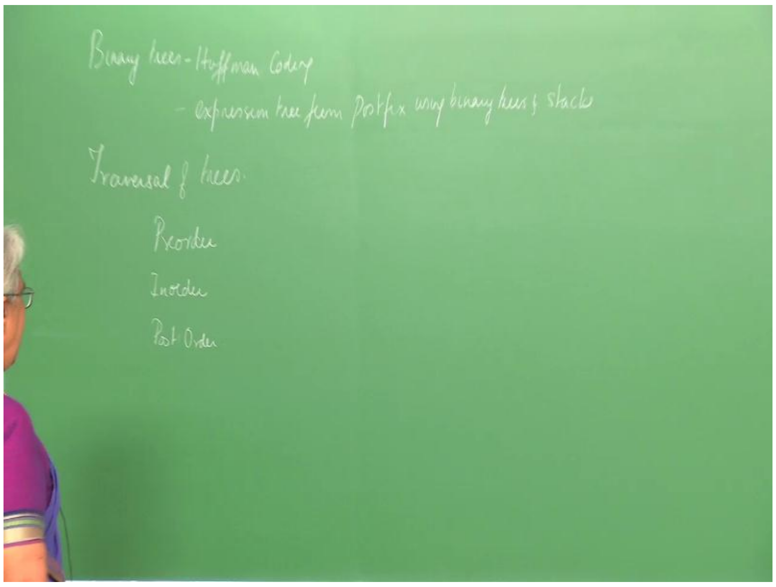
Programming, Data Structures and Algorithms

Prof. Hema Murthy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

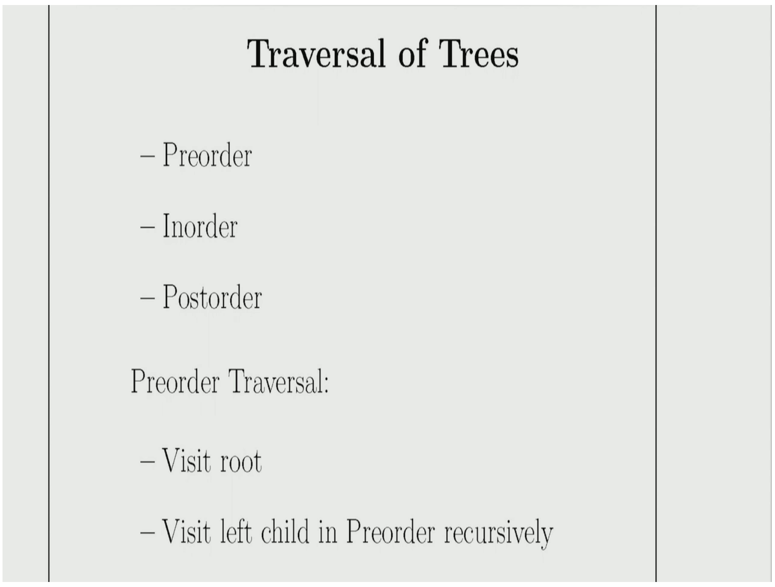
Module – 11
Lecture -- 50
Summary of list, stack, queue
Introduction to Trees
Motivation using family tree
Representation of a tree: array representation, linked list and leftmost child right sibling representation
Binary trees
Operations on binary trees: create, findchild, findparent

(Refer Slide Time: 00:15)


Binary trees - Huffman Coding
- expression tree from postfix using binary trees & stacks
Traversal of trees.
Preorder
Inorder
Post Order

Good morning, in the last class we looked at binary trees and we looked at two applications; one is constructing of a Huffman code using binary trees, and we also talked about the construction of an expression tree from a postfix expression using both binary trees and stacks. Today and then I talked about the different types of traversals of a binary tree, and we saw that when we take the expression tree and traverse it in order, we got the infix expression. And when we traversed it post order, we got the postfix expression and preorder we did not do, but you will get the prefix expression.

(Refer Slide Time: 01:00)



Traversal of Trees

– Preorder

– Inorder

– Postorder

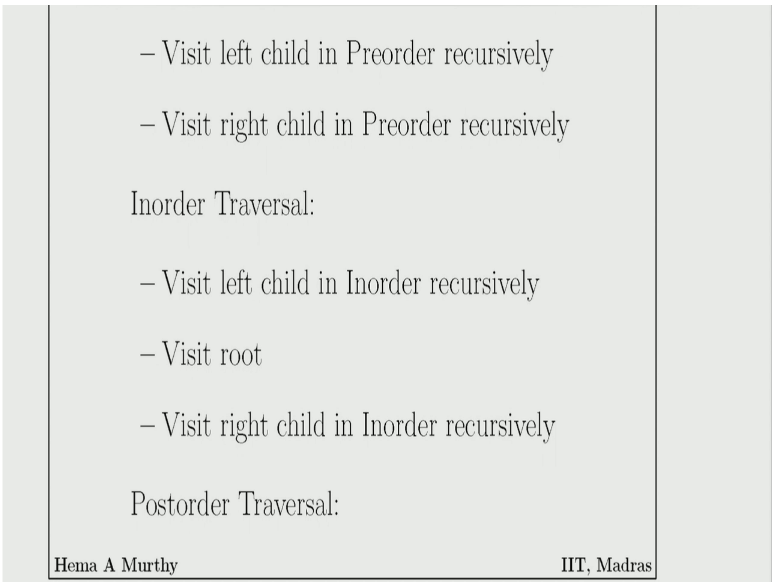
Preorder Traversal:

– Visit root

– Visit left child in Preorder recursively

So, today what I am going to talk about is I am going to, and so just to recap, what is preorder traversal. Preorder traversals visit the root and visit the left child in preorder, look recursively.

