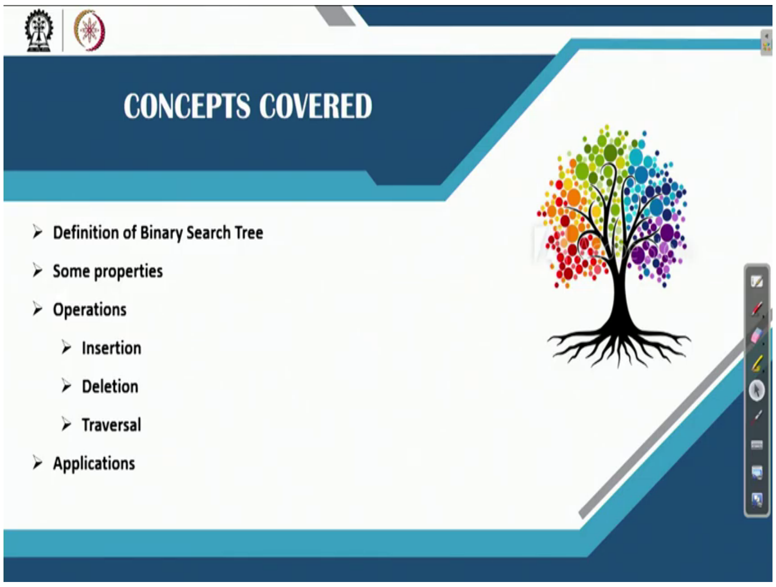
Data Structures and Algorithms using Java
Professor Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 29
Binary Search Tree

So, let us discuss about a specific binary tree, it is called a binary search tree.

(Refer Slide Time: 0:34)



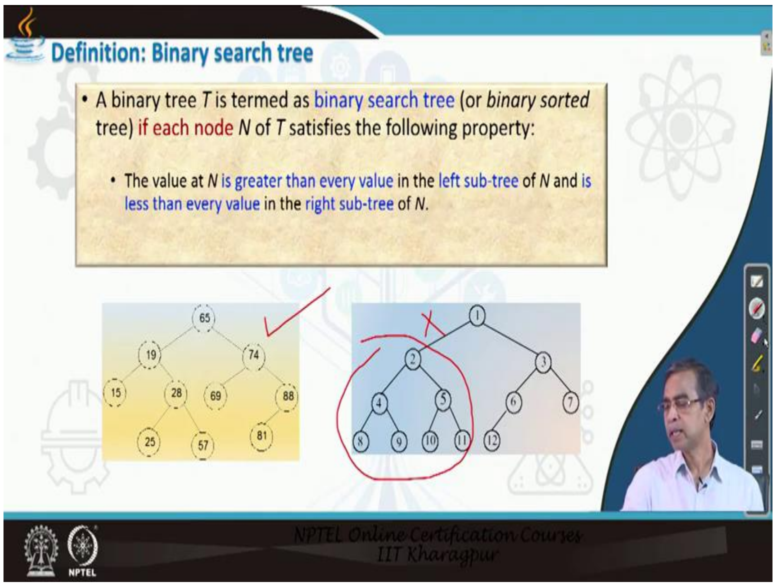
CONCEPTS COVERED

- Definition of Binary Search Tree
- Some properties
- Operations
  - Insertion
  - Deletion
  - Traversal
- Applications

So, today's topic include how we can define a specific binary tree, namely binary search tree the name of that tree is called Binary Search Tree. And what are the different properties that this binary search tree holds? And then we will discuss about how to perform the three different operations, namely insertion, deletion and traversal on this binary search tree. And finally, we will conclude giving an application of this type of tree.

So, it is a binary search tree. Now, let us define when a binary tree will be termed as the binary search tree.

(Refer Slide Time: 1:11)



Definition: Binary search tree

- A binary tree *T* is termed as binary search tree (or *binary sorted* tree) if each node *N* of *T* satisfies the following property:
    - The value at *N* is greater than every value in the left sub-tree of *N* and is less than every value in the right sub-tree of *N*.

So, there is a precise definition, which I have mention here, if you follow this definition, a binary search tree is a binary tree. Also, sometimes in some book it is called a Binary sorted tree, why the name so I will tell it. But a tree binary tree will become as binary search tree or binary sorted tree if each node in this tree satisfy, these are the property, this property, what is this property?

The property that each node in this tree is that it needs value should be greater than every value in the any node in the left sub tree and its value, the value of the node should be less than any value in the right sub tree. Now, let us come to an example, for example, any node.

So, let us start this node. Now if you see this node value 65 and this is a left sub tree and you see this value is less than any value, any value or any value in any node in this left sub tree. Similarly, this is a right sub tree of this node so this is a node N. Then if you see, then the value of this node is greater than the value of any node in this right sub tree.

Now, again, continue the same procedure to any other node, let the node B here let the node B says 74 then if you see its left sub tree is 69 and right sub tree is this one only, okay. Now this value is greater than this one and less than any value in this part.

So, this basically this property satisfy for all nodes. Now if it is, if it satisfies this property that I have mentioned, if it satisfies this property, this property basically which satisfy this property, then we can say that the binary tree in this case, for example is a binary search tree.

Now, here is an example you can check whether this also satisfied the property of binary search tree or not, the answer is yes. So, this is also another example of binary search tree. Now I am sorry, this does not satisfy the property of binary search tree, sorry. So, this basically there is a, okay. So, this does not satisfy the property of a binary search tree because here this node is basically should be greater than the value of any node here, it is not there.

So, this is not a binary search tree. However, this is a binary search tree. Anyway, okay. So, we have learned about that specific property and this is the specific property. Now, if we impose these properties into a binary tree, then it can resolve many problems that we have encountered once while we are trying to perform insertion, deletion and merging operation to a general binary tree where there is no properties are there.

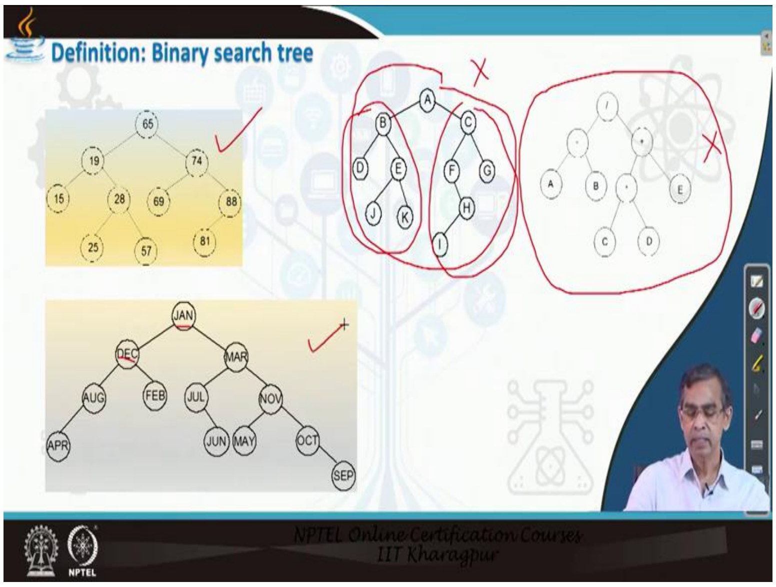
Now, so it is the concept now, this kind of tree is called a binary search tree because this tree has immense application for the searching purpose. So, that means we can quickly search an element in a binary tree if it satisfy the property of binary search tree that is how as it is useful for searching purpose, then this binary tree is called binary search tree.

