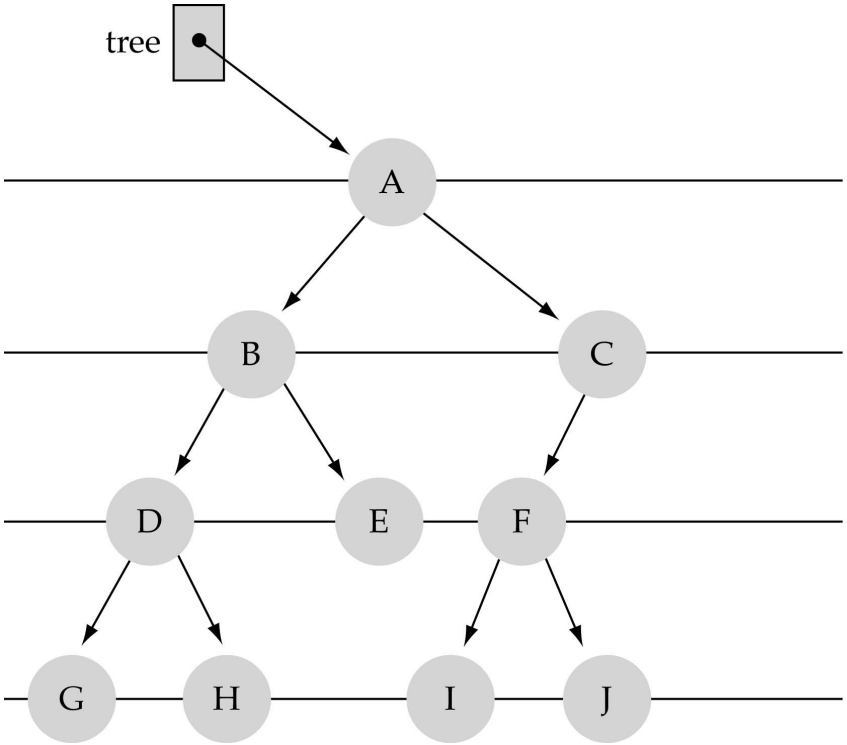
Binary Search Tree

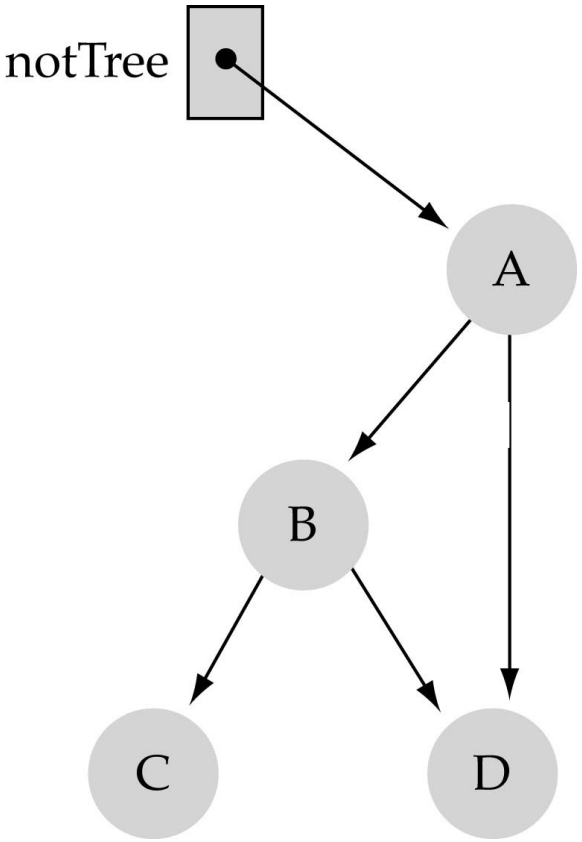
What is a Binary Tree?

- Property 1: Each node can have up to two successor nodes.

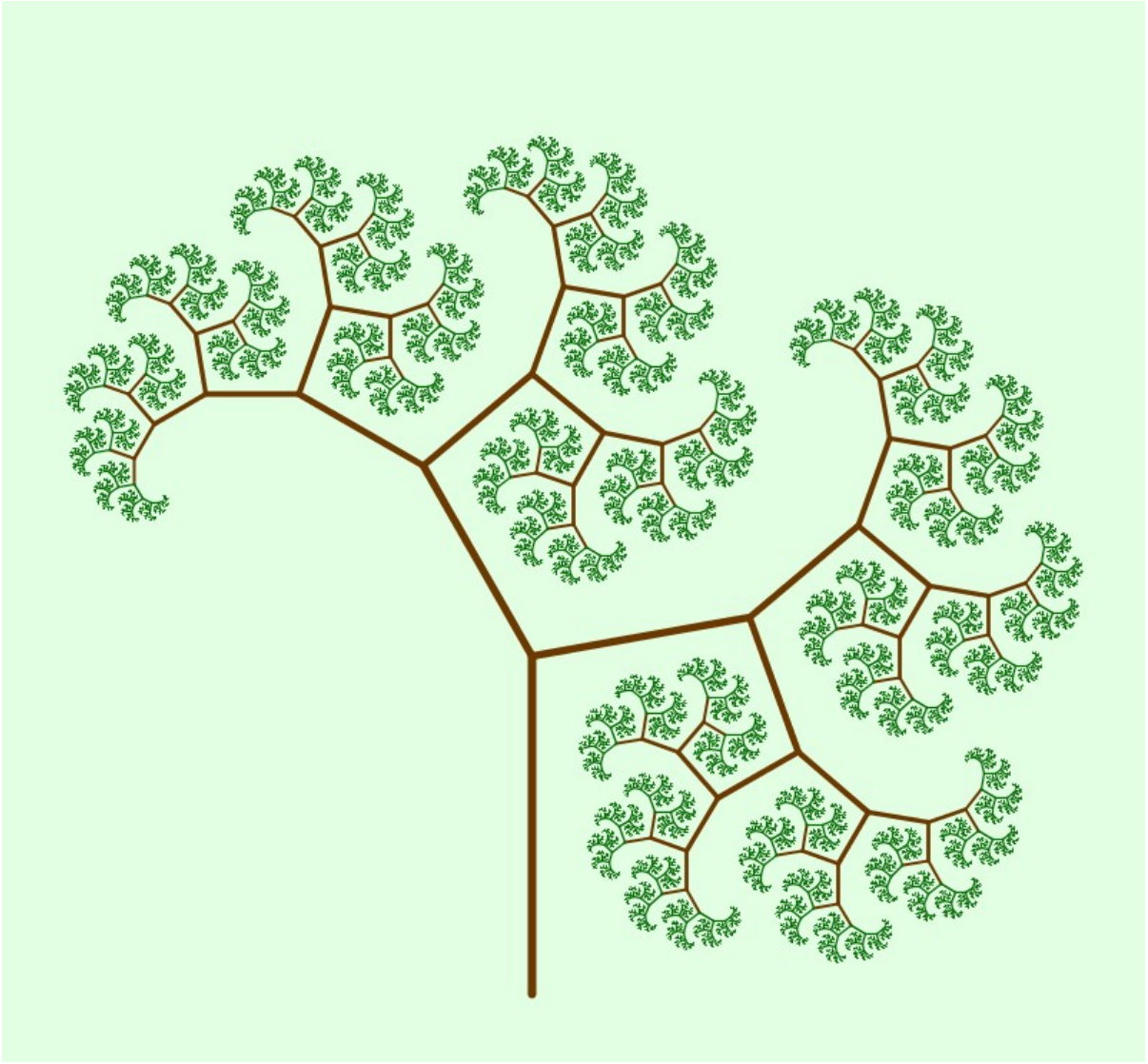


What is a Binary Tree?

- Property 2: a unique path exists from the root to every other node



A Binary Tree

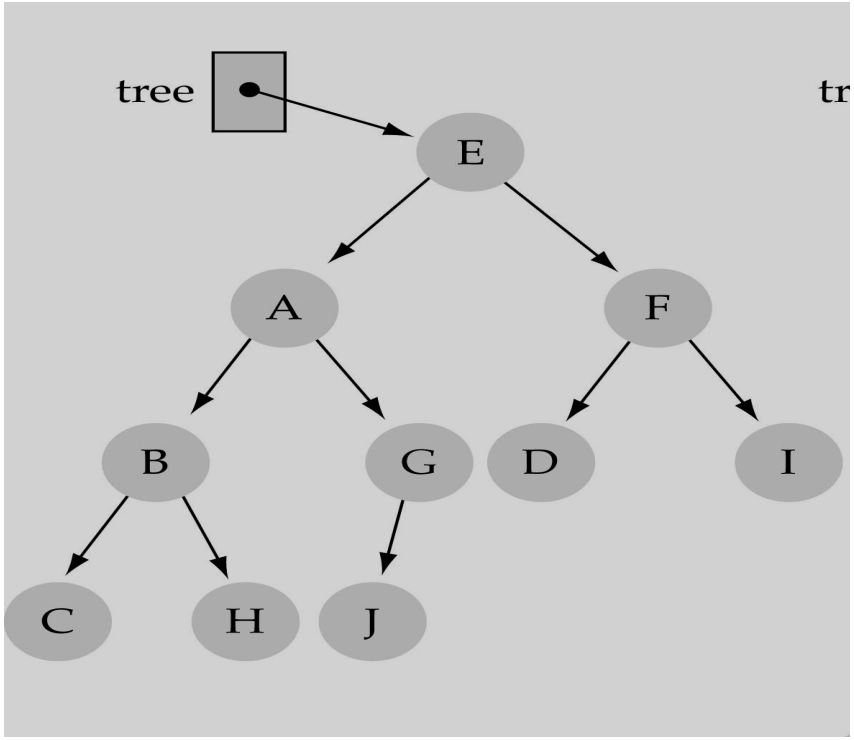


Binary Tree

```
typedef struct tnode *ptnode;
  typedef struct node {
                  short int  key;
                  ptnode right, left;
    } ;
```