(Refer Slide Time: 01:06)



– Visit left child in Preorder recursively

– Visit right child in Preorder recursively

Inorder Traversal:

– Visit left child in Inorder recursively

– Visit root

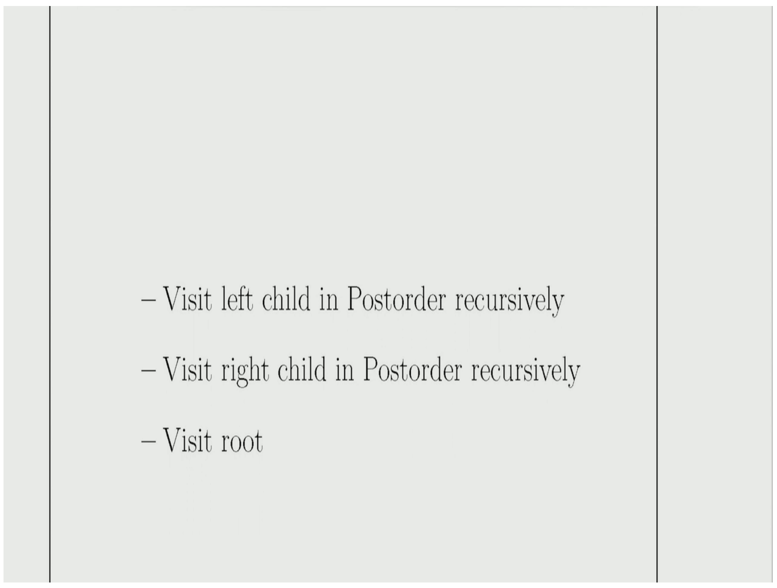
– Visit right child in Inorder recursively

Postorder Traversal:

Hema A Murthy IIT, Madras

And then visit the right child in preorder, recursively. Inorder is visit the left child in order recursively, then visit the root, then visit the right child inorder recursively.

(Refer Slide Time: 01:17)



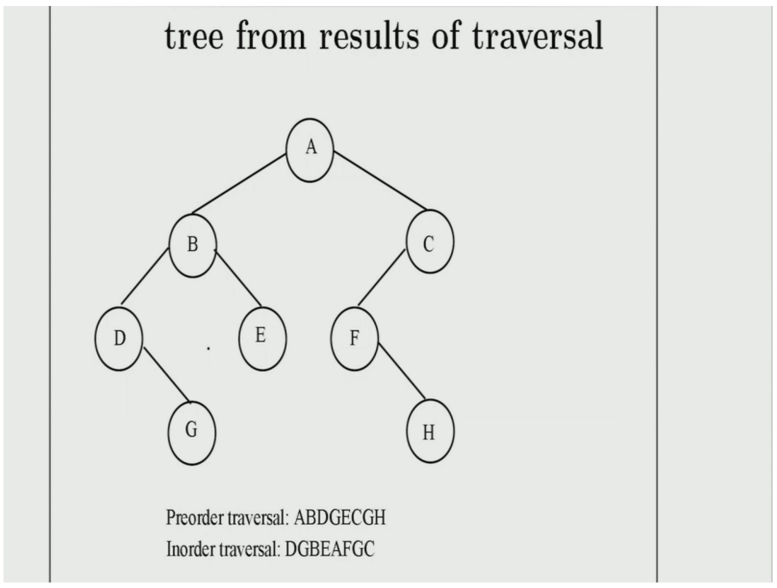
– Visit left child in Postorder recursively

– Visit right child in Postorder recursively

– Visit root

Now, and what is postorder? Visit left child in postorder recursively, visit right child in postorder recursively, visit root.

(Refer Slide Time: 01:28)