Now, it is also called binary sorted tree, this is because if we apply a particular traversal namely in order traversals, then output of these binary search tree will always give you the element in the sorted order. If it is number then number should be available as an output of ascending order of the numbers.

If it is a string again in the dictionary order like this. So, as one traversal gives a sorted output for a binary tree then this is also called binary sorted tree. In other words, what you can say is that these binary search tree can be applied for the searching purpose as well as sorting purpose.

So, if you want to sort a list then idea will that you create binary search tree and then perform a particular traversal. If you want to have a regular searching operations for a list of data, then you have a binary search tree, then your searching can be made easier. So, this is why the name of this tree is so that it is called a binary search tree or binary sorted tree.

(Refer Slide Time: 6:06)


Definition: Binary search tree
NPTEL Online Certification Courses
IIT Kharagpur

Now let us consider a few more example, and you just okay I give it that responsibility to check whether they are they are binary search tree or not. For example, this is the tree whether this is binary search tree or not. This is not a binary search tree because the ordering property ordering means greater than less than cannot be applicable to this kind of form.

So, this is not a binary search tree. However, ordering property can be applied here because we can check that whether A is before B or after B means greater than or less than whatever it is there then ordering and now all the ordering is applicable, but it does not satisfy the property of binary search tree because A greater than any part of this B does not hold good or it is less than any value. It does not hold good like this.

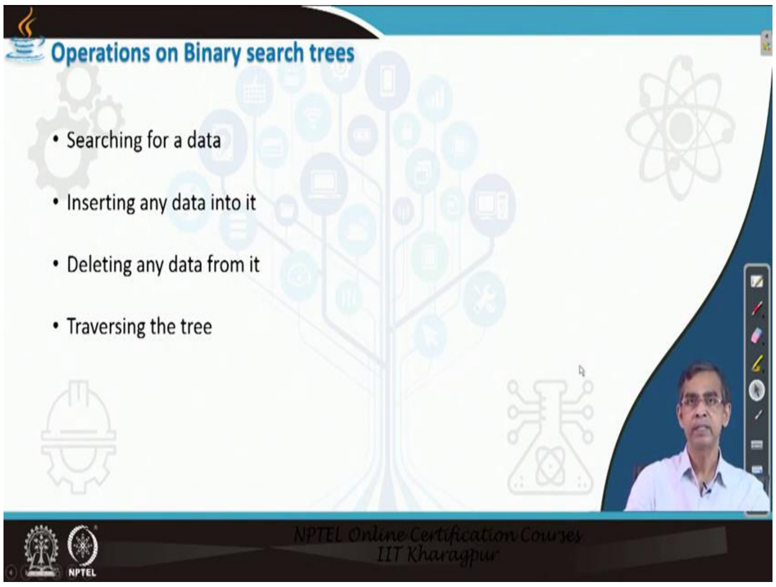
So, although this basically this nodes satisfied the second property that it value is less than any value, this one but it is greater than this is not satisfied. So, this way it is ordering is also applicable here are possible, but it is not a binary search tree. On the other hand, if you consider this, this is an example and you can check that it satisfy all the properties of the binary search tree.

So, it is a binary search tree. Now, this is again, the word the word is compatible, for example, January and December we can say that January is lesser than December because in a collecting sequence or in a dictionary order January appears before December or whatever it is here January appear to be after December actually, sorry January appears after December and similarly say August appear before December, whatever it is here.

So you can see this way they were satisfy binary search tree or not, that you can check it and that that's, okay it gives an idea that it is also a binary search tree, like.

So, now the two conditions whether the ordering rule can be applied to the node, which basically storing which binary search tree stores, it is one it is one important consideration. Second thing is that and if each node follows this property or not. So, so these are the way that binary search tree can be defined.

(Refer Slide Time: 8:42)



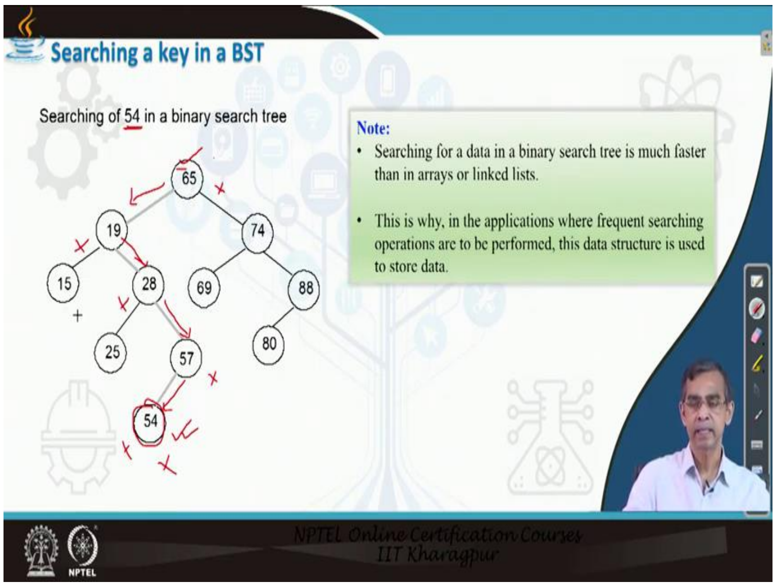
Operations on Binary search trees

- Searching for a data
- Inserting any data into it
- Deleting any data from it
- Traversing the tree

NPTEL Online Certification Courses
IIT Kharagpur

Now, let us let us try to understand different operations on the binary search tree different operations, namely searching we will see how it can be done given a binary search tree and then insertion, deletion and traversing the tree. So, these are the 4 operations we will likely try to understand.

