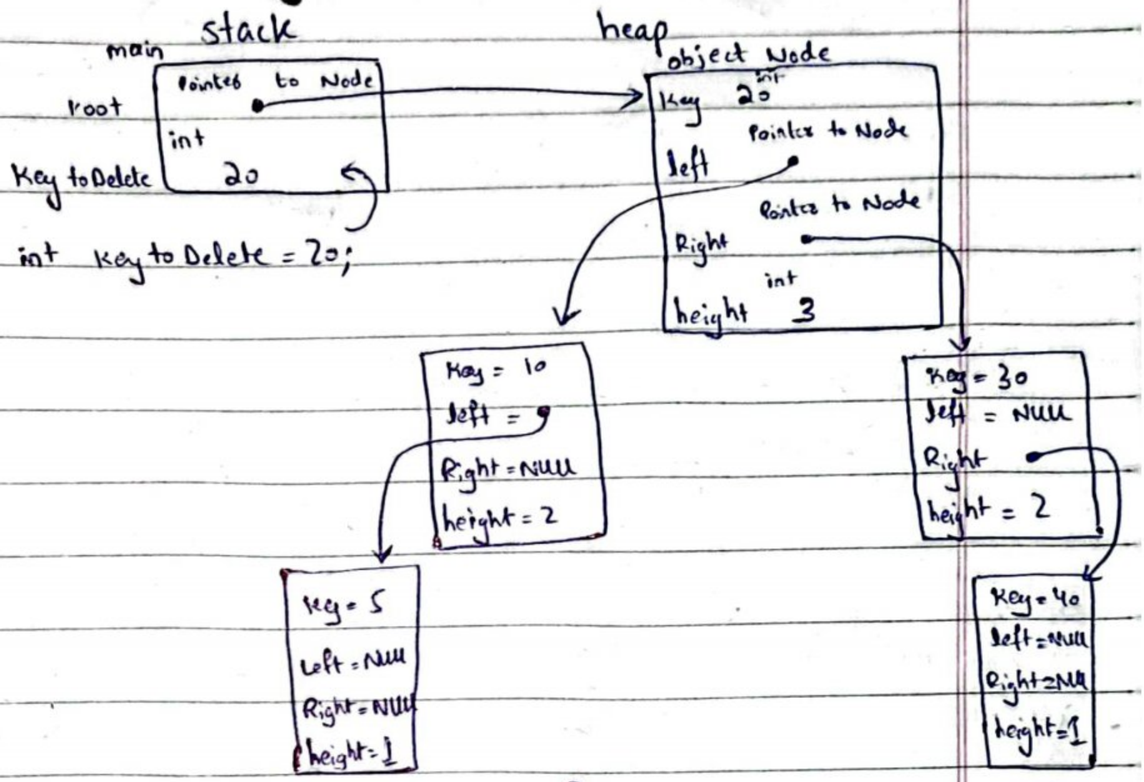
rest will be same

add 40 and 5 taka easy raha sab

ka diya. ... ☺

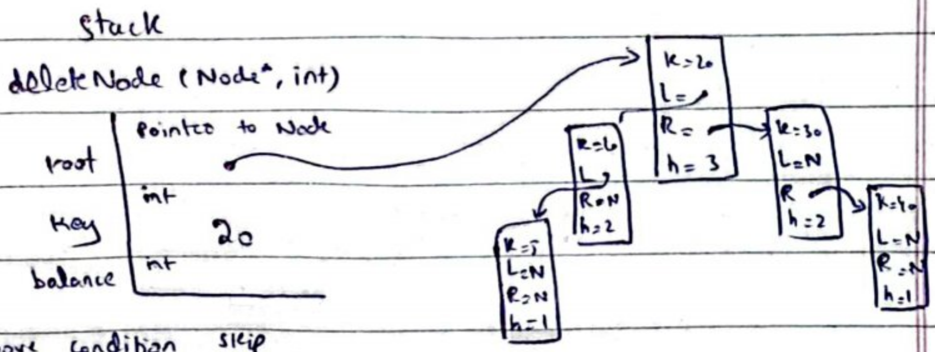
Node* return a pointer to the node

Dry Run Deletion



int key to Delete = 20;

root = deleteNode(root, keytoDelete) function call.