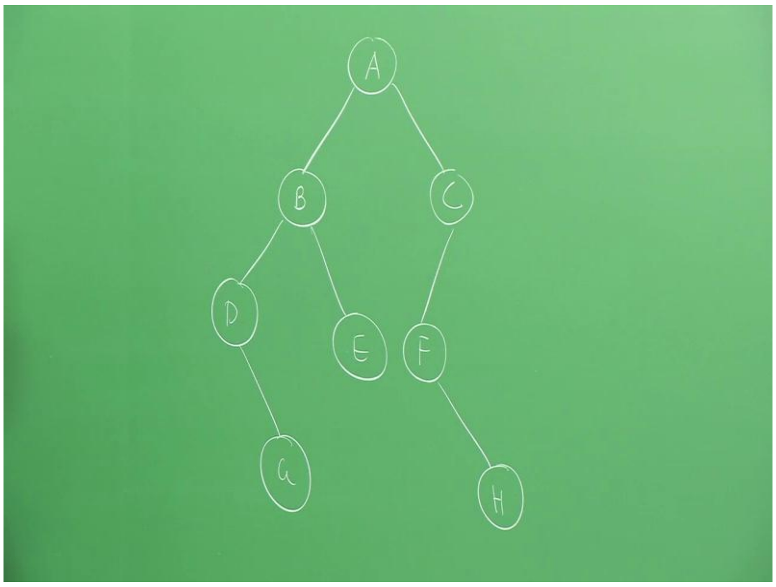
tree from results of traversal

Preorder traversal: ABDGECGH

Inorder traversal: DGBEAFGC

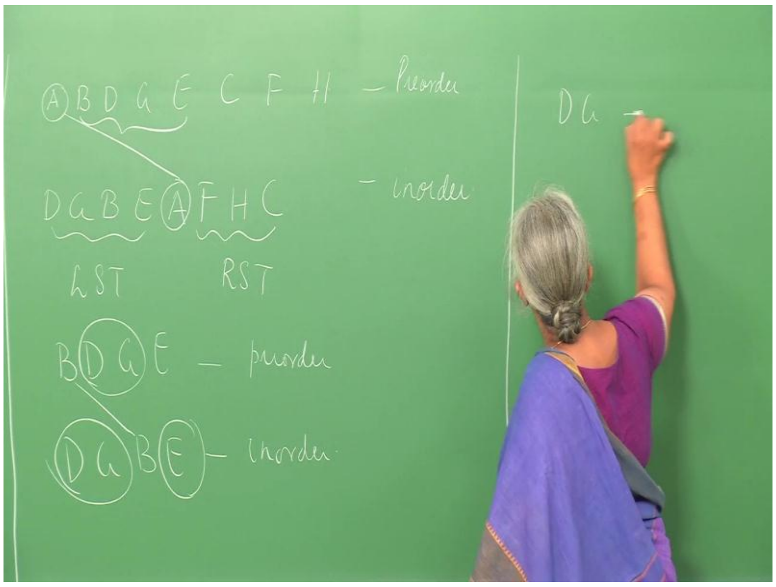
And postorder is what gives your postfix expression. Now, what we will do is suppose now we are given the traversal of the trees. Given the traversal of the trees, can we construct a unit tree. So, here we have a binary tree which is given to you and let me go through this example for you.

(Refer Slide Time: 01:46)



So, what we have? We have A here, B, C, D, E, F and G and H, some arbitrary binary tree. Well, what is that I am assuming, I have given a binary tree like this. And given this binary tree, we want to construct it, determinates it is preorder and postorder traversals.

(Refer Slide Time: 02:31)



Let us look at the preorder traversal. What are preorder traversals tell me? Visit root, then visit left child and right child recursively. That means, what am I going to do? I am going to the left child first, then find B, then this has to be done recursively. Remember, in any recursion what do we have to do? We go down till the recursion terminates and only then you can back turn. So, visit B, then I go down, this is the next node which I will display and D, G, then we will come back, because this left sub trees completely over now. Remember, we started with A, we came down to B, we came down to D, this has no child. Therefore, D then look at the right sub tree. Then I can be go back up, then all the sub trees of D are covered. Then we go to the node B, go back one level up, just as it could happen in recursion, and then you look at the right sub tree and it has no other children. Again, this has to be traversed in preorder. Therefore, you do not have since it has no children, E is the last symbol that will be output.

And then what happens? all these nodes have been visited, you go back here and go to the right sub tree and visit this in preorder. So, you will get C, F and H in that is what is said over here. So, preorder traversal, give me this, so this is preorder, now let us look at inorder traversal. What is inorder traversal? This is the left child recursively, then root and then the right child recursively. So, what happens now, I go down A, I go down to B, visit it is left child, left child, left child.

Because, the first node to be displayed will be the left child, then the root and then the right sub tree. So, we keep going down, what do I do? I come up to D, D has no left child, then I display the root. So, I display root over here, D makes right sub tree G. Then I go back one level up, we have B where it has a right sub tree E. For example, if this one had, then you would again done this in inorder. Everything has to be done inorder.

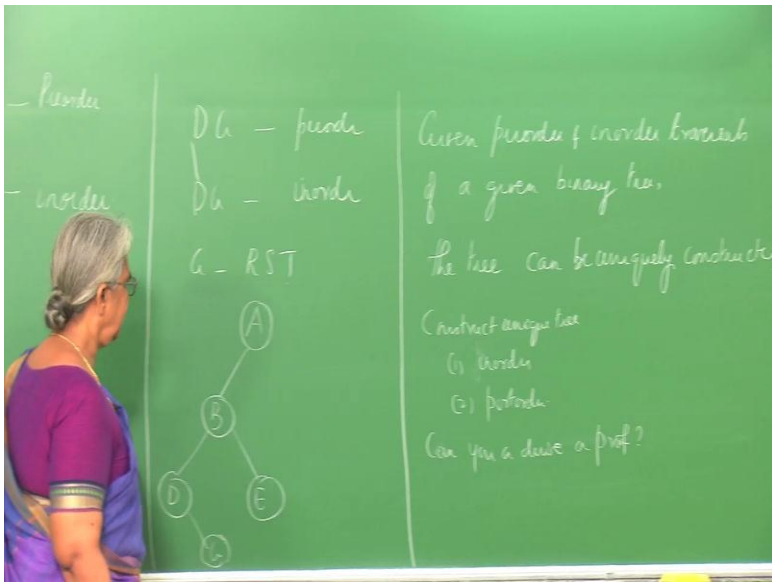
Then we go back, then now it display the root A. We come down here, again now you go to the right sub tree which extends B again, traverse recursively inorder, recursively inorder. So, it goes to it is left child, so now this will get displayed. Because, this one has no left child, then it is root F, H and C. This is what will be displayed. F, it is something wrong in that, instead of G there it should be H over there, F, H and C.

Now, this is the inorder traversal I encourage you to check this, you need at least inorder and preorder or inorder and postorder, constructed tree uniquely. So, let us look at this particular example and see how we can get this corrected. What we will you do is the following. Now, clearly in the preorder traversal the root is the node which is displayed first. So, now I look at this particular root node, this givens wrongly, so let we go with this fully.

So, what I will do is we know that the root node is always displayed first. So, now this must correspond to the root. Then what we do is we go to the inorder traversal, find out where that root node exists, this is where the root node exists. So, what happens in inorder traversal, the left sub tree nodes are before the root and the right sub tree nodes are after the root. So, what do we have now, we have now that D, G, B, E correspond to the left sub tree and F, H, C correspond to the right sub tree.