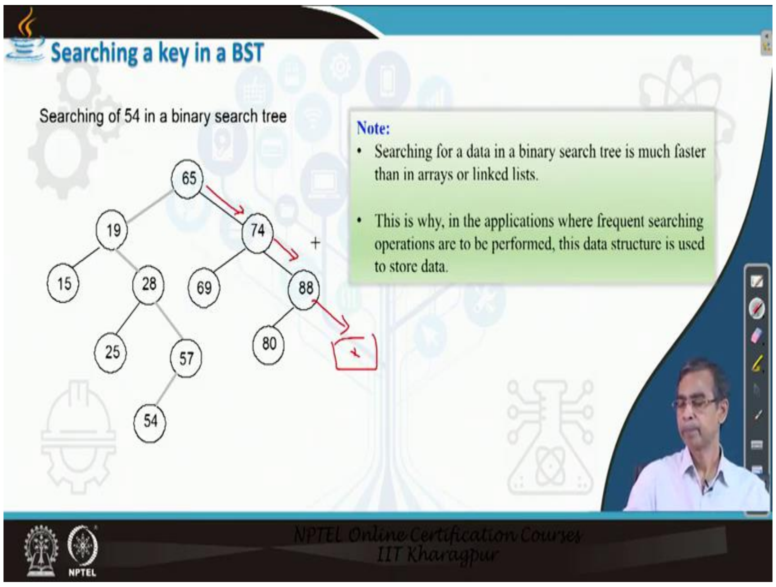
(Refer Slide Time: 8:55)



Searching a key in a BST

Searching of 54 in a binary search tree

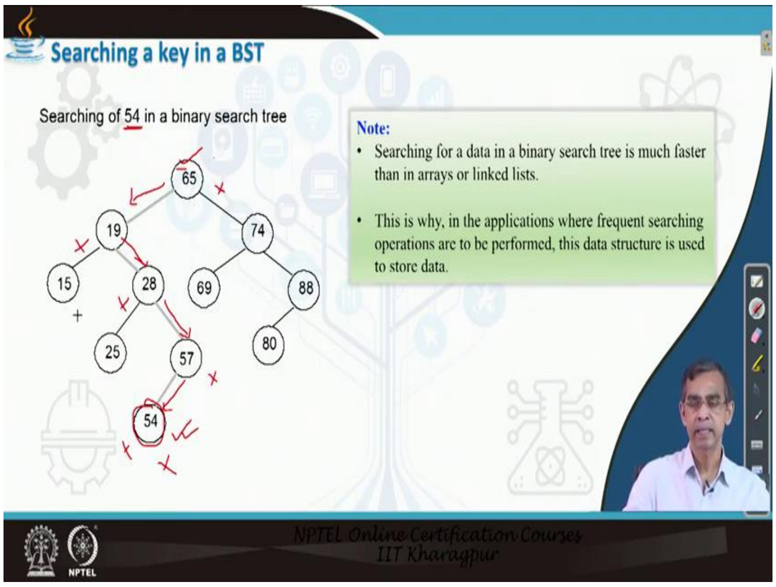
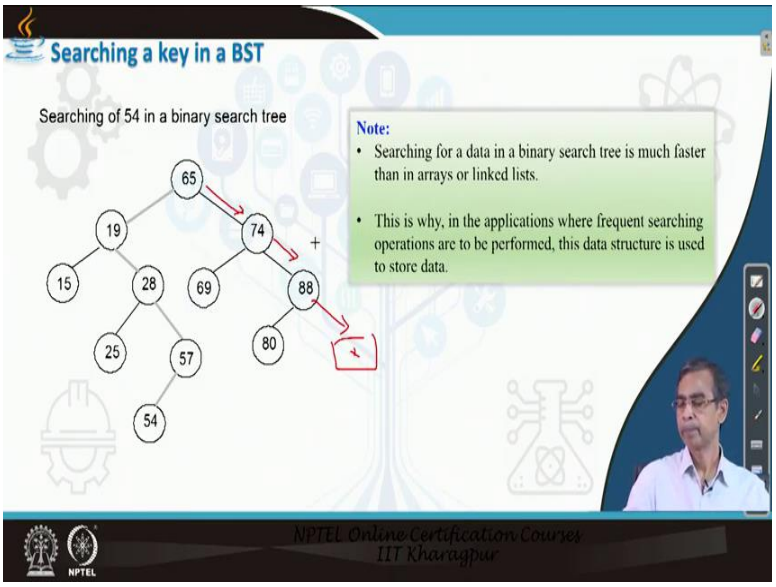
Note:
- Searching for a data in a binary search tree is much faster than in arrays or linked lists.
- This is why, in the applications where frequent searching operations are to be performed, this data structure is used to store data.

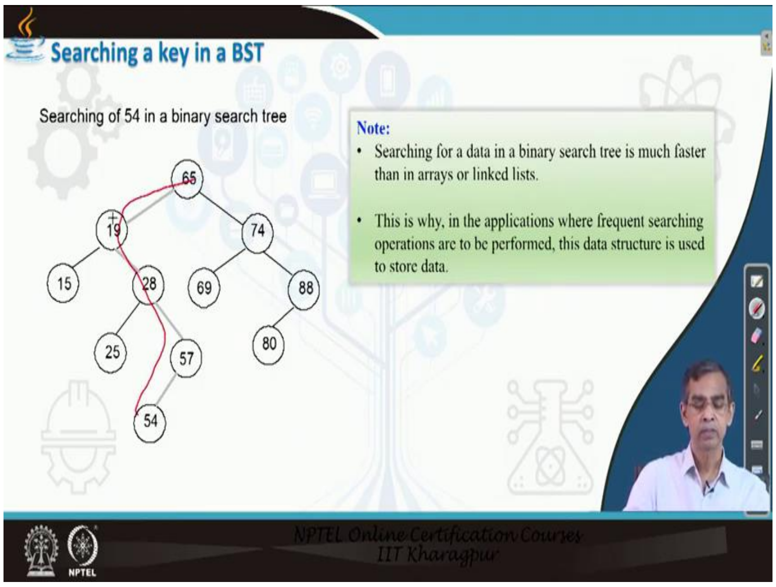


Searching a key in a BST

Searching of 54 in a binary search tree

Note:
- Searching for a data in a binary search tree is much faster than in arrays or linked lists.
- This is why, in the applications where frequent searching operations are to be performed, this data structure is used to store data.


Searching a key in a BST

Searching of 54 in a binary search tree

Note:

- Searching for a data in a binary search tree is much faster than in arrays or linked lists.
- This is why, in the applications where frequent searching operations are to be performed, this data structure is used to store data.

Now, searching is very simple. This is because see suppose in this example. We have to search for an element let the element be 54 that we have to check in this tree, whether 54 is present or not. So, searching can start place from this root node only because this is a starting point.

Now we have to compare 54 with this node if 54 is equals to the current node. Then we got it so search is successful and the element is present. Now, if it is not there, then we have two ways either this sites or that site now we will follow one way, which basically according to these because see we will we will not follow this way, this is because if we follow all this part, then basically we will encounter only the value, which is greater than 65 because our searching is for 54.

So, we should not follow this path, rather we should follow this path. Now, following this path from this starting from this node we will arrive to this node 19, again, we compare 19 and 54 and we will see that 19 and 54 it does not match. And the next two paths, this path of this path but we should not go this path because in this path, all the value should be less than 19.

So, no way of no results will be obtained if we follow this one. So, we will go to this path, then we will compare 28 and when 28 is compare with 54 it is does not match. Then we have the two options, either these sides or that side. But these sites is not a fruitful site because all elements in this path if any than they are less than 28 but we are searching for 54.