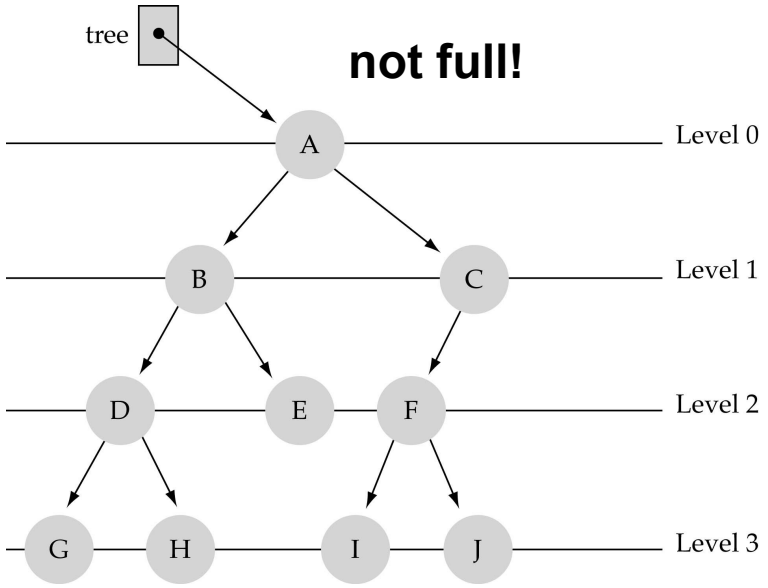
Basic Terminology

- The successor nodes of a node are called its children
- The predecessor node of a node is called its parent
- The "beginning" node is called the root (has no parent)
- A node without children is called a leaf



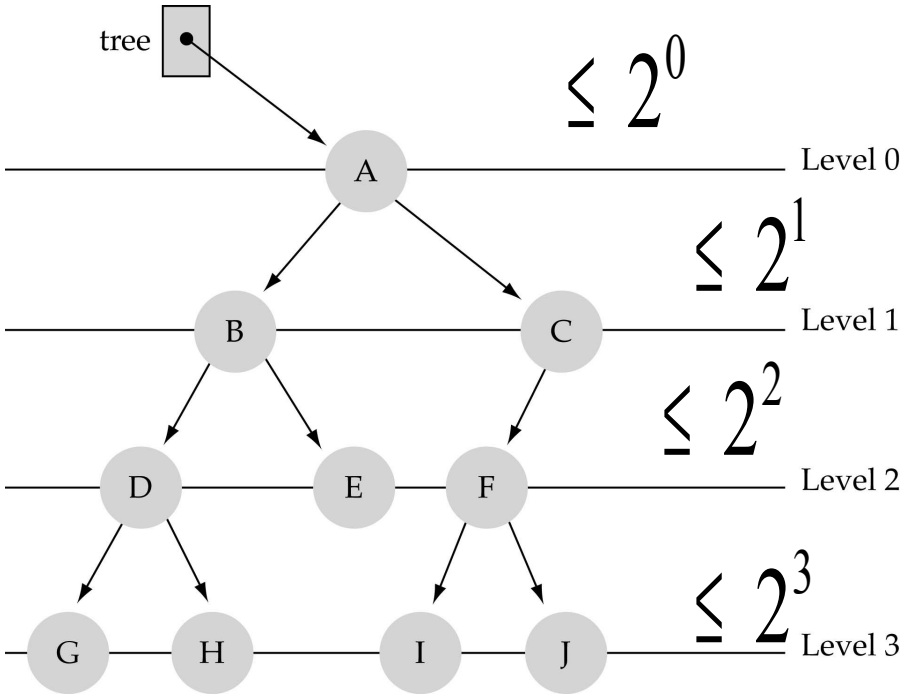
Basic Terminology

- Nodes are organize in levels (indexed from 0).
- **Level (or depth) of a node**: number of edges in the path from the root to that node.
- **Height of a tree h**: #levels = L
  (some books define h as number of levels-1).
- **Full tree:** every node has exactly two children ***and*** all the leaves are on the same level.



What is the max number of nodes at any level l ?

The maximum number of nodes at any level *l* is less than or equal to 2^l where *l*=0, 1, 2, 3,..... L-1



What is the total number of nodes N of a full tree with height h?

$$N = \underset{l=0}{2^0} + \underset{l=1}{2^1} + \ldots + \underset{l=h-1}{2^{h-1}} = 2^h - 1$$

Derived according to the geometric series:

$$x^0 + x^1 + \ldots + x^{n-1} = \sum_{i=0}^{n-1} x^i = \frac{x^n - 1}{x - 1}$$

What is the height h of a full tree with N nodes?

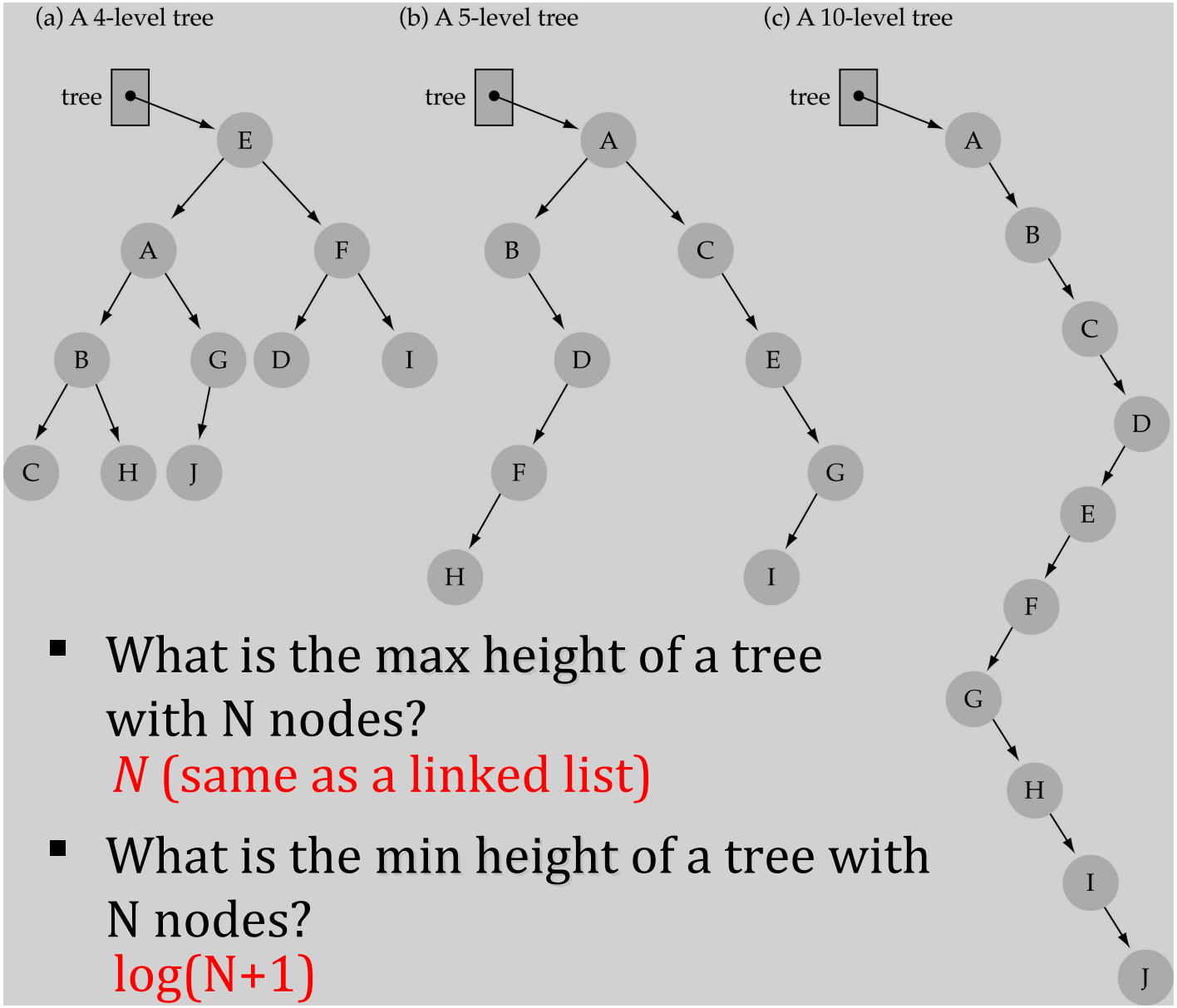
$$2^h - 1 = N$$

$$\Rightarrow 2^h = N + 1$$

$$\Rightarrow h = \log(N+1) \rightarrow O(\log N)$$

Why is h important?

- The Tree operations like insert, delete, retrieve etc. are typically expressed in terms of the height of the tree **h**.
- So, it can be stated that the tree height **h** determines running time!



- What is the max height of a tree with N nodes?
  N (same as a linked list)
- What is the min height of a tree with N nodes?
  log(N+1)

Binary Search

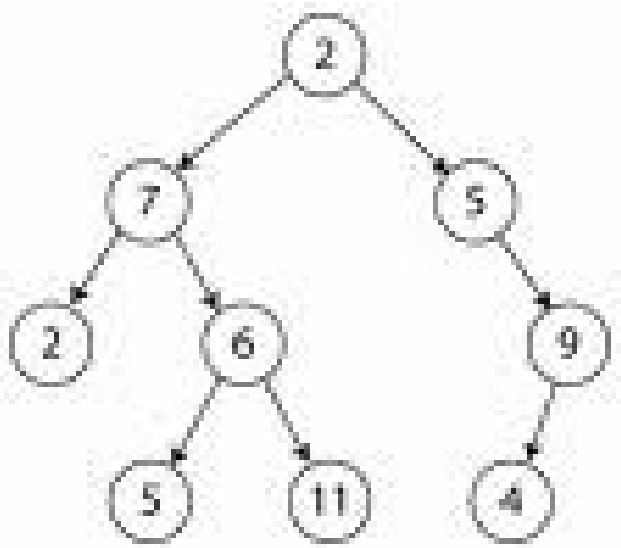
- Suppose DATA is an array which is sorted in increasing numerical order and we want to find the location LOC of a given ITEM in DATA. Then there is an extremely efficient searching algorithm called **Binary Search**.

Binary Search

- Consider that you want to find the location of some word in a dictionary.
- I guess you are not fool enough to search it in a linear way or in other words apply linear search. That is no one search page by page from the start to end of the book.
- Rather, I guess you will open or divide the dictionary in the middle to determine which half contains the word.
- Then consider new probable half and open or divide that half in the middle to determine which half contains the word. This process goes on.
- Eventually you will find the location of the word and thus you are reducing the number of possible locations for the word in the dictionary.

How to search a binary tree?