This is left sub tree and this is right sub tree, we know this information. So, I go back to the preorder traversal. So, this corresponds to these four nodes. That means, now I have B, D, G, E and D, G, B, E as the preorder traversal of left sub tree and this is the inorder traversal of left sub tree. Now, again what we have the first node in the preorder traversal must correspond to the root of the left sub tree, so I mark this. So, what is this mean now, D, G here belongs to the left sub tree, E belongs to the right sub tree. So, once again I mark this in the preorder traversal.

(Refer Slide Time: 07:55)

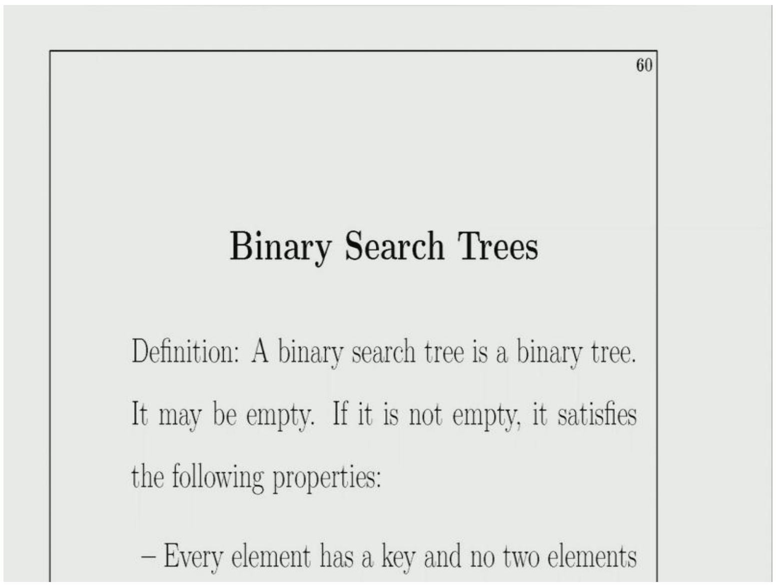


Now, what do have? We just doing the left sub tree now. So, it has D, G as the preorder and we have D, G also as the inorder traversal. So, we know that D has to be root type and it is in the same order, therefore this is the root. That means, in inorder traversal what is it mean, now G is part of the right sub tree. Therefore, now we can claim what is that we have done, we have A here, then we have B which was the root of the left sub tree.

Then we had D and G which were part of the left sub tree of A. So, what have we done, now we were essentially constructed this part of the tree. Similarly, here what we have now if you look at the right sub tree of the inorder traversals is only E and there only one node. Therefore, it has to come over here. So, in much as we can go ahead and construct the entire binary tree and now, you can convince yourself that given the preorder and inorder traversals of a given tree, given binary tree.

The tree can be uniquely constructed, you can convenience yourself about this. What it, I would like you to go back and see how you can construct a unique tree from, do this as homework, from inorder and postorder. Now, those of few words if you want to be more challenged, can you derive a proof to show that this proof can be done by construction and using induction, that preorder and inorder will give you a unique tree.

(Refer Slide Time: 10:39)


60

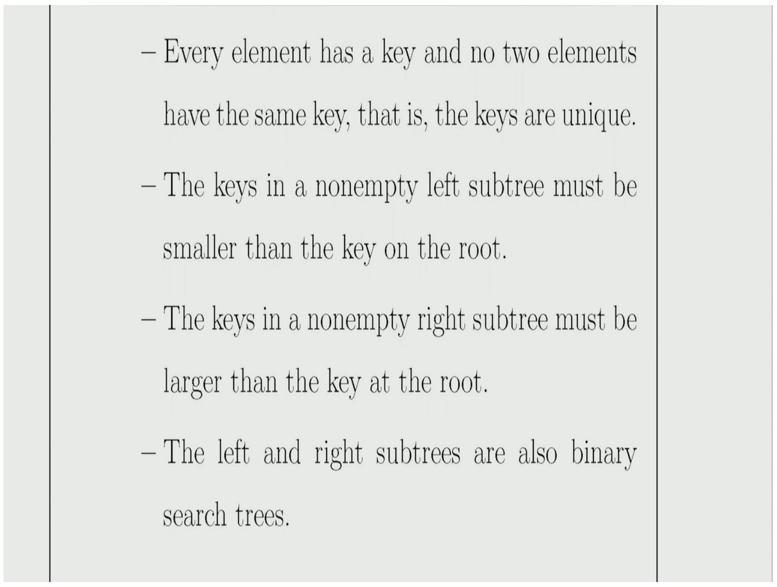
Binary Search Trees

Definition: A binary search tree is a binary tree. It may be empty. If it is not empty, it satisfies the following properties:

– Every element has a key and no two elements

So, let us next what we will look at is, so we have looked at the binary tree, where binary tree is one where every node has two children and both the children are binary trees. It may also optionally have only one child or it may optionally have no children. Even an empty binary tree, there also exists empty binary trees. This is what we have already seen. Now, today what I am going to talk about is I am going to talk about another type of application of binary trees called the binary search tree. Now, what is the binary search tree? A binary search tree is it is also a binary tree, it may be empty.