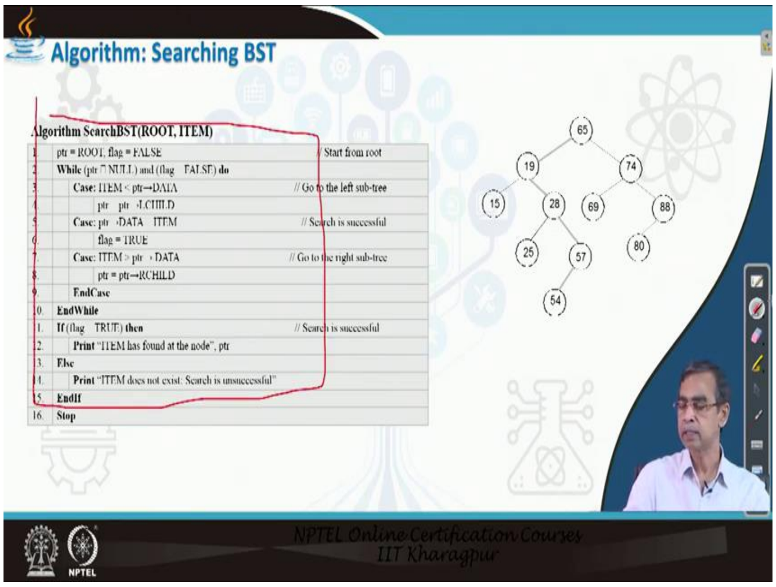
So, we will follow this path. Now whenever we follow this path we finally learned that this one 57 and we can see that 57 is not the same as this one. Then we have to two ways this one but this part has an empty. So, no way of going there. So, now we have to only way that 50 here it is there.

Now if we find the 54 here then we can say the search is successful suppose in link of 54 and here the element maybe say 53 then definitely our search will fail here because here we reached here and then there is no path and we did not find the elements, this means that 54 is not present in that net.

So, in li of 54 if we can search for 53 it will give the fail case. So, this is a procedure so for the searching element is concern. We follow the same tricks up to hundred. So, what is the idea? Idea is, again, that we can start from here, then we have to decide that this path or this path so 100 should be this path. Then here we should come here and then here we should come here, and ultimately that means the search is unsuccessful. So, this is basically the idea about that 100 is not present in this structure.

So, this is idea about searching. And you understand that searching is very simple because starting from the root node, we have to visit not all the nodes, but only fewer nodes. What is the fewer nodes or which nodes it is basically at the most, which is a longest path it is there for example in this binary search tree this is the longest path. So, and longest path consist of so, so many nodes for example in this way 5 nodes. So, maximum five comparison, although this node contains around 12 elements so out of 12 only 5, so the comparison will takes very less.

(Refer Slide Time: 12:52)

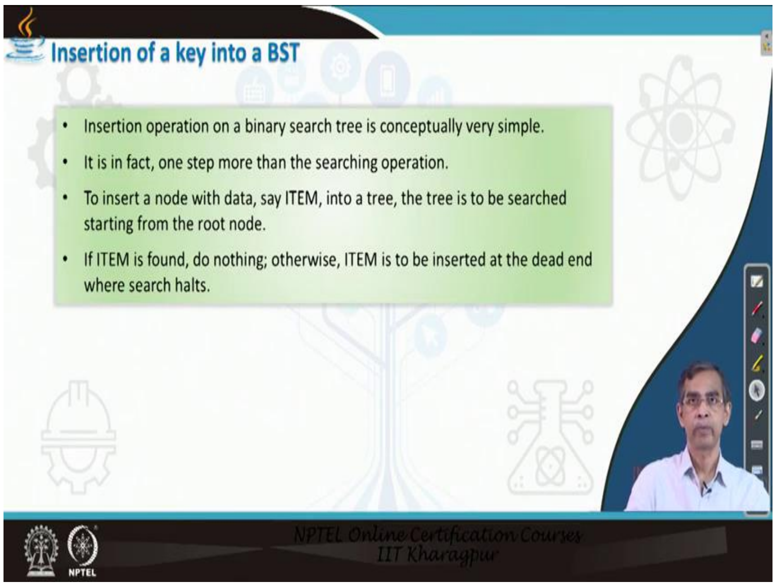


Algorithm: Searching BST

```
Algorithm SearchBST(ROOT, ITEM)
1.  ptr = ROOT, flag = FALSE                              // Start from root
2.  While (ptr ≠ NULL) and (flag = FALSE) do
3.      Case: ITEM < ptr→DATA                             // Go to the left sub-tree
4.          ptr = ptr→LCHILD
5.      Case: ptr→DATA = ITEM                             // Search is successful
6.          flag = TRUE
7.      Case: ITEM > ptr→DATA                             // Go to the right sub-tree
8.          ptr = ptr→RCHILD
9.      EndCase
10. EndWhile
11. If (flag = TRUE) then                                 // Search is successful
12.     Print "ITEM has found at the node", ptr
13. Else
14.     Print "ITEM does not exist: Search is unsuccessful"
15. EndIf
16. Stop
```

So, that is why this is on faster technique. So, for the search operation is concerned. Now so this is a search operation and this is the algorithm that I have given for your implementation, if you want to write the program, you can follow this algorithm. And so next is, next operation, let us consider about insertion, inserting an element into a binary tree, more specifically this called a binary search tree. Now, you can recall we are trying to insert into binary search tree, we could or we may not, depending on where to because ambiguity was there.

(Refer Slide Time: 13:21)

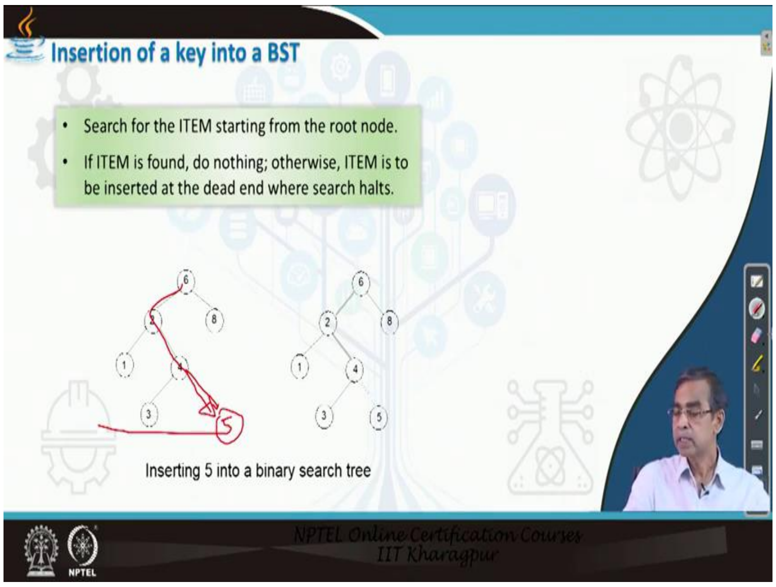


Insertion of a key into a BST

- Insertion operation on a binary search tree is conceptually very simple.
- It is in fact, one step more than the searching operation.
- To insert a node with data, say ITEM, into a tree, the tree is to be searched starting from the root node.
- If ITEM is found, do nothing; otherwise, ITEM is to be inserted at the dead end where search halts.