above condition skip

else{

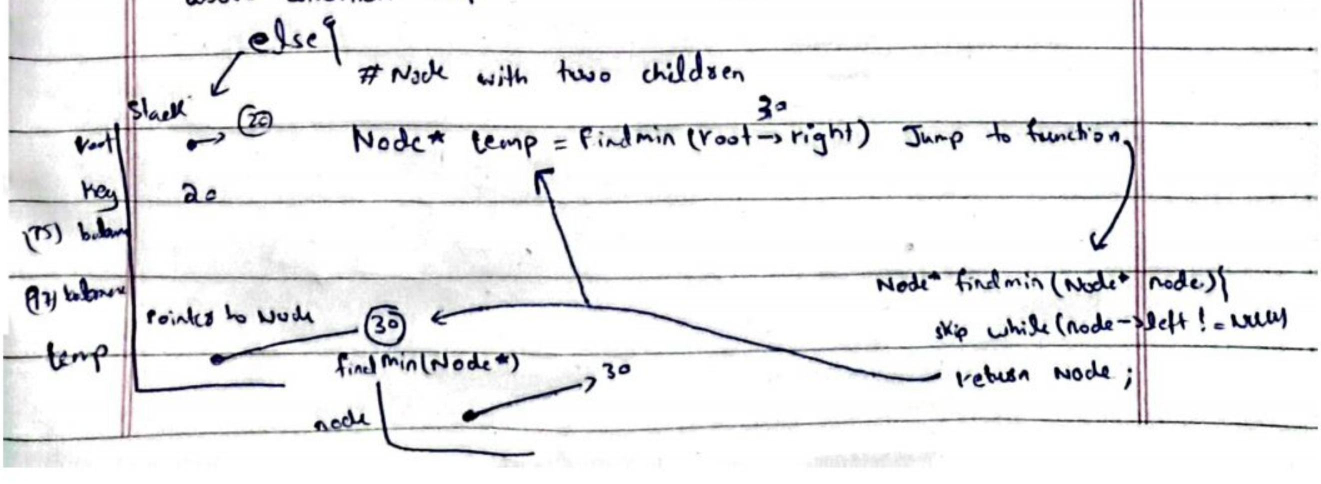
Node with two children

Node* temp = findmin(root→right) Jump to function

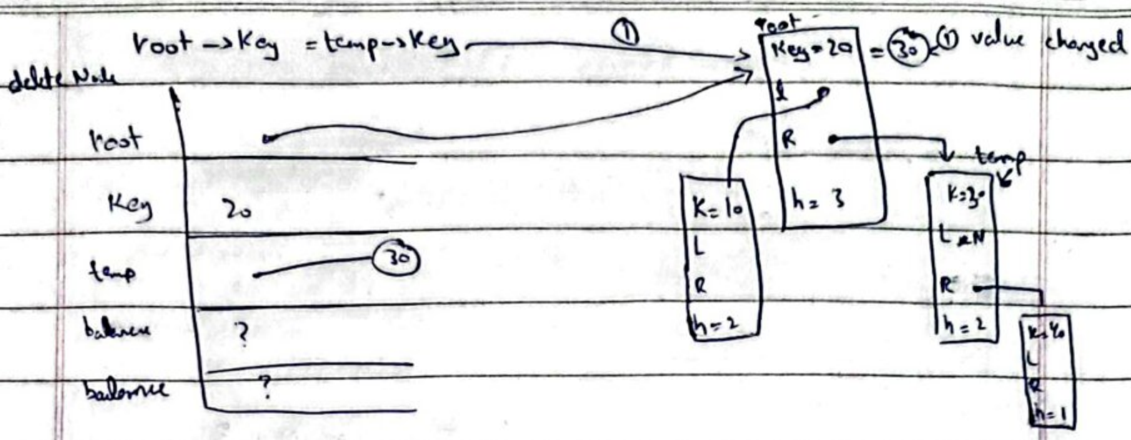
Node* findmin(Node* node){

skip while(node→left != NULL)