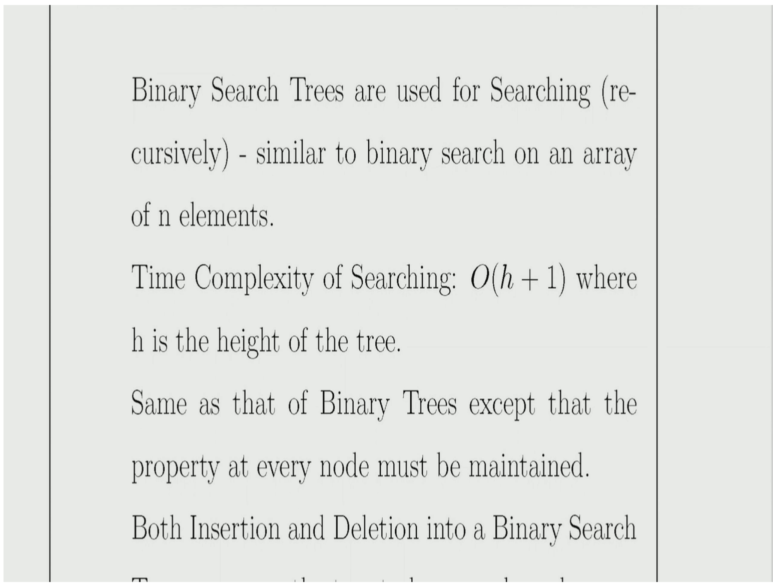
(Refer Slide Time: 11:29)



– Every element has a key and no two elements have the same key, that is, the keys are unique.
– The keys in a nonempty left subtree must be smaller than the key on the root.
– The keys in a nonempty right subtree must be larger than the key at the root.
– The left and right subtrees are also binary search trees.

If it is not empty, then it is satisfies the following properties. Every node, every element or node has a key and node to elements have the same key. What is the meaning of it that means, the keys are all unique and there is another property. The keys in a non empty left sub tree must be smaller than the key on the root. The key in a non empty right sub tree must be larger than the key at the root. The left and right sub trees are also binary search trees.

(Refer Slide Time: 12:06)



Binary Search Trees are used for Searching (recursively) - similar to binary search on an array of n elements.

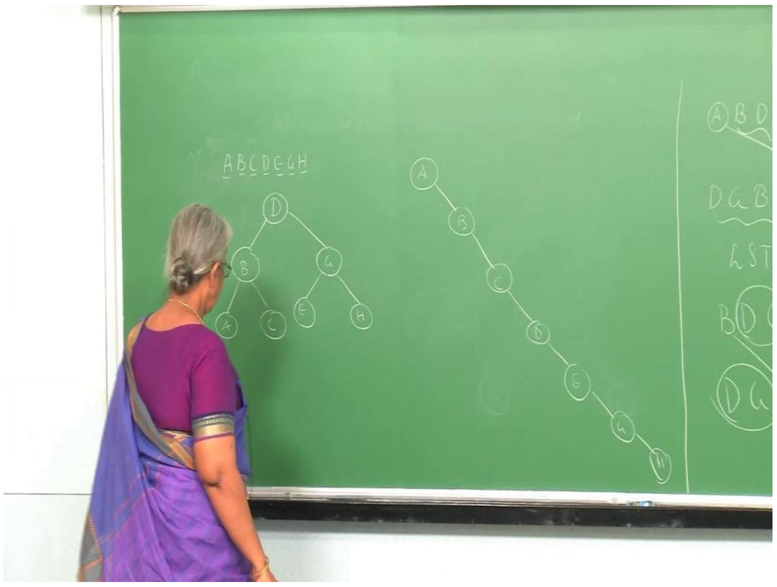
Time Complexity of Searching: $O(h+1)$ where h is the height of the tree.

Same as that of Binary Trees except that the property at every node must be maintained.

Both Insertion and Deletion into a Binary Search

Now, let us take an example and see how this will look like.

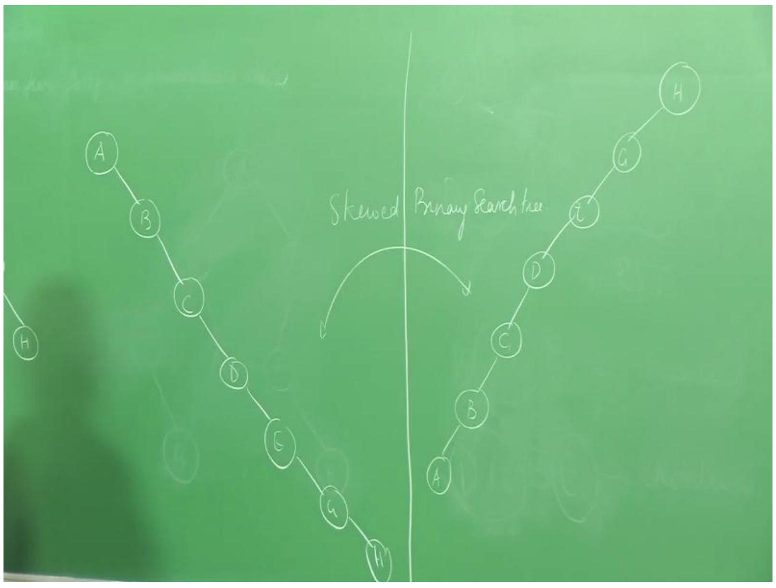
(Refer Slide Time: 12:30)



Let us say take the same letters of the alphabet. Let us say I have notice that all the keys are unique. Let us say each one of the letters of the alphabet is a key. Then this is a binary search tree. Notice that all nodes on the left sub tree have a key which is smaller than the root, I am assuming lexicographic ordering here and all the nodes on the right sub tree have a key which is greater than the root and that is true of every node.