But here you can see ambiguity is not there. Now, let us see how this binary search tree insertion can takes place here. So, we have to insert if we do not find the element there in the binary search tree, if we find that already the element is available or exist, then we do not have to insert it. Now on the other hand, if it does not find it, then we have to insert at the dead end. Actually dead end mean where your search halts without finding the element there. So, this is called a dead end.

(Refer Slide Time: 13:59)



Insertion of a key into a BST

- Search for the ITEM starting from the root node.
- If ITEM is found, do nothing; otherwise, ITEM is to be inserted at the dead end where search halts.

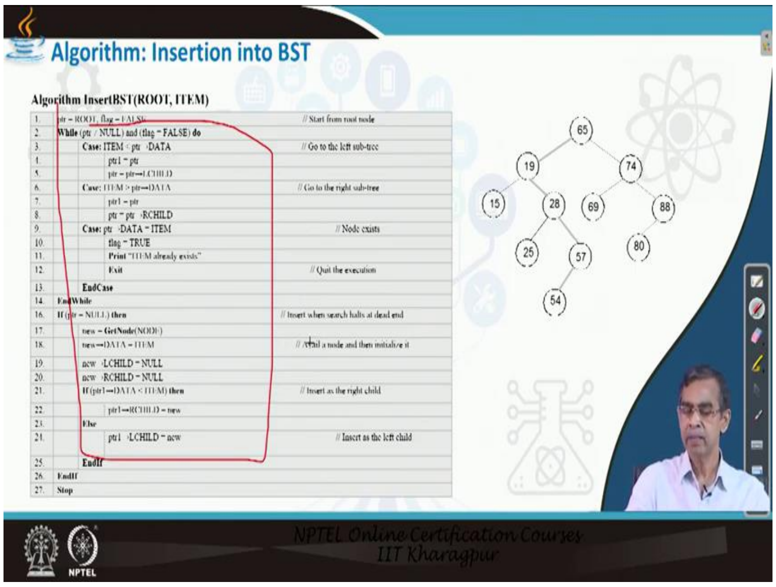
Inserting 5 into a binary search tree

Now let us see how we can reach to the dead end. Here is an example that I can give you. Say suppose this is a input binary search tree and we want to insert the node 5. Now let us search 5 first whether 5 it is there or not? So, search will follow you can see this path actually, this path. But here we can stop. So, this is a dead end. So, that it means we simply okay insert the 5 here.

So, insert the 5 here means so this link fill will store that this of the new node which store 5 for example it is like this. So, this is this is the idea about insertion. So, what you can say insertion in a binary search tree allow unique and it is also very straightforward and again, like binary search tree also maximum number of nodes that needs to be compared in have to find that dead end it is a length of the maximum path, actually.

And like say for example here also length of maximum path is 4 or whatever it is there. So, maximum 4 nodes to be compared in this case, but in this case, only 3 nodes we have to compare whatever it is there. So, this is idea about insertion operation in binary search tree truly both searching and insertion is very straightforward and also very fast.

(Refer Slide Time: 15:23)

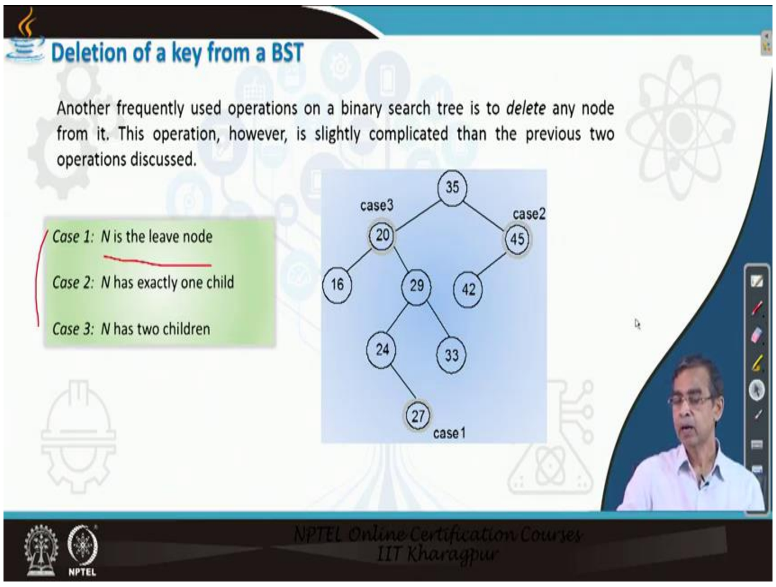


Algorithm: Insertion into BST

```
Algorithm InsertBST(ROOT, ITEM)
1.  ptr = ROOT, flag = FALSE                          // Start from root node
2.  While (ptr ≠ NULL) and (flag = FALSE) do
3.      Case: ITEM < ptr→DATA                          // Go to the left sub-tree
4.          ptr1 = ptr
5.          ptr = ptr→LCHILD
6.      Case: ITEM > ptr→DATA                          // Go to the right sub-tree
7.          ptr1 = ptr
8.          ptr = ptr→RCHILD
9.      Case: ptr→DATA = ITEM                          // Node exists
10.         flag = TRUE
11.         Print "ITEM already exists"
12.         Exit                                       // Quit the execution
13.     EndCase
14. EndWhile
16. If (ptr = NULL) then                               // Insert when search halts at dead end
17.     new = GetNode(NODE)
18.     new→DATA = ITEM                                // Avail a node and then initialize it
19.     new→LCHILD = NULL
20.     new→RCHILD = NULL
21.     If (ptr1→DATA < ITEM) then                     // Insert as the right child
22.         ptr1→RCHILD = new
23.     Else
24.         ptr1→LCHILD = new                          // Insert as the left child
25.     EndIf
26. EndIf
27. Stop
```

Now, let us consider the another operation, okay. This is a algorithm I have written and then that algorithm can be considered both searching as well as inserting, the element that at the dead end like and you can implement it, this algorithm written in pseudo code English like sentence. So, you can easily implement it in any programming language in Java also no exception. So, this is the insertion operation.

(Refer Slide Time: 15:49)



Deletion of a key from a BST

Another frequently used operations on a binary search tree is to *delete* any node from it. This operation, however, is slightly complicated than the previous two operations discussed.

Case 1: *N* is the leave node

Case 2: *N* has exactly one child

Case 3: *N* has two children

Now, let us consider the deletion operation from a binary search tree. Now, how the deletion operation can be taken place, but deletion operation bit okay unlike that to other operation like searching and insertion bit a complex, complicated, because there are three different cases are to be there. And according to the three different cases we have to follow.