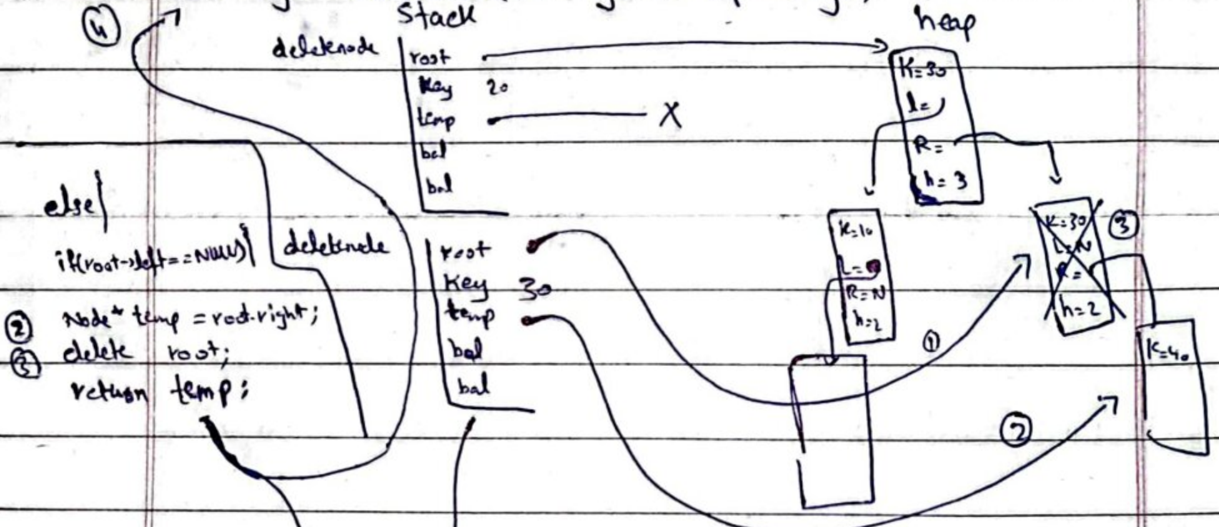
return Node;



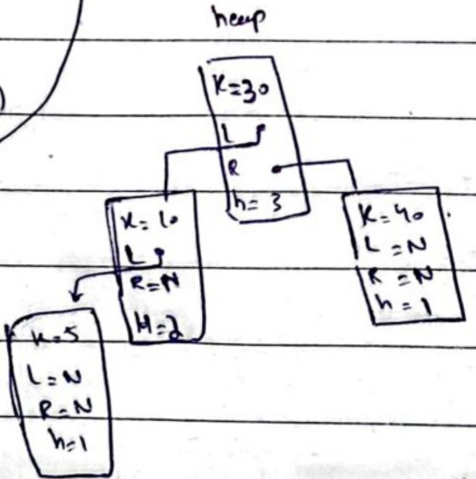
root -> Key = temp->Key



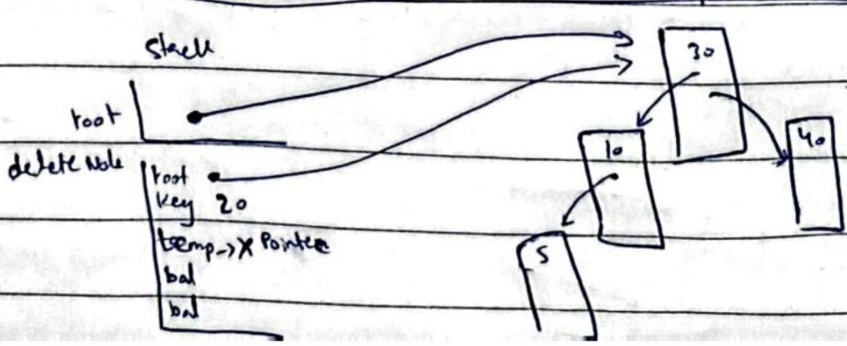
30 30
root -> right = deleteNode(root right , temp -> Key);



else{
if(root->left==NULL){
② Node* temp = root.right;
③ delete root;
return temp;



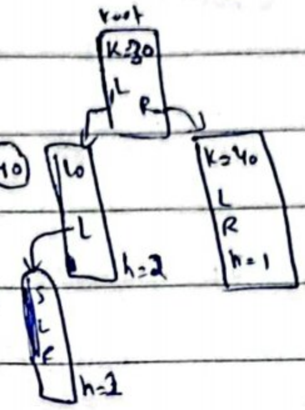
This will end after ④



root->height = 1 + man (height (root->left), height (root->right));

height (root->right)

node | •——— (40)



return node->height; (1)

⋮

1+ man (2, 1)

man (int, int

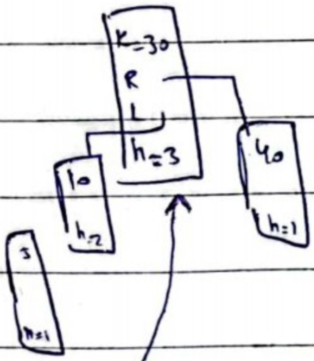
a | 2

b | 1

return (2)

root->height = 1 + 2 = 3

int balance = getbalance(root)
(30)



~~int~~ getbalance (Node*)

node | •———

return height(node->left) - height (node->right)
2-1 => 1
Same
height for both

node | ——> (40)

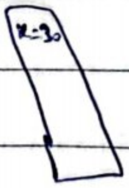
deletenode

root | ———

key | 20

temp | X

bal | 1



no condition will run because tree is balance.