For example, if take the node B here, left sub tree has smaller key values than B, right sub tree has larger key values than B, E here again smaller than G and right sub tree has larger value than the given node. Interestingly, this is also a binary tree.

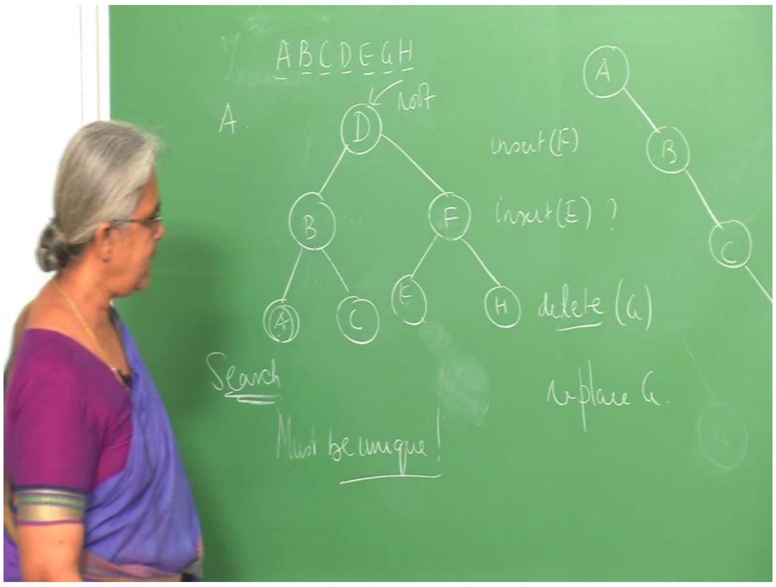
(Refer Slide Time: 14:23)



And by the same token, we also have that this is also a binary search tree. Now, what will we do is we look at the sub tree. So, basically why this is also a binary search tree and this is also a binary search tree? By definition what we have, a binary search tree has keys that are unique, no doubt about it, all the keys are unique. And what did we say, elements on the left sub tree, binary search tree can also be empty and elements on the left sub tree should be smaller than the root, we just smaller than the elements on the right sub tree.

And this is true for this tree and it is also true for this tree. These two trees are what we called skewed binary search trees. What is the application of this binary search tree? For example, you can put the roll numbers in a class in a binary search tree and search for a particular, if you want to find a marks of a particular student, you can find the marks of particular student using a binary search tree.

(Refer Slide Time: 15:46)



How would I search now? This corresponds to the root. Suppose, I want to find out what was the marks that A obtained in this tree, then I give the key and say, key is A, go back. Let us say this is additional information along with A, I only given the key here. Marks and other things are stored over here, I compare A with D. It does not match, then clearly, but A is less than D. So, go down to left sub tree, compare with B, A and B again is not matched, but A is less than B.

Therefore, I go down. It is left sub tree and I find this particular node and I can retrieve all the information corresponding to the given node. This is about searching, the operation of search, I want to perform. Now, if I want to insert a node what happens now, let us say I want to insert F, E, G, H is what I have. Suppose I want to insert the node F, then I compare with the root where should it go now. Compare with the root, then clearly it is not matched, but F is larger than D.

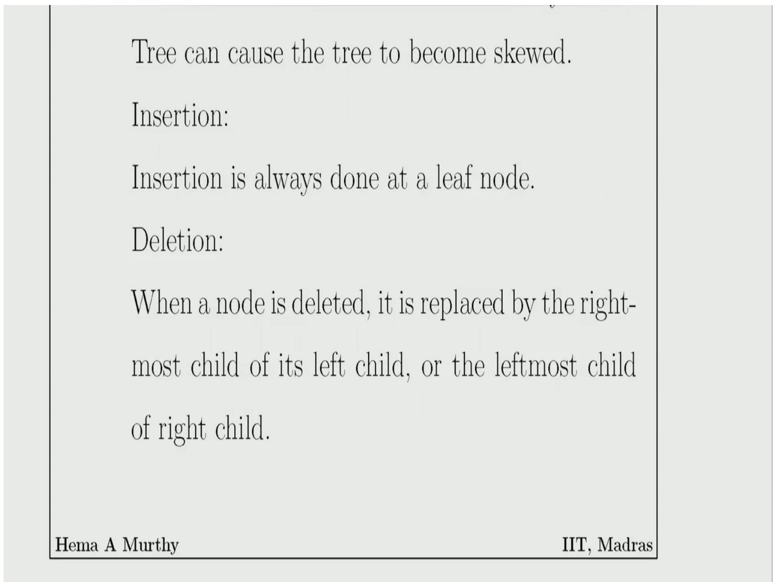
So, I go down to sub tree, then I compare with G, G and F are again matched, but clearly G is larger than F. If I go down it is left sub tree, I compare with E, E and F are again not matched, but F is clearly larger than E. Therefore, I go down here and I insert F over here. Now, suppose I wanted to insert E, can this be done? I compare E with D again starting from the root, clearly E is larger than D. Therefore, go down the right sub tree, I compare with G, G is larger than E, therefore I go down it is left sub tree.

I found a match, but since already the binary search tree has a key E. Clearly, I cannot insert the node E, because the keys in a binary search tree must be unique. This is the very, very important property. Keys in a binary search tree must be unique, this has to be satisfied. Now, suppose I want to delete the node G, so what are the operations you are looking at search, we looking at in insertion, we are looking at deletion.

This is also common, A student leaves a program and you want to delete the particular name from the list. So, if you want to delete the particular node G, then what should I replace it. I should replace it with some node, such that the binary search tree property is not satisfied. We can do one of two things, we can replace, suppose I want to delete G, I can replace G with either the largest node on the left sub tree or the smallest node on the right sub tree.