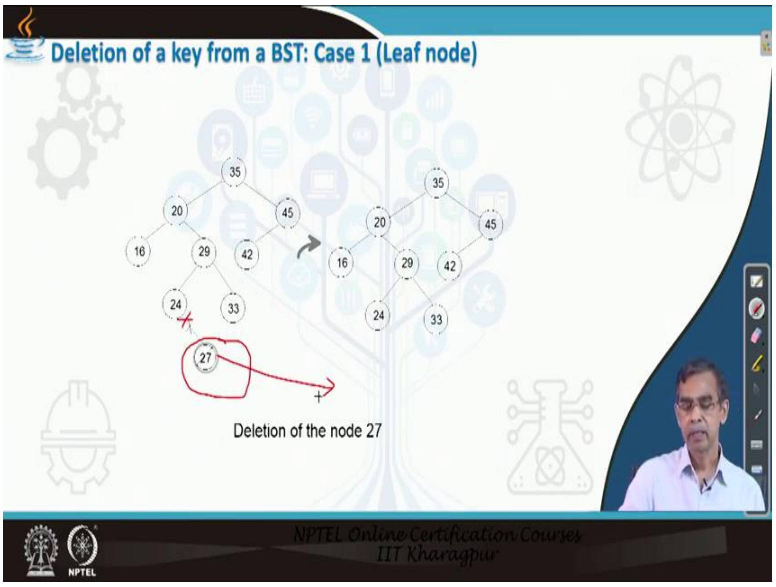
Now so let us consider first case one case one is that now so far, the deletion is concerned again the same. The first step is that we have to search whether element is present or not. If element is not present in the binary search tree, then we do not have to do anything.

So, deletion operation does not have to do there. On the other hand, if we find that node is available, then there are maybe three cases that maybe. So, case one is that the element is available, at leap node. Case two is that element is available in a node which has exactly one child, either left child or right child. And second case that element is available at a node where the node has both the child left child as well as right child.

So, two cases are there. And depending on the two cases, our link management or updation of the links after deleting after removing the element from the tree are to be followed. Now, here, here that three different cases. For example say suppose you want to delete 27.

So, it is case one because searching these 27 found it here and this is a leap node. Now, this is a second case we have to delete 45, 45 has only the left child. So, it does not have any right child. So, we can have the case two and case three as you can see it has both left child as well as right child.

(Refer Slide Time: 17:42)



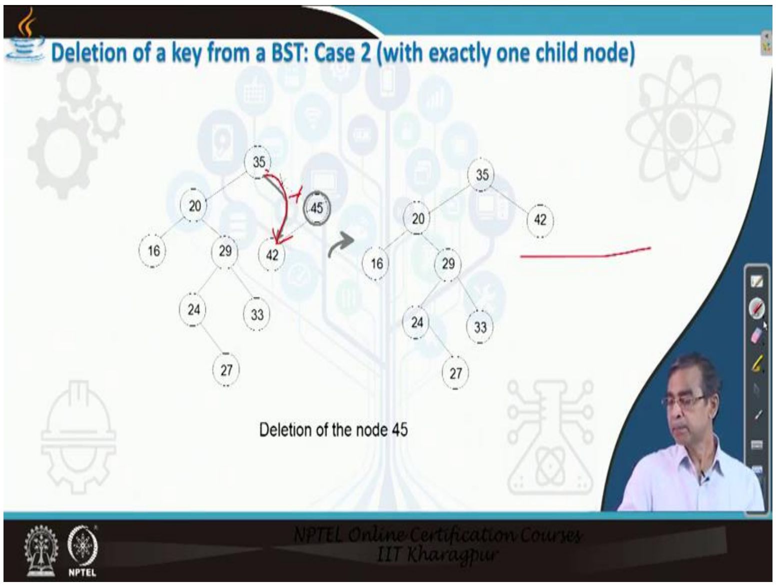
Deletion of a key from a BST: Case 1 (Leaf node)

Now, let us see the operations, how the different operations can be taken place so for the searching I mean, so for the removal or node is concerned. Now, so the first case, the case is on very simple and trivial case.

So, here the last node, leap node is to be removed. Idea is that the last node removal of last node means basically the each parent who pointing this node, the link field of that part will be removed, for example, 24 is the parent of 27, so the link part of this, this basically the 24, the 27 appears as a child of the parent 24 in a say right part.

So, the right link of parent 24 needs to be null and needs to be made null. So, if we made null then this basically node is deleted and this node can be made free or written to the memory bank. So, this is the idea about the first case, the case one, case one really very easy or simple case.

(Refer Slide Time: 18:53)



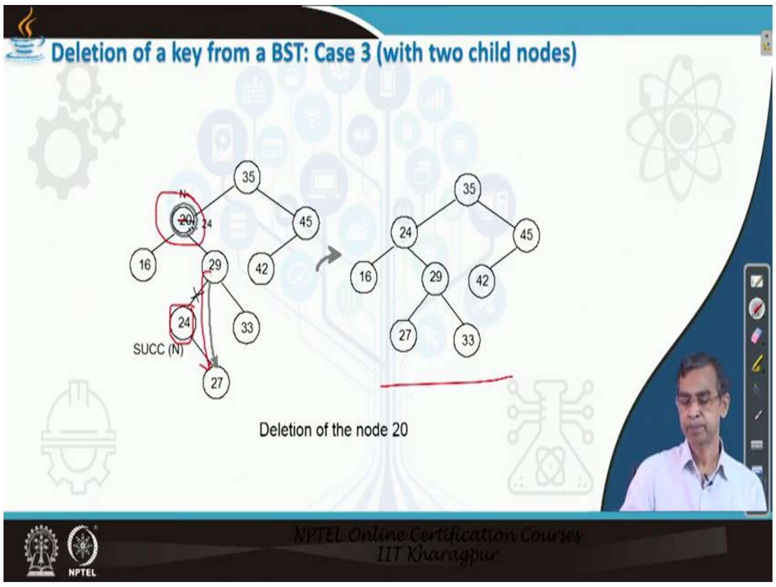
Deletion of a key from a BST: Case 2 (with exactly one child node)

Deletion of the node 45

Now, let us consider the next case, which contains the node, the node that needs to be deleted. It has only one child, for example here 45. So, what we should do is that let us first see who is the parent of 45. So, 35 is the parent. So, what the idea is that the 45 points to node this address will be assigned to the as this for each parent was pointing to the node under deletion. So, this basically needs to be removed. And instead of this, this link will point to the node that this node basically store. So, this basically the idea.

So, this way this is the result and that you can get it. Now, here the link management, I am telling you. And so this is a link management that it needs to be followed in case of case two.

(Refer Slide Time: 19:46)



Deletion of a key from a BST: Case 3 (with two child nodes)

Deletion of the node 20

Now, case three is more complex. This is a possibly the toughest operation so far the binary search tree deletion is concerned. Now, here is an example you have to little bit follow it carefully. Say suppose do you want to remove the node say 24, 24 is to be deleted here, sorry, I am sorry thus two cases suppose. We want to delete the node. 22 be deleted here, now 20 the nodes that needs to be deleted, it has both left child right child.