1. Start at the root
2. Search the tree level by level, until you find the element yo are searching fo

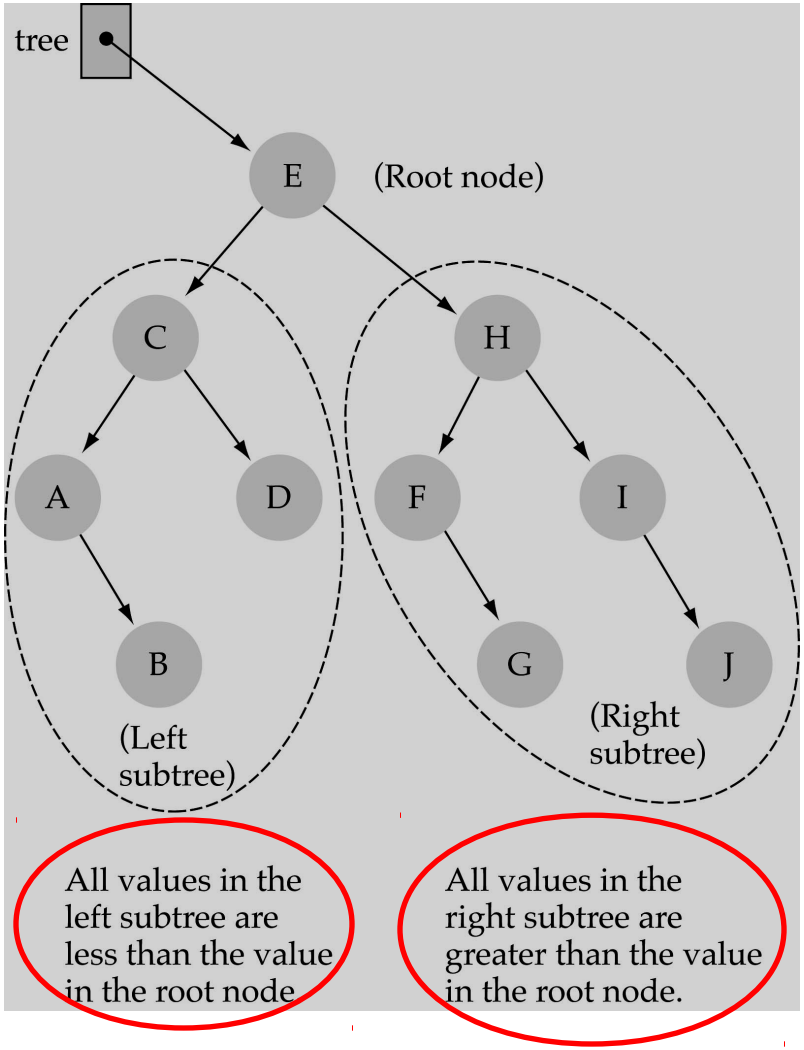


Is this better than searching a linked list?

No → → O(N)

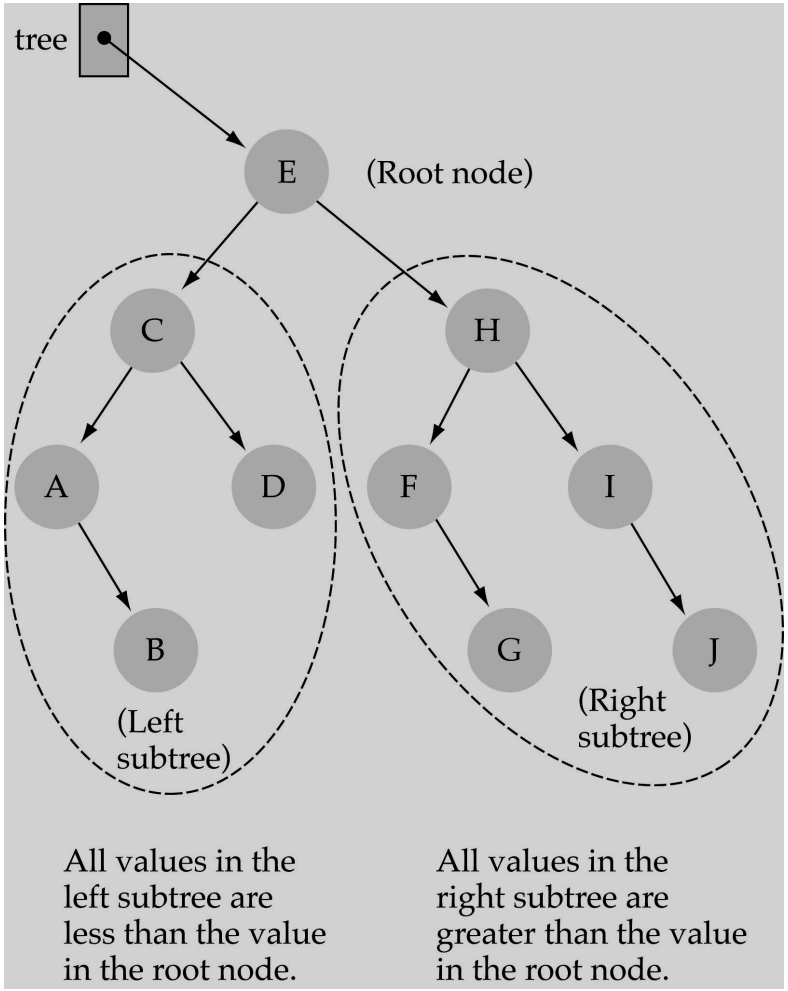
Binary Search Trees

- **Binary Search Tree Property**:
  The value stored at a node is *greater* than the value stored at its left child and *less* than the value stored at its right child



Binary Search Trees

- In a BST, the value stored at the root of a subtree is *greater* than any value in its left subtree and *less* than any value in its right subtree!



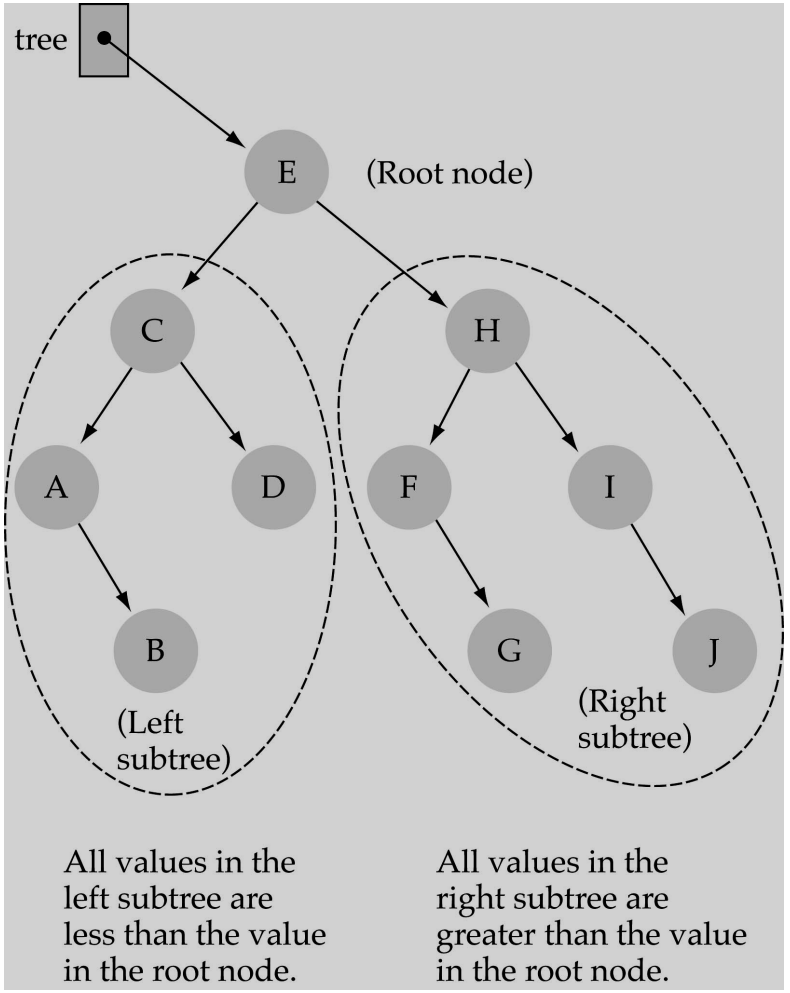
Binary Search Trees

Where is the smallest element?

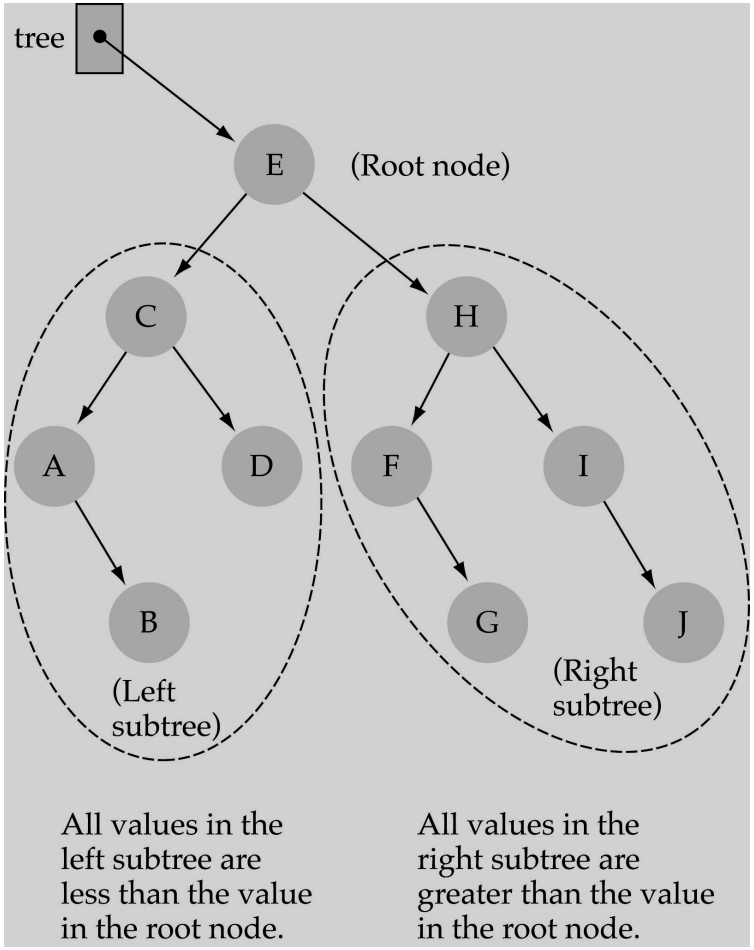
Ans: leftmost element

Where is the largest element?