Why do we do that, because then we get guaranteed that the tree will still remain to be a binary search tree. So, this is exactly what we do. So, I put F over here, the largest node on the left sub tree, so I find out. So, I go to the right most node on the left sub tree, find that and put that node over here, delete it, but what might happen is this node might have some left children. But, since clearly in this particular case, F did not have any left children. But suppose F had some left children and what do I do? I take this left children, make it the right children of E, because that is, because all of them have to be larger than this given node. So, this is your operation of deletion.

(Refer Slide Time: 19:47)


Tree can cause the tree to become skewed.

Insertion:

Insertion is always done at a leaf node.

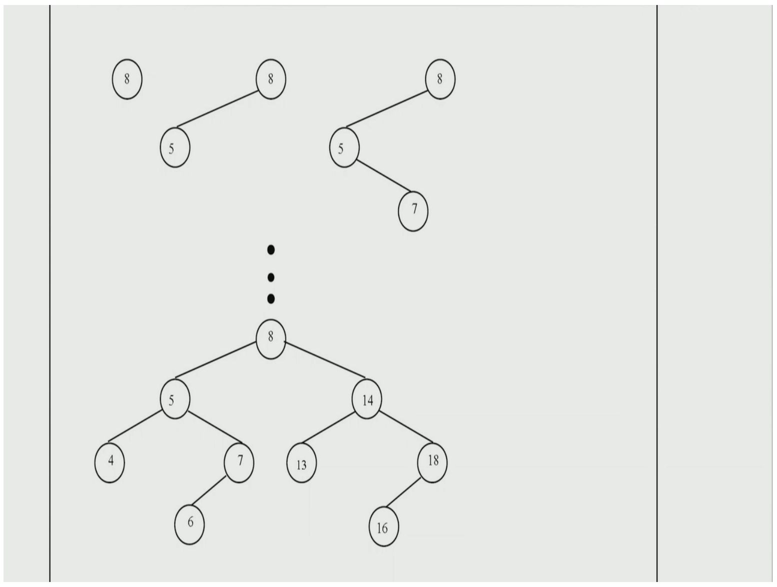
Deletion:

When a node is deleted, it is replaced by the right-most child of its left child, or the leftmost child of right child.

Hema A Murthy IIT, Madras

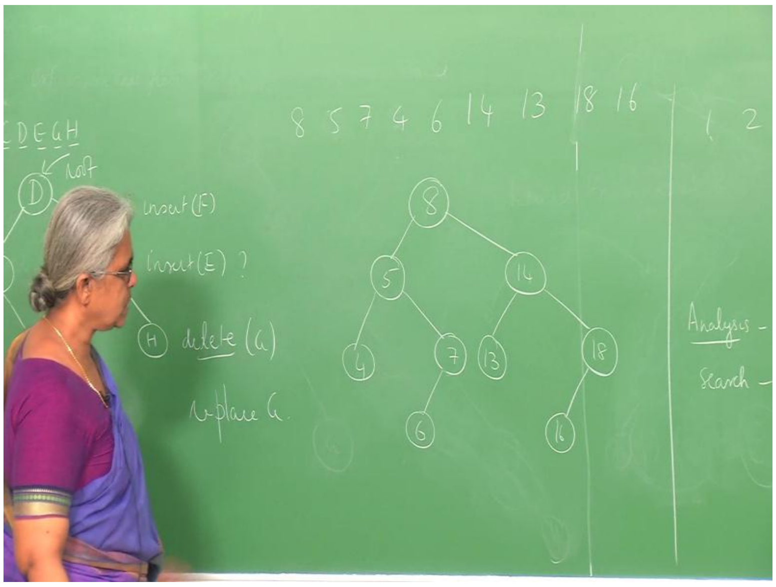
So, the operations on the binary search tree that we would like to perform are insertion, search insertion and deletion. So, these are the three operations that we would like to perform.

(Refer Slide Time: 20:03)



Here is one same example, I have done it with numbers in the slide over here to show the operation of insertion. Notice, how the… Now, let us see how do you get these skewed binary trees basically, if I start out with.

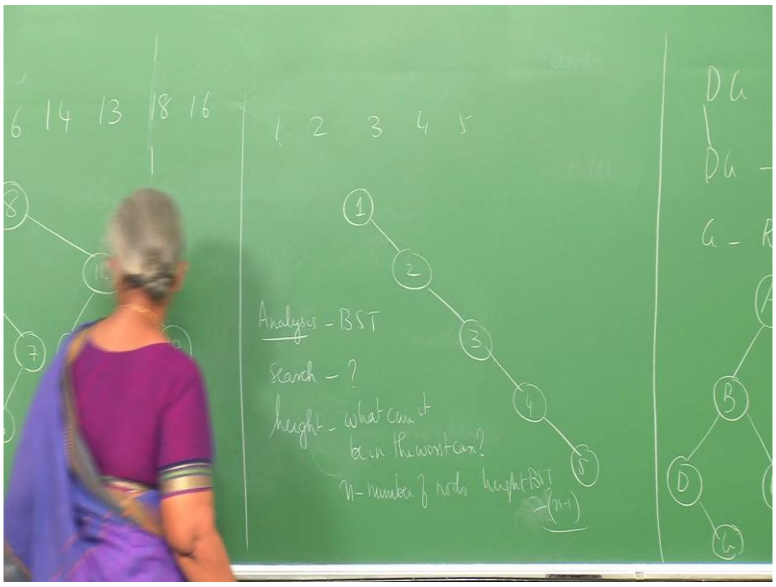
(Refer Slide Time: 20:35)



Let us look at the insertion over here, let us say I have been given 8, 5, 7, the keys 8, 5, 7, 4 and 6, 14, let us say 13, 18 and 16. Suppose, these keys are given and you want to insert them into a binary search tree. This will give us an idea, how, why these trees become, because we are always inserting that just let us go through the example. I have given a partial example in the nodes. So, I have 8 here, so there is initially an empty tree.