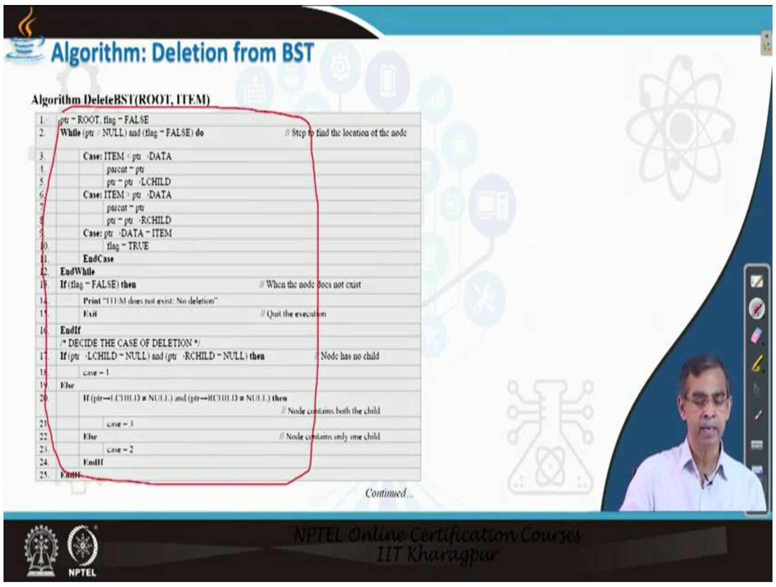
Now, what is the procedure? Procedure is that which is the next node, next smallest node rather, next smallest node of the 28th node with the tree now if you see the next smallest node after 20 in this three is basically 24. So, what basically the idea is that we just remove this node 24, rather, we can say that the node this 20 is to be replaced by the value of the node, which is the minimum value in this.

So, if we do that means that 20 should be replaced by 24 in just simply replacing no link manipulation here. And as the 24 is removed and then 24s parent, namely 29 who is link fill is store for part two to link the child it is to be updated by the child of this node.

So, this basically child of this node is 27 and we have to update the link which basically used to point to the node on the deletion to be many. So, the, there will be so link of this field will with this one. So, this field will be assigned by the address of the child of the node under division, so this is the case. Now, this is the procedure that is that takes place. So, basically now in this particular algorithm, so we have to search we will get it after searching, then we have to find another search.

This search is basically indicating that which is the next lowest value after the node under consideration. So, this is the idea about the deletion operations in three different cases that we have learned about it.

(Refer Slide Time: 22:09)



Algorithm: Deletion from BST

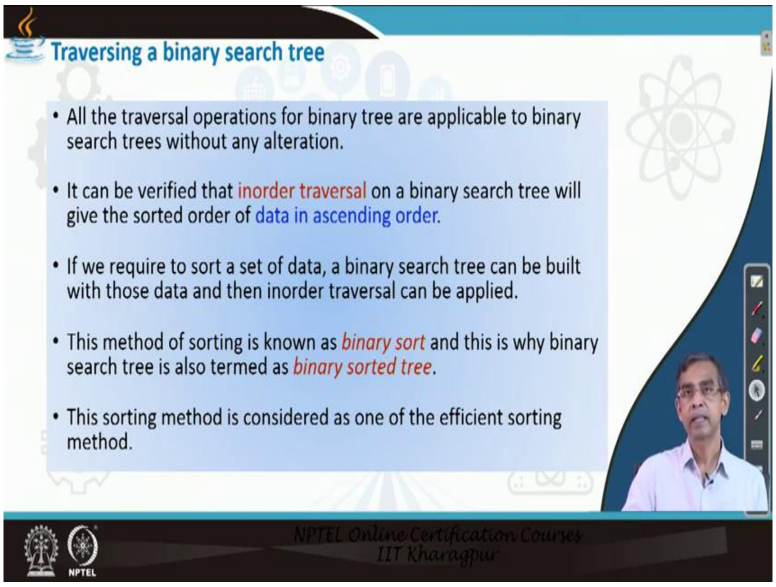
Algorithm DeleteBST(ROOT, ITEM)

```
1.  ptr = ROOT, flag = FALSE
2.  While (ptr ≠ NULL) and (flag = FALSE) do          // Step to find the location of the node
3.          Case: ITEM < ptr→DATA
4.                  parent = ptr
5.                  ptr = ptr→LCHILD
6.          Case: ITEM > ptr→DATA
7.                  parent = ptr
8.                  ptr = ptr→RCHILD
9.          Case: ptr→DATA = ITEM
10.                 flag = TRUE
11.         EndCase
12. EndWhile
13. If (flag = FALSE) then                             // When the node does not exist
14.         Print "ITEM does not exist: No deletion"
15.         Exit                                       // Quit the execution
16. EndIf
    /* DECIDE THE CASE OF DELETION */
17. If (ptr→LCHILD = NULL) and (ptr→RCHILD = NULL) then      // Node has no child
18.         case = 1
19. Else
20.         If (ptr→LCHILD ≠ NULL) and (ptr→RCHILD ≠ NULL) then
                                                       // Node contains both the child
21.                 case = 3
22.         Else                                       // Node contains only one child
23.                 case = 2
24.         EndIf
25. EndIf
```

Continued...

And this is the algorithm that is written very precisely and algorithm learning this algorithm, or understanding it is matter of time I will suggest you to go through step by step, then you will be able to follow. So, this a little step that okay if you follow, then you can write a program implementing the deletion, whatever the case that we have mention here all three cases are basically covered here in this particular example.

(Refer Slide Time: 22:44)



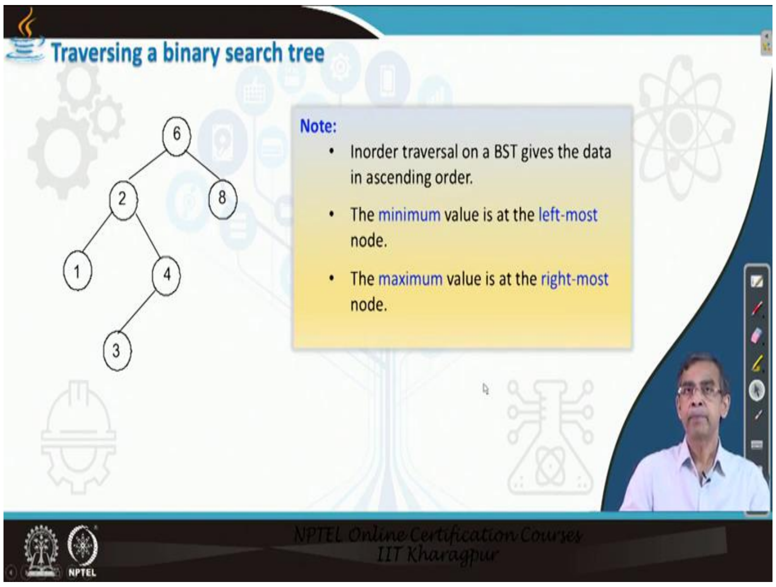
Traversing a binary search tree

- All the traversal operations for binary tree are applicable to binary search trees without any alteration.
- It can be verified that inorder traversal on a binary search tree will give the sorted order of data in ascending order.
- If we require to sort a set of data, a binary search tree can be built with those data and then inorder traversal can be applied.
- This method of sorting is known as *binary sort* and this is why binary search tree is also termed as *binary sorted tree*.
- This sorting method is considered as one of the efficient sorting method.