Ans: rightmost element

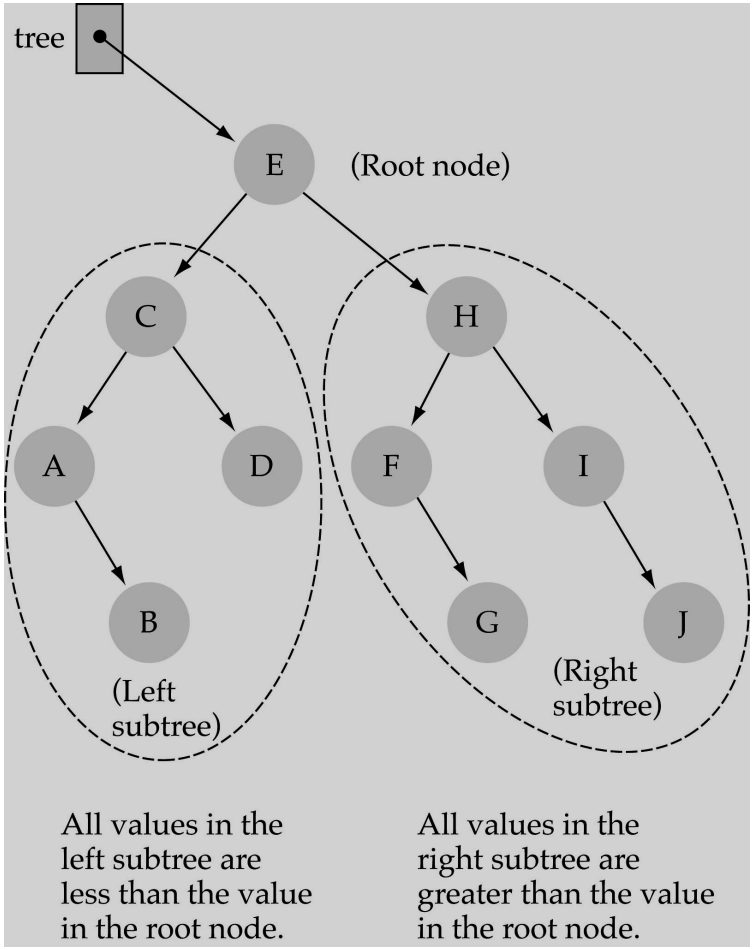


How to search a binary search tree?



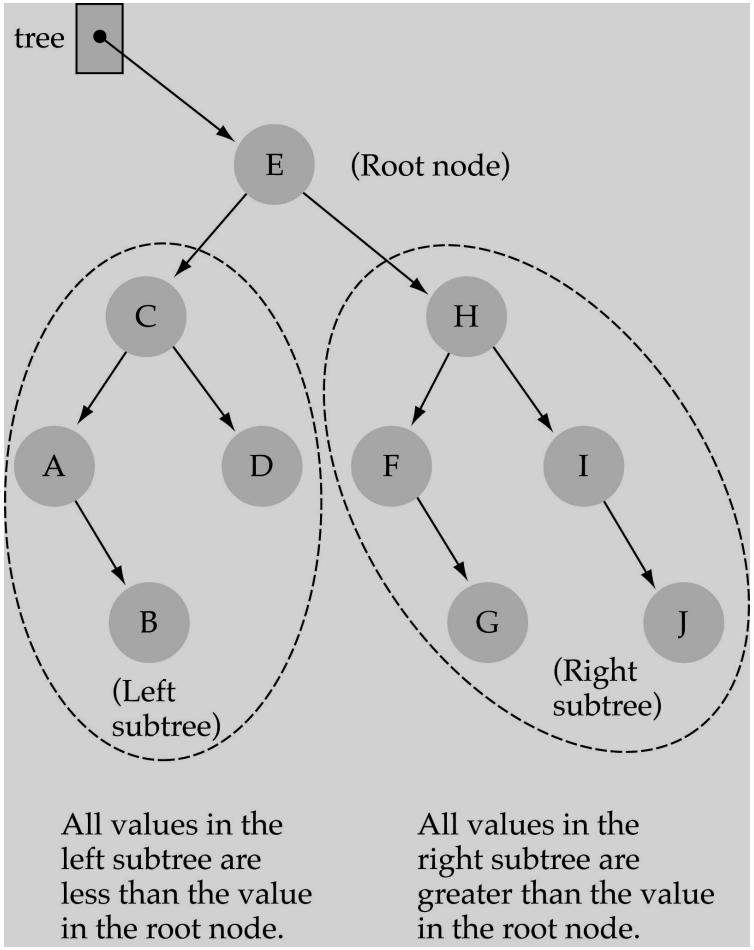
1. Start at the root
2. Compare the value of the item you are searching for with the value stored at the root
3. If the values are equal, then *item found*; otherwise, if it is a leaf node, then *not found*

How to search a binary search tree?



4. If it is less than the value stored at the root, then search the left subtree
5. If it is greater than the value stored at the root, then search the right subtree
6. Repeat steps 2-6 for the root of the subtree chosen in the previous step 4 or 5

How to search a binary search tree?



Is this better than searching a linked list?

Yes !! → → O(logN)

Difference between BT and BST

- A binary tree is simply a tree in which each node can have at most two children.
- A binary search tree is a binary tree in which the nodes are assigned values, with the following restrictions :

1. No duplicate values.
2. The left subtree of a node can only have values less than the node
3. The right subtree of a node can only have values greater than the node and recursively defined
4. The left subtree of a node is a binary search tree.
5. The right subtree of a node is a binary search tree.

Binary Tree Search Algorithm

- Let x be a node in a binary search tree and k is the value, we are supposed to search.
- Then according to the binary search tree property we know that: if y is a node in the left subtree of x, then y.key <= x.key. If y is a node in the right subtree of x, then y.key>=x.key. (x.key denotes the value at node x)
- To search the location of given data k, the binary search tree algorithm begins its search at the root and traces the path downward in the tree.
- For each node x it compares the value k with x.key. If the values are equal then the search terminates and x is the desired node.

Binary Tree Search Algorithm

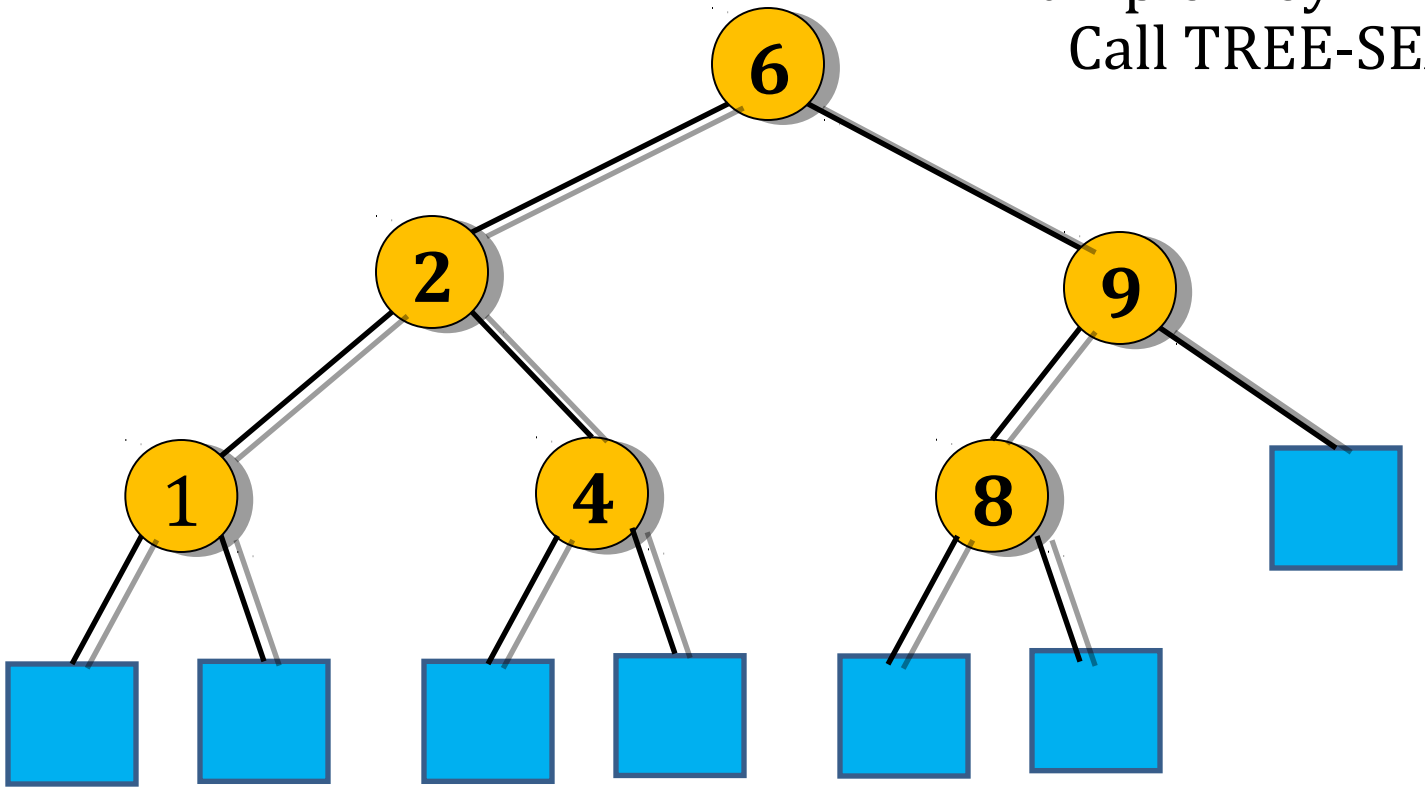
- If k is smaller than x.key, then the search continues in the left subtree of x, since the binary search tree property implies that k could not be in the right subtree.
- Symmetrically, if k is larger than x.key, then the search continues in the right subtree.
- The nodes encountered during the recursion form a simple path downward from the root of the tree, and thus the running time of TREE-SEARCH is O(h), where h is the height of the tree.