So, into the empty tree I insert the node 8, the next come across the node 5, 5 is smaller than 8, therefore I make it the left sub tree. Notice that again it is being inserted as the leaf node. Then 7 for example becomes the, again you compare with 8, 7 smaller than 8 greater than 5, therefore this has to become the right sub tree of this 4 compare with 8 smaller, but smaller than 5, therefore 4 goes over here 6 compare with 8, compare with 5, compare with 7 and then it comes over here and so early here, compare 14 and 13, 18 and 16. Always what we are seen in this process, all the nodes simply get inserted at the leaf node at every stage.

(Refer Slide Time: 22:19)


6 14 13 18 16
1 2 3 4 5
Analysis - BST
search - ?
height - what can it be in the worst case?
n - number of nodes height BST
(n-1)

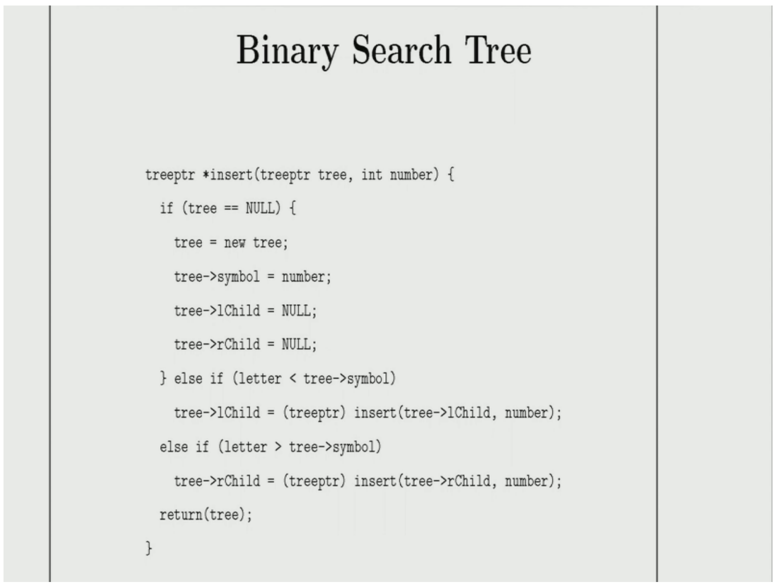
Now, suppose is that giving the sequence I had given you the sequence 1, 2, 3, 4, 5, then initially there is an empty tree, I insert 1, then I insert 2 it lies right larger than this, then 3 then 4 and then 5. So, this is how a binary search tree can become very skew. So, let us do some analysis of the complexity of this binary search tree. What are the costs of search now? How expensive can we search now? So, clearly in the worst case what is that it going to be, I may have to traverse down the height of the tree.

Now, what is the height of the tree now? Basically, you compute the top from length of the path from root to the leaf node 1, 2, 3. To the worst that you have to do, suppose I want to compare with 6, if the key that I am searching for the 6, therefore, the complexity becomes the height of these trees 1, 2, 3, but I have to make 1, 2, 3, 4 order height plus 1. So, I am introduced a new concept today, the height of the tree.

So, the height of the tree is the number of comparisons plus 1 is set to the maximum number of the comparisons that you may have to perform, if you want to search for a given node and this is also true for deletion, because I want to delete or insert. For example, I have to come down and then move back up and deletion, whereas in insertion again, insertion happens at the leaf. Therefore, I have to go down to the height of the tree.

What can be the height be in the worst case, height what can it be in the worst case? In the worst case, what can happen is you can have a completely skewed tree and the height can be n minus 1. If n is the number of nodes in the tree nodes, then the height of the binary search tree can be asked all as n minus 1. So, when you are talking about order of h plus 1 comparisons, in the worst case you have to do order in comparisons in the binary search tree.

(Refer Slide Time: 25:09)



Binary Search Tree

```
treeptr *insert(treeptr tree, int number) {
  if (tree == NULL) {
    tree = new tree;
    tree->symbol = number;
    tree->lChild = NULL;
    tree->rChild = NULL;
  } else if (letter < tree->symbol)
    tree->lChild = (treeptr) insert(tree->lChild, number);
  else if (letter > tree->symbol)
    tree->rChild = (treeptr) insert(tree->rChild, number);
  return(tree);
}
```

So, this kind of completes the and I am just, let me give you a small implementation of insertion, here is a small segment of C code which is given here. So, let us say that you have a pointer to tree as we had in the Huffman tree. So, what we are doing over here is if tree is equal to null, then what are we doing, we are creating a new symbol. In this particular case, I am assuming a binary search tree which only has numbers, otherwise what I am doing, I am comparing if it is not in empty tree, I compare with tree symbol and then what we do, we insert it.

If it is smaller, then you go to the left sub tree and then do the insertion again, insertion is called recursively over here. Similarly, otherwise if letter is greater than B given node, then what we are doing, we are inserting in the right sub tree. Notice that we are not doing anything, if the symbol is already present in the tree. So, this is the very important point in binary search trees. So, there is something that is a small segment of code over here, recursive implementation of the insertion into a binary search tree. There should be a correction here, this letter should read number, please correct this. It is else if number less than tree symbol, number less greater than trees, so this is about binary search trees.