NPTEL Online Certification Courses
IIT Kharagpur

So, this is about the different operations and then finally, I will quickly cover about the traversal operation, the traversal operations, the traversal operations again, that three different ways that we have learned about in order or post order traversal, they can be applied there.

(Refer Slide Time: 23:05)



Traversing a binary search tree

Note:

- Inorder traversal on a BST gives the data in ascending order.
- The minimum value is at the left-most node.
- The maximum value is at the right-most node.

NPTEL Online Certification Courses
IIT Kharagpur

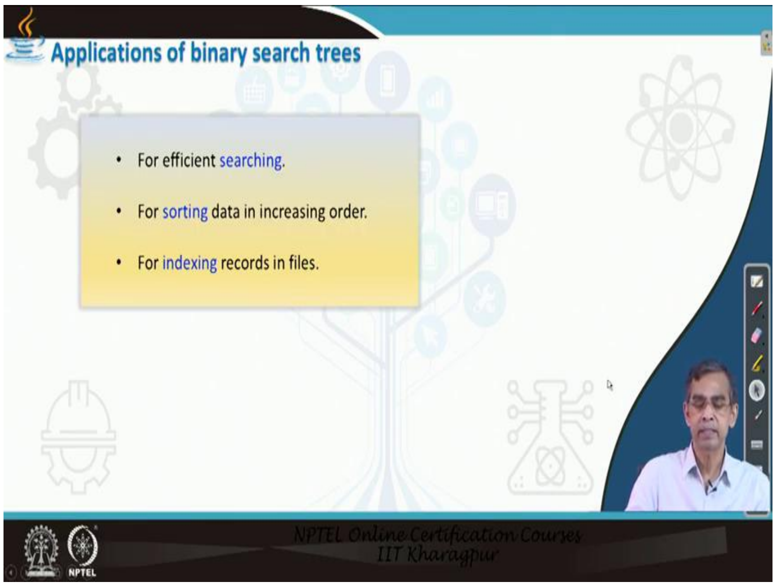
Now here, okay. So, so application I mean in order traversals is a same as binary tree that we have discussed so if you can apply in order traversal so you can find one, we have ordering the elements. And if we follow pre order another we have ordering post order like this one.

And one important observation here that if we apply in order traversals, then the binary search tree after the successful after the traversals it will give the output of all the elements in the ascending order of the numbers. And in that sense, actually binary search tree, we can suppose in other applications suppose you have to find which is the lowest value, which is the highest value.

So, some min value and max value that also can be calculated very easily. So, if you go left, left, left, then at the end where you can stop, it is basically the min value. Then starting from the root. If we go to the right, right always right and then if we finish somewhere we do not find it there, then the last elements, where we stop it is called the right value.

So, this is basically the minimum value is the leftmost node and the maximum value that is stored there in a binary search tree is called the right most node whatever it is there. Now, this is also a matter of traversal actually, that you can follow a little bit customizing the traversal you can have it.

(Refer Slide Time: 24:33)



Applications of binary search trees

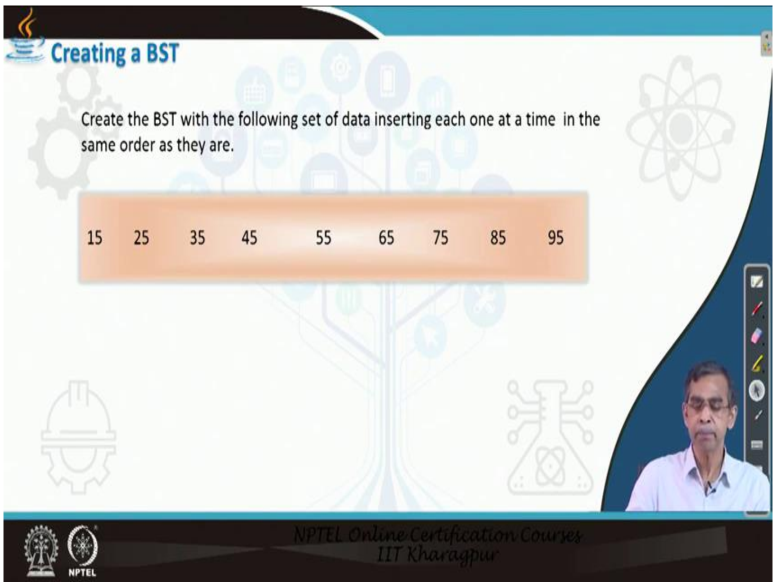
- For efficient searching.
- For sorting data in increasing order.
- For indexing records in files.

NPTEL Online Certification Courses
IIT Kharagpur

So, these are the traversals algorithm we have studied and so far the application of binary search tree is concerned searching is a good way of use it and another sorting and indexing of records I mean, that is basically searching another. So, that means I told you that all the elements is basically stored in a noncontagious location.

Now, here elements means a record and each record will be identified by a field it is called the primary field or primary key. So, a binary search tree will store the primary key values as a node and then a pointer from the primary key value can point exactly to the memory location where the record is stored. So, this way we can have a very fastest indexing mechanism to search a file where a file stores number of records.

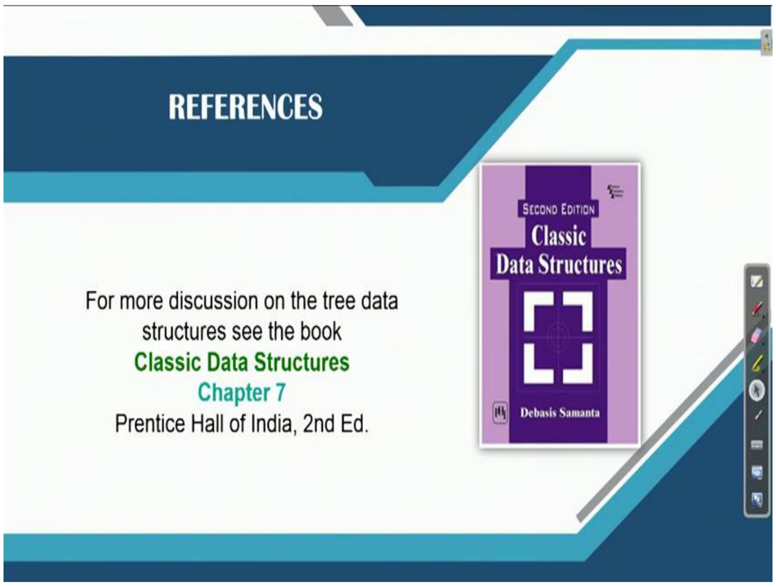
(Refer Slide Time: 25:18)



Creating a BST

Create the BST with the following set of data inserting each one at a time in the same order as they are.

15 25 35 45 55 65 75 85 95



REFERENCES

For more discussion on the tree data structures see the book
Classic Data Structures
Chapter 7
Prentice Hall of India, 2nd Ed.

So, these are the few applications that we can consider and for details about the applications and then different what is called the algorithms and application there are many more applications also of this binary discuss in detail in this book in chapter 7. So, you are highly encourage to go through the book and study much more. Thank you very much.