Binary Tree Search Algorithm

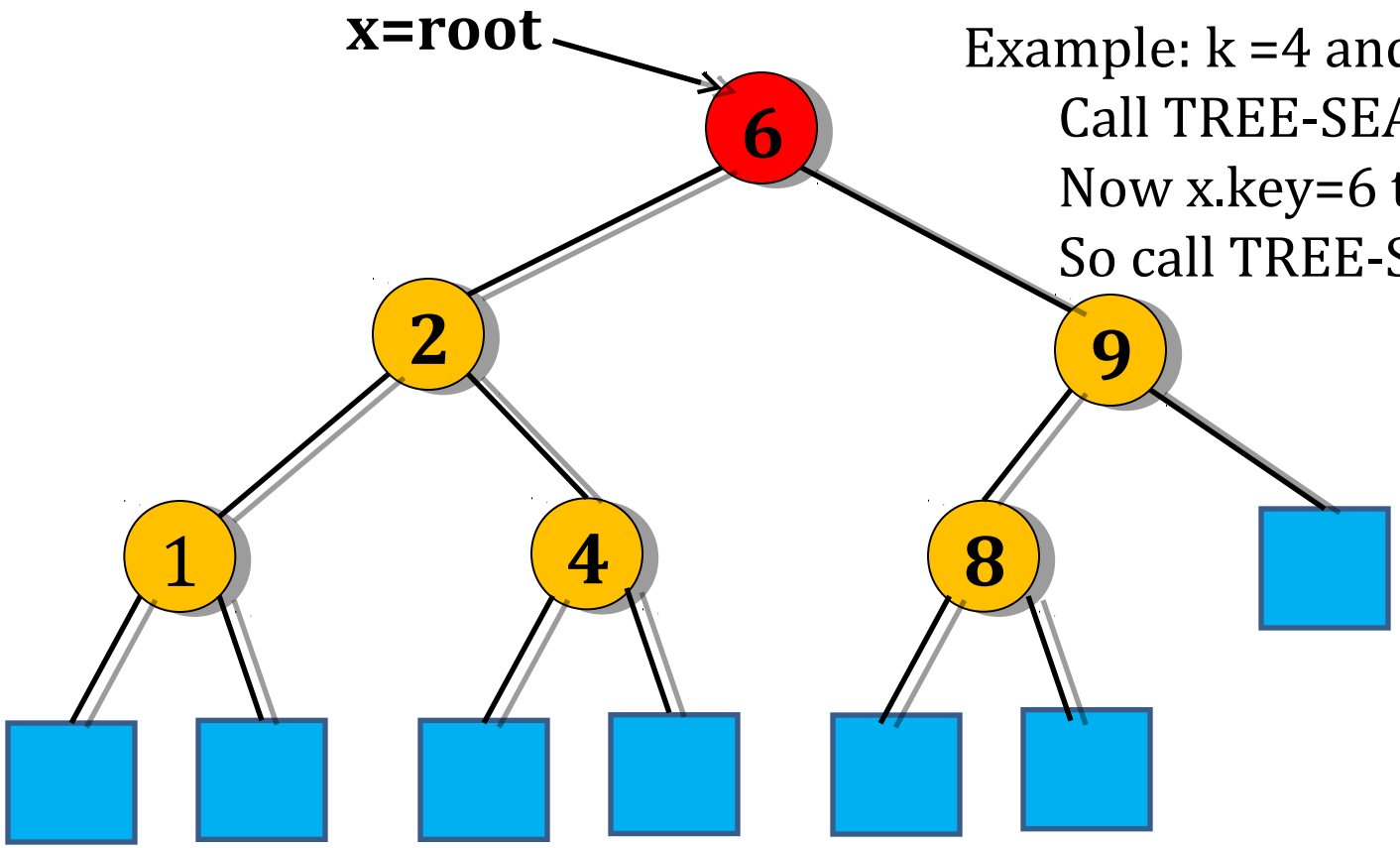
```
TREE-SEARCH(x,k)
1.If x==NIL or k==x.key
2.          return x
3.If k < x.key
4.          return TREE-SEARCH(x.left,k)
5.else return TREE-SEARCH(x.right,k)
```

Binary Tree Search Algorithm

Example: key =4 then find(4)
Call TREE-SEARCH(x,k)



Binary Tree Search Algorithm



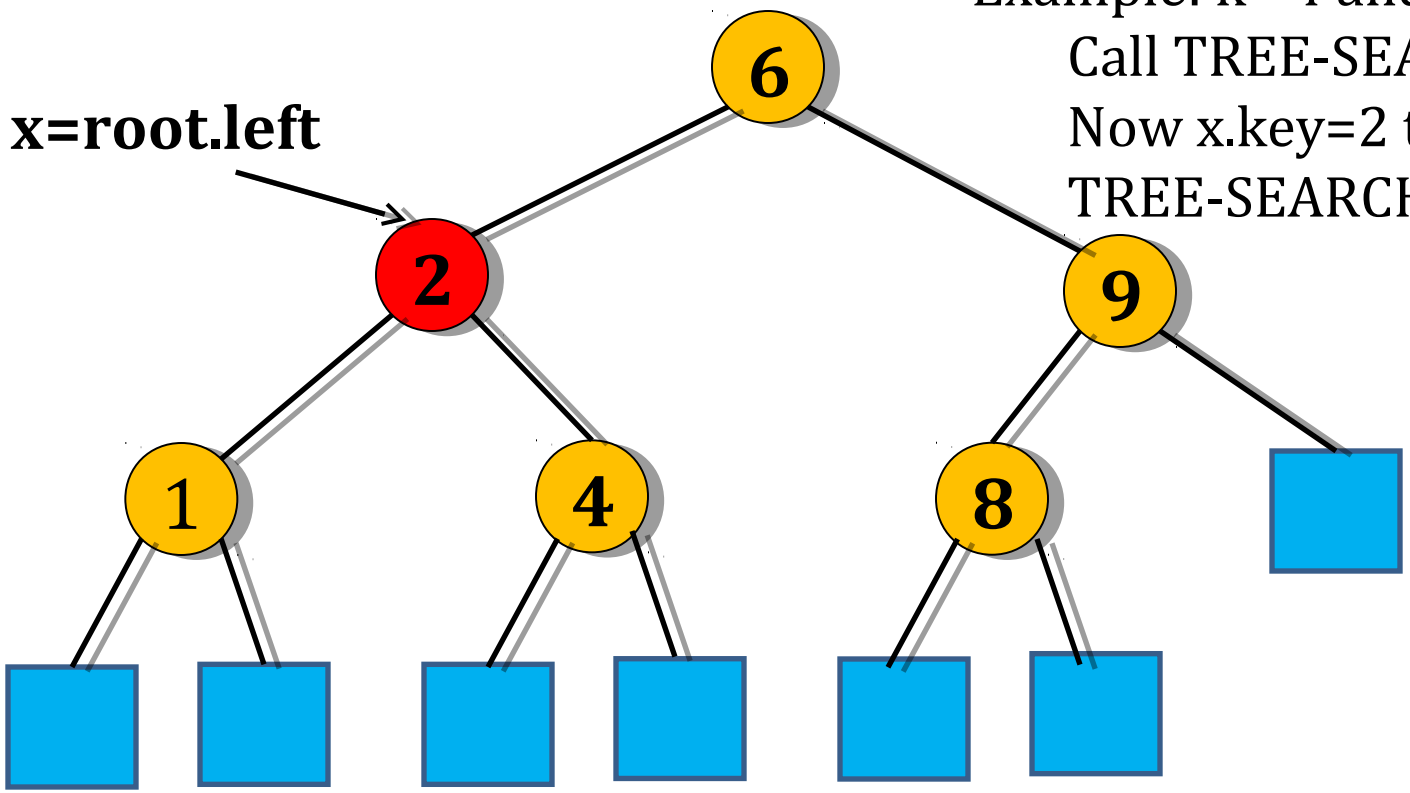
Example: k =4 and x=root

Call TREE-SEARCH(root,4)

Now x.key=6 then k<x.key

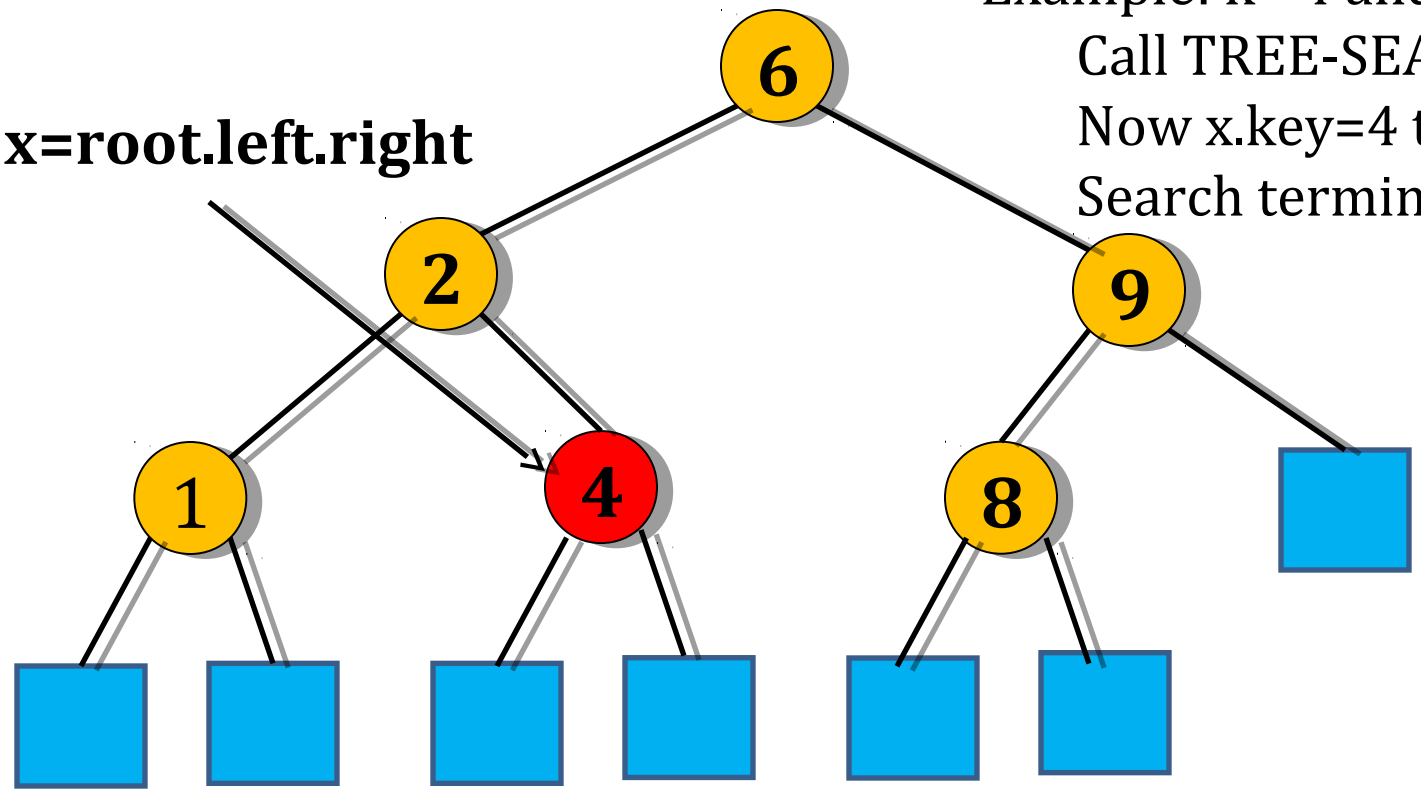
So call TREE-SEARCH(x.left,k)

