Explanation of Question 4

Firstly I made a struct tree node consisting of long long int value , left pointer and right pointer .
I typedef it to Tree and PtrtoNode.

Functions that I made for Tree :-

```
PtrNode makenode(long long int e)
```

```
Tree insert(Tree t, long long int e)
```

```
long long int sum(Tree t, long long int n)
```

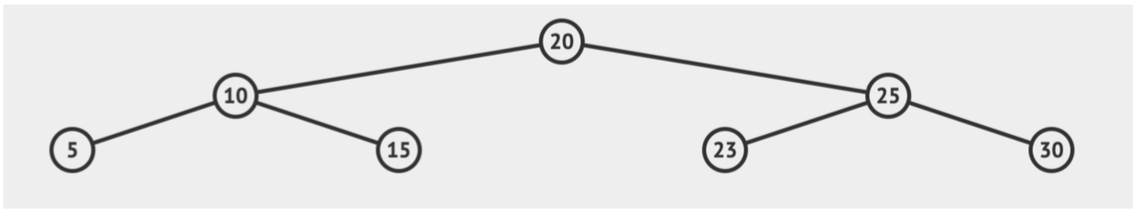
Makenode function is used to take input in long long int and then make a Tree pointer and return it which has all it's left and right pointer has NULL.. It's time complexity is basically O(1)..

Insert function is taking the Tree and long long int number and make a recursive function to insert the number . Base condition is that when the tree is NULL we makenode of the long long int number and return it and other conditions are like if t->key >e we go to the left of the tree and make a recursive call again for t->left
Similarly for t->key<e and t->right …….

Time complexity in worst case is O(n) for entering a single node ..
The worst time complexity for making the whole tree is O(n^2)….

Then for returning the sum I made a global variable which stores the sum …

The base condition for changing the values inside the tree is that if it is NULL we need to return the the number that we received bcoz we are not changing any node …
The second base condition is that if the given input tree is a leaf node we are changing its key to key+number we receive and also increase the sum h by key of the Tree .
The recursive definition is for going left and then going right ..
In the main I pass root of the tree with 0 as 0 has to be added in the smallest node.
Dry run for the base case is given below…



First call for function . t=20 and n=0 (It will go in else)

Second call (t=10,n=0) It will go in else again as it is NOT NULL and NOT a leafnode.

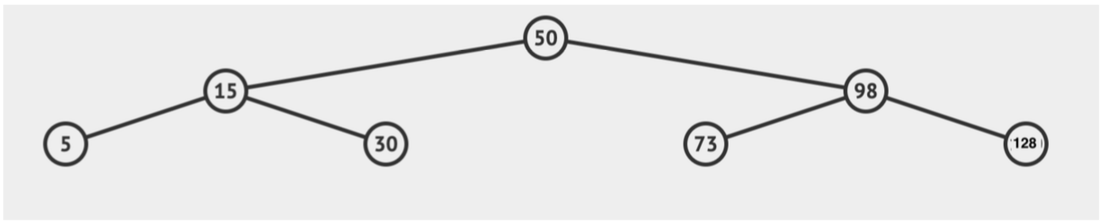
Third call(t=5,n=0) , It will go into second if as 5 is a leaf node .
Update its key by 5+0
Update h by h+5 making h=5 and then return 5

Returns into second call d=5 … Updating the t=10 key by t->key=t->key+d which will update it by 10 and make it 15.
Also adds 15 to the h which makes h=20
Then calls function 4th time (t=15,n=15) It will go into second if as 15 is a leaf node .
Update its key by 15+15(n)
Update h by h+30 making it 50 and then return 30
This will then return into second function call which will then return into 1st call with d=30;

Now updates the value of t=20 key with 20+30
Also adds t->key to h which making it 100 ..
Calls the function for fifth time (t=25,n=50) It will go in else as it is NOT NULL and NOT a leafnode.

Calls function for 6th time (t=23,n=50) , It will go into second if as 23 is a leaf node .
Update its key by 23+50=73
Update h by h+73 making it 173 and then return 73

It returns into 5th call making d=73 . it adds 73 to the key 25 making it 98 and adds 98 to h making it 271. And then calls the functions for 7th time (t=30,n=98)
As t=30 is a leaf node updates the value to 30+98=128 also adds 128 to h making h 399
And then it returns 128 and the whole recursion finishes
With h=399 and tree t as follows



PS:- The condition if(T==NULL) is used if t is not a leaf node and one of the child is missing and t==null is used …

To print the tree in level order I have made a struct queue .
 Que in , enqueue ,dequeue are basic functions to initialize queue and add element to the last and to delete the front element in the queue…

In main function I initialized the queue with the max elems as 1000100. And I enqueued the root of the tree to queue …

Now I wrote a function to calculate the height of tree ..

Height of a tree is calculated using a recursive function with the base case that tree is NULL return 0 . else conditions are such that we recursively find the left_ht and right_ht and return their maximum+1 .. Its time complexity is O(n) where n is number of nodes

Now to print the level order of a tree a loop which runs for number of height .
When i=0 queue has only root ..
It goes into the function levelorder
Queue is dequeued to get first member for a particular order ..
Now we check if that a member has a left child or not . If it has it enqueued to queue similarily for right ..
The condition i!=(t-1) is there as when i==(t-1) and when we dequeue from the queue we are dequeueing the first element of the next level . To prevent this I have implemented this if condition . The function has a time complexitiy of O(a) where a is number of nodes in a particular level .

The whole complexity for printing the tree in level order is O(n) where a is number of nodes acc to me bcoz I am going to each node only once and printing it….

Long long int is used in this function bcoz if every number has a value of order of 10^7 and there are 10^6 nodes which is given in constraints and the highest node in this will have order of 10^13 which is outside the capability of int so long long int is used …