Binary Tree Search Algorithm



Example: k =4 and x=root.left
Call TREE-SEARCH(x,4)
Now x.key=2 then k>x.key
TREE-SEARCH(x.right,k)

Binary Tree Search Algorithm



Example: k =4 and x=root.left.right

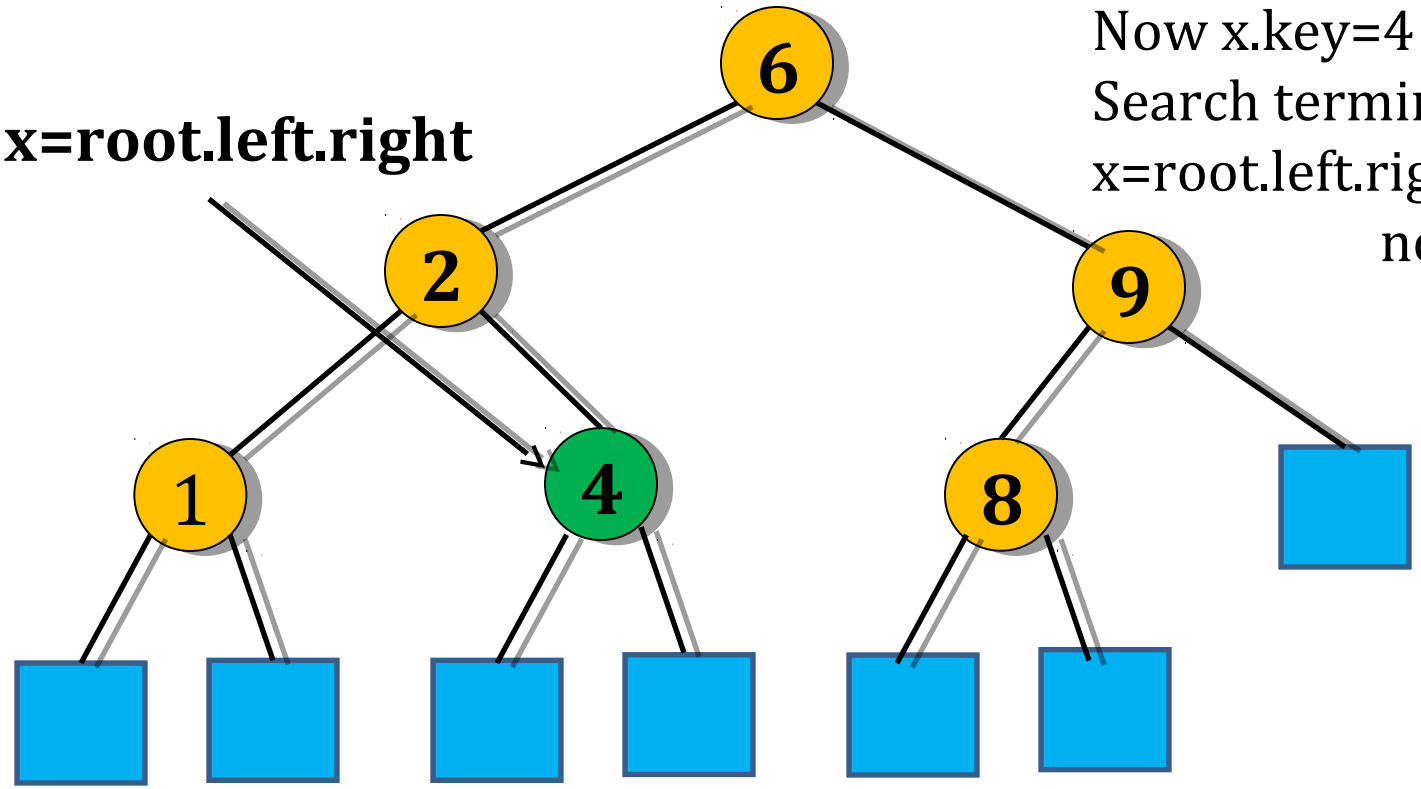
Call TREE-SEARCH(x,4)

Now x.key=4 then k=x.key

Search terminates

Binary Tree Search Algorithm

Example: k =4 and x=root.left.right
Now x.key=4 then k=x.key
Search terminates and
x=root.left.right is the desired
node or location



Animation of How Works in a Sorted Data Array

BST - Pseudo code

```
if the tree is empty
        return NULL

else if  the key value in the node(root) equals the target
        return the node value

else if  the key value in the node is greater than the target
        return the result of searching the left subtree

else if  the key value in the node is smaller than the target
        return the result of searching the right subtree
```

Search in a BST: C code

```
Ptnode search(ptnode root,
              int key)
{
/* return a pointer to the node that
   contains key. If there is no such
   node, return NULL */

  if (!root) return NULL;
  if (key == root->key) return root;
  if (key < root->key)
      return search(root->left,key);
  return search(root->right,key);
}
```

Minimum Key or Element

- We can always find an element in a binary search tree whose key is minimum by following the left children from the root until we encounter a NIL.
- Otherwise if a node x has no left subtree then the value x.key contained in root x is the minimum key or element. The procedure for finding the minimum key:

- TREE-MINIMUM(x)

```
1. while x.left ≠ NIL
2.         x = x.left
3. return x
```

Maximum Key or Element

- We can always find an element in a binary search tree whose key is maximum by following the right children from the root until we encounter a NIL.
- Otherwise if a node x has no right subtree then the value x.key contained in root x is the maximum key or element. The procedure for finding the maximum key:

- TREE-MAXIMUM(x)

```
1. while x.right ≠ NIL
2.         x = x.right
3. return x
```

Insert a value into the BST

- To insert a new value v into a binary search tree T, we use the procedure TREE-INSERT.
- The procedure takes a node z for which z.key=v , z.left=NIL and z.right=NIL.
- It modifies T and some of the attributes of z in such a way that it inserts z into an appropriate position in the tree

Insert a value into the BST

- Suppose v is the value we want to insert and z is the node (New or NIL) we are supposed to find to insert the value v.
  z.p denotes the parent of z.
- X is a pointer that traces a simple path downward the tree and y is the trailing pointer as the parent of x. T.root denote the root of the tree.
- Now the intention is to find a new or NIL node that will satisfy the BST property after placing the value v. The procedure first consider x as the root of the tree thus the parent of the root y=NIL.
- In steps 3 to 7 the procedure causes the two pointer y and x to move down the tree, going left or right depending on the comparison of z.key with x.key, until x becomes NIL.

Insert a value into the BST

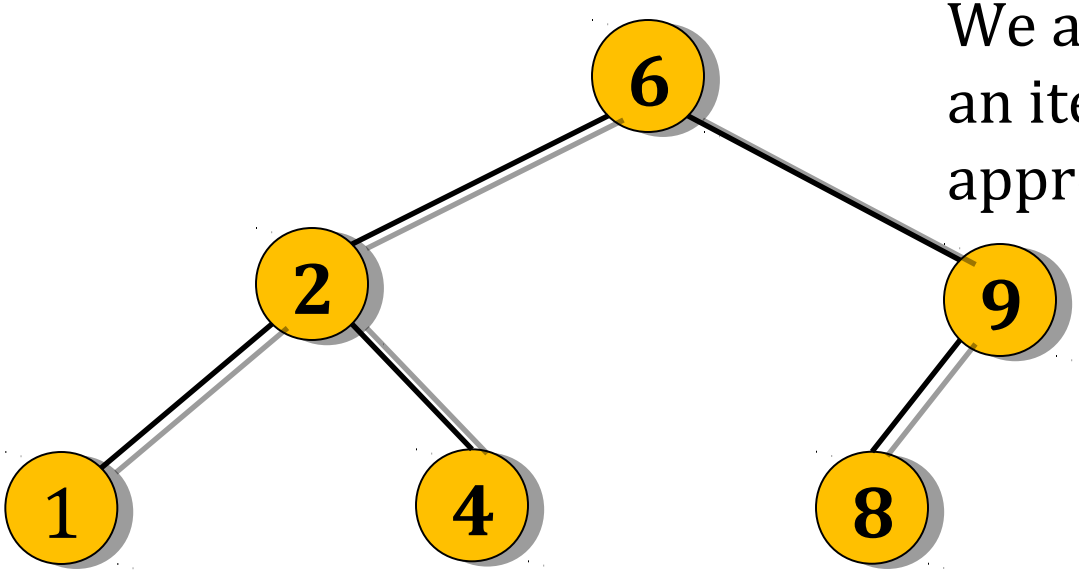
- Now this NIL occupies the position z, where we wish to place the input item.
- At this time we need y the parent of the desired node. This is why at step four we always storing the parent of current node x while moving downward. At the end of step 7 (in step 8) we make this node the parent of z (z.p).
- From steps 9 to 11:
- Now if tree is empty (y==NIL) then c*reate a root* node with the new key(T.root=z)
- If the value v is less than the value of the parent(z.key<y.key) then make it as the left-child of the parent(y.left=z)
- If the value v is greater than the value of the parent(z.key>y.key) then make it as the right-child of the parent(y.right=z)

Insert a value into the BST

- TREE-INSERT(T , z)

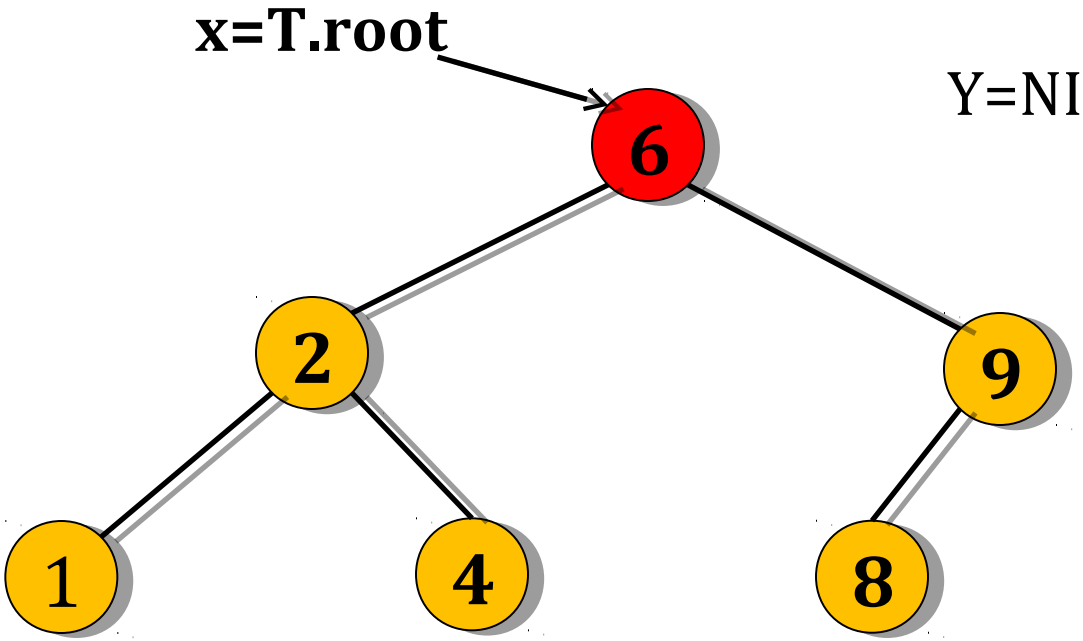
```
1.  y=NIL
2.  x= T.root
3.  While x ≠ NIL
4.          y=x
5.          if z.key<x.key
6.             x=x.left
7.          else x=x.right
8.  z.p=y
9.  if y== NIL
10.      T.root = z
11.elseif  z.key < y.key
12.         y.left=z
13.else y.right = z
```

BST Insertion Algorithm



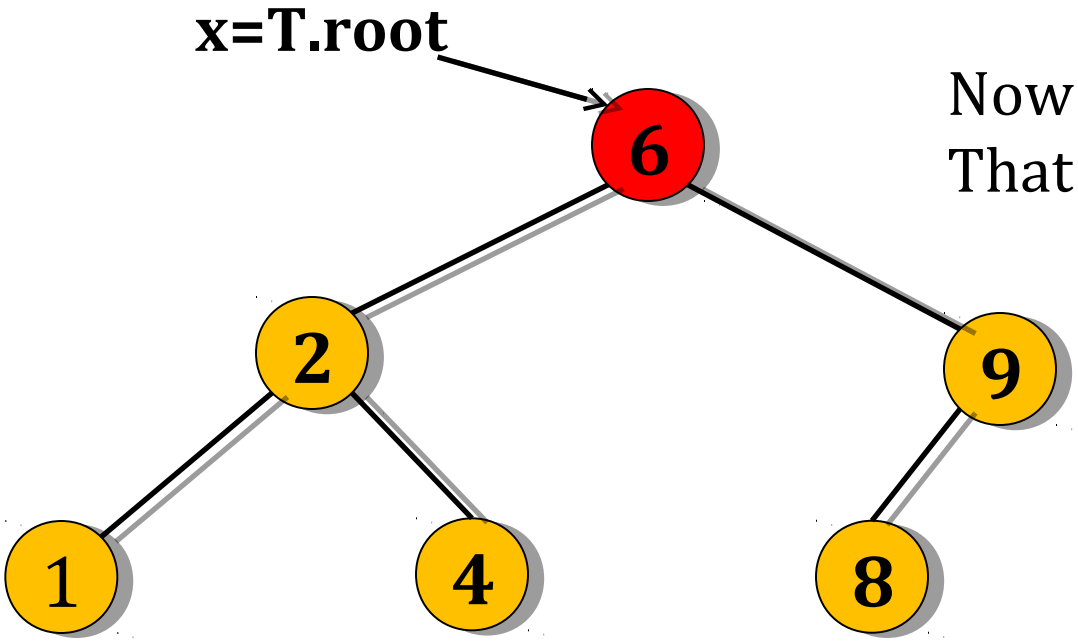
We are supposed to insert an item value 5 and find an appropriate node z for it

BST Insertion Algorithm



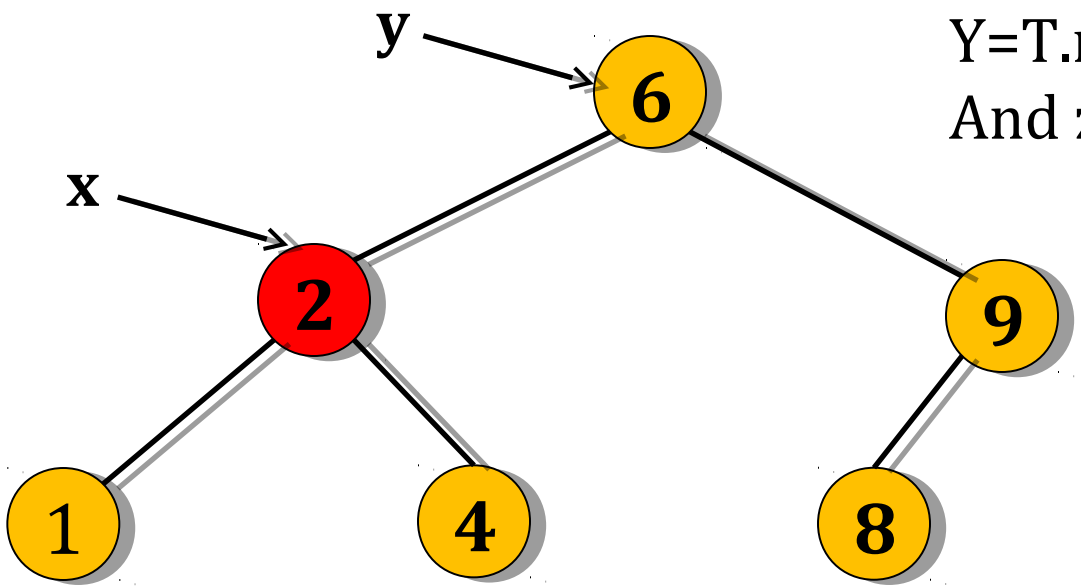
Y=NIL and x=T.root

BST Insertion Algorithm



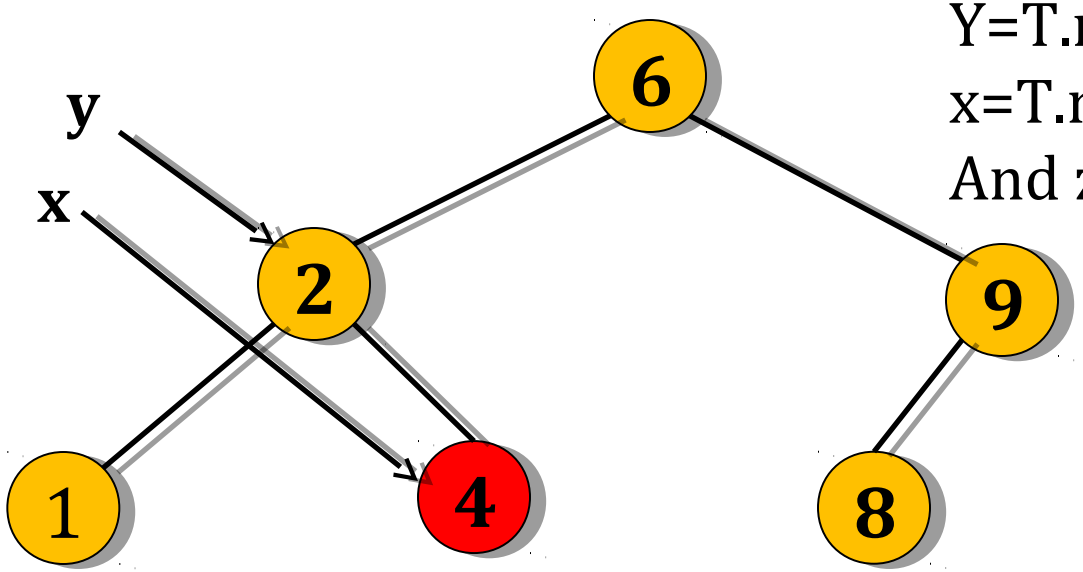
Now x≠NIL and z.key<x.key
That is [5<6]

BST Insertion Algorithm



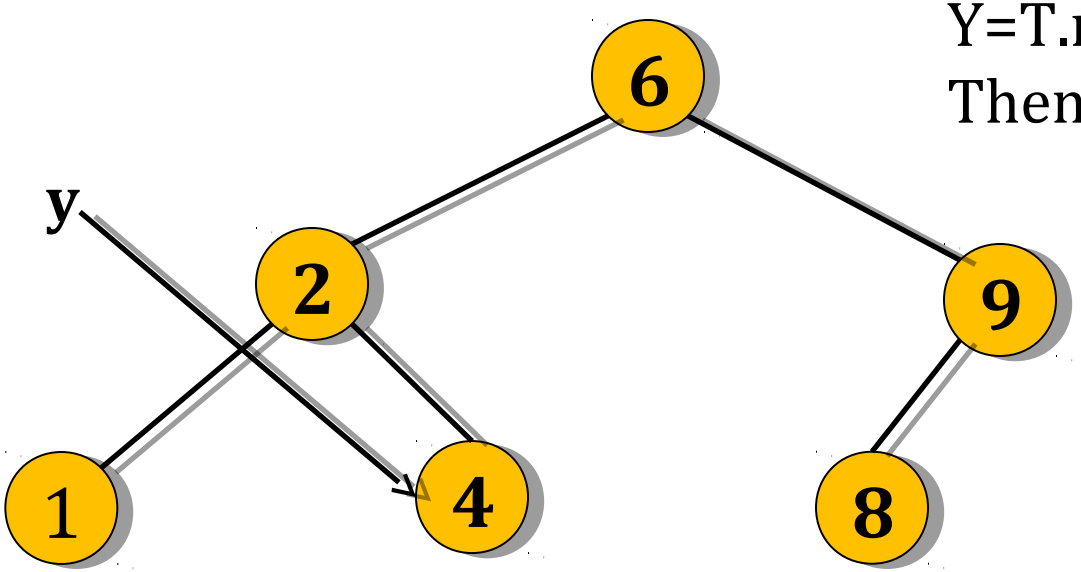
Y=T.root and x=T.root.left
And z.key>x.key[5>2]

BST Insertion Algorithm



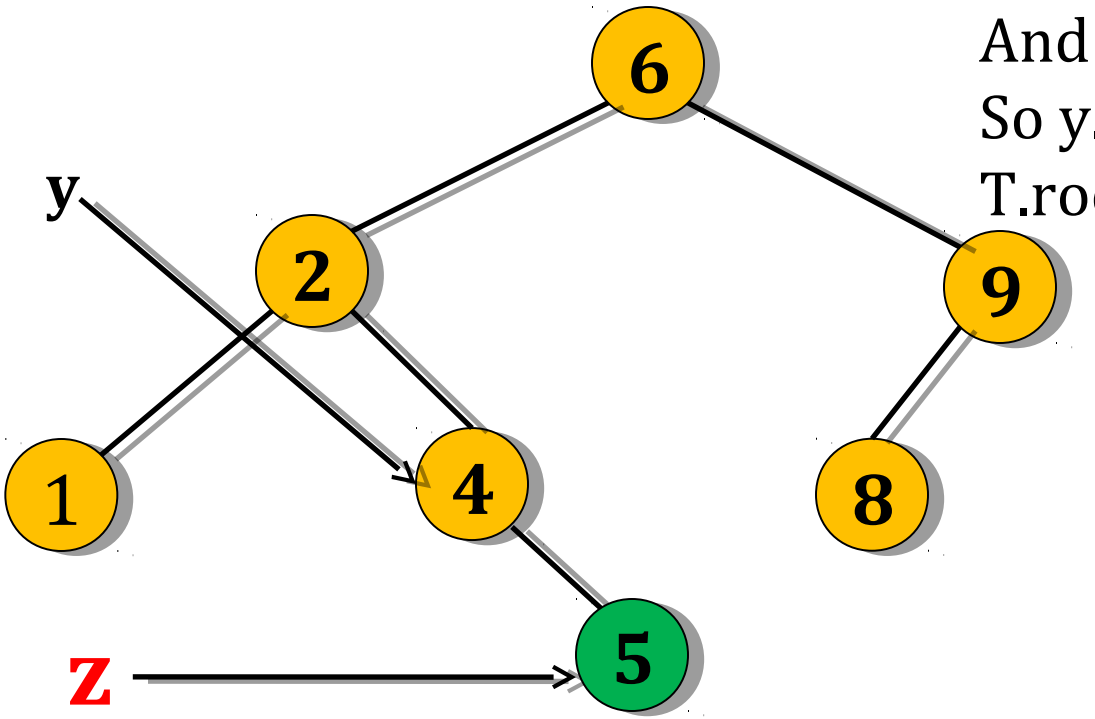
Y=T.root.left and
x=T.root.left.right
And z.key>x.key[5>4]

BST Insertion Algorithm



Y=T.root.left.right and x=NIL
Then z.p=T.root.left.right

BST Insertion Algorithm



Y=T.root.left.right [y ≠ NIL]
And z.key>y.key [5>4]
So y.right=z that is
T.root.left.right.right=z

Insertion in BST – Pseudo code

```
if  tree is empty
    create a root node with the new key
else
    compare key with the top node
    if key =  node key
        replace the node with the new value
    else if  key >  node key
        compare key with the right subtree:
         if  subtree is empty create a leaf node
         else add key  in right subtree
     else  key <  node key
        compare key with the left subtree:
         if the subtree is empty create a leaf node
         else add key to the left subtree
```

Insertion into a BST: C code

```
void insert (ptnode *node, int key)
{
  ptnode ptr,
      temp = search(*node, key);
  if (temp || !(*node)) {
   ptr = (ptnode) malloc(sizeof(tnode));
   if (IS_FULL(ptr)) {
     fprintf(stderr, "The memory is full\n");
     exit(1);
   }
   ptr->key = key;
   ptr->left = ptr->right = NULL;
   if (*node)
     if (key<temp->key) temp->left=ptr;
        else temp->right = ptr;
   else *node = ptr;
  }
}
```

Delete a value from the BST

- Removing a node from a BST is a bit more complex, since we do not want to create any "holes" in the tree. The intention is to **remove** the specified item from the BST and **adjusts** the tree
- The binary search algorithm is used to locate the target item: **starting at the root** it probes down the tree till it finds the target or reaches a leaf node (target not in the tree)
- If the node has one child then the child is spliced to the parent of the node. If the node has two children then its successor has no left child; copy the successor into the node and delete the successor instead TREE-DELETE (T, z) removes the node pointed to by z from the tree T. IT returns a pointer to the node removed so that the node can be put on a free-node list
- The overall strategy for deleting a node or item from a binary search tree can be described through some cases.

Delete a value from the BST

Experimenting the cases:

- if the tree is empty return false
- else Attempt to locate the node containing the target using the binary search algorithm:

  if the target is not found return false

  else the target is found, then remove its node. Now while removing the node four cases may happen

Delete a value from the BST

Case 1: if the node has 2 empty subtrees
- replace the link in the parent with null

Case 2: if the node has a left and a right subtree
- replace the node's value with the max value in the left subtree
- delete the max node in the left subtree

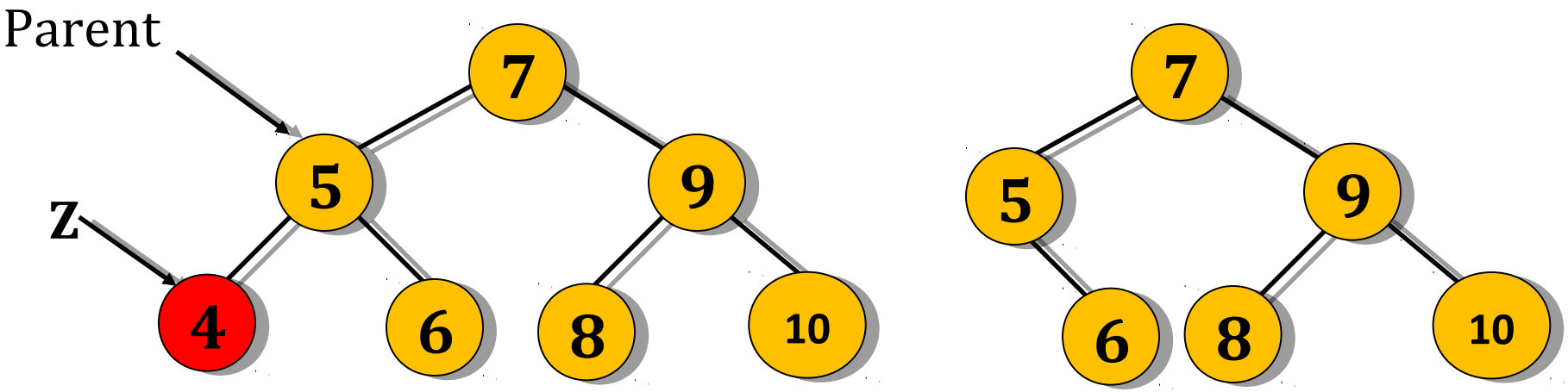
Case 3: if the node has no left child
- link the parent of the node to the right (non-empty) subtree

Case 4: if the node has no right child
- link the parent of the target to the left (non-empty) subtree

Delete a value from the BST

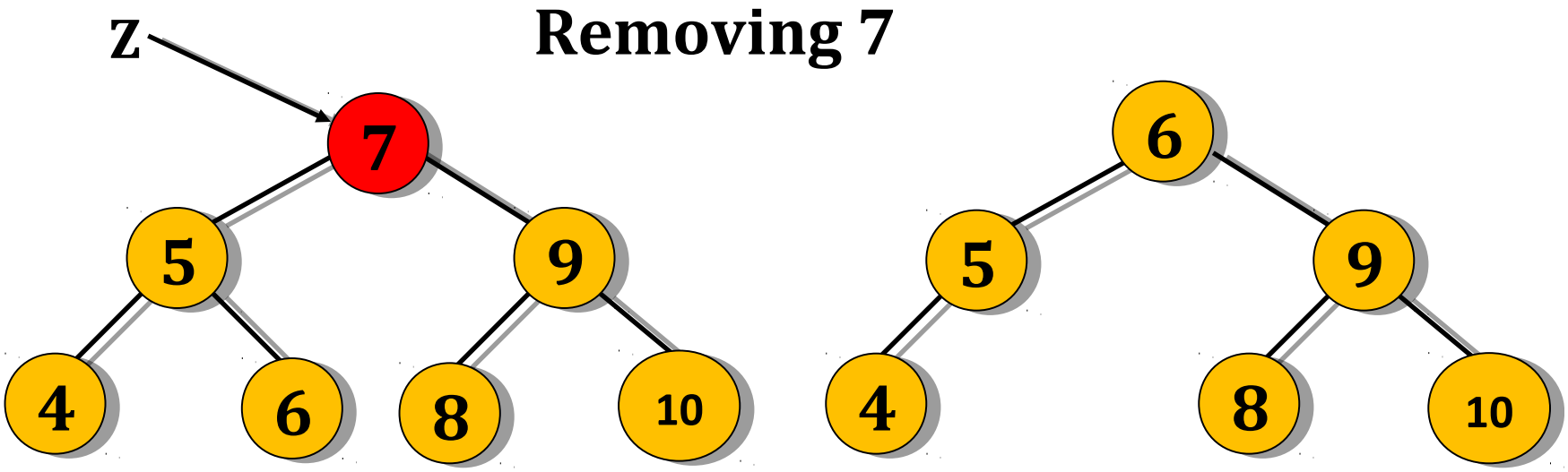
Case 1: removing a node with 2 EMPTY SUBTREES
-replace the link in the parent with null

Removing 4



Delete a value from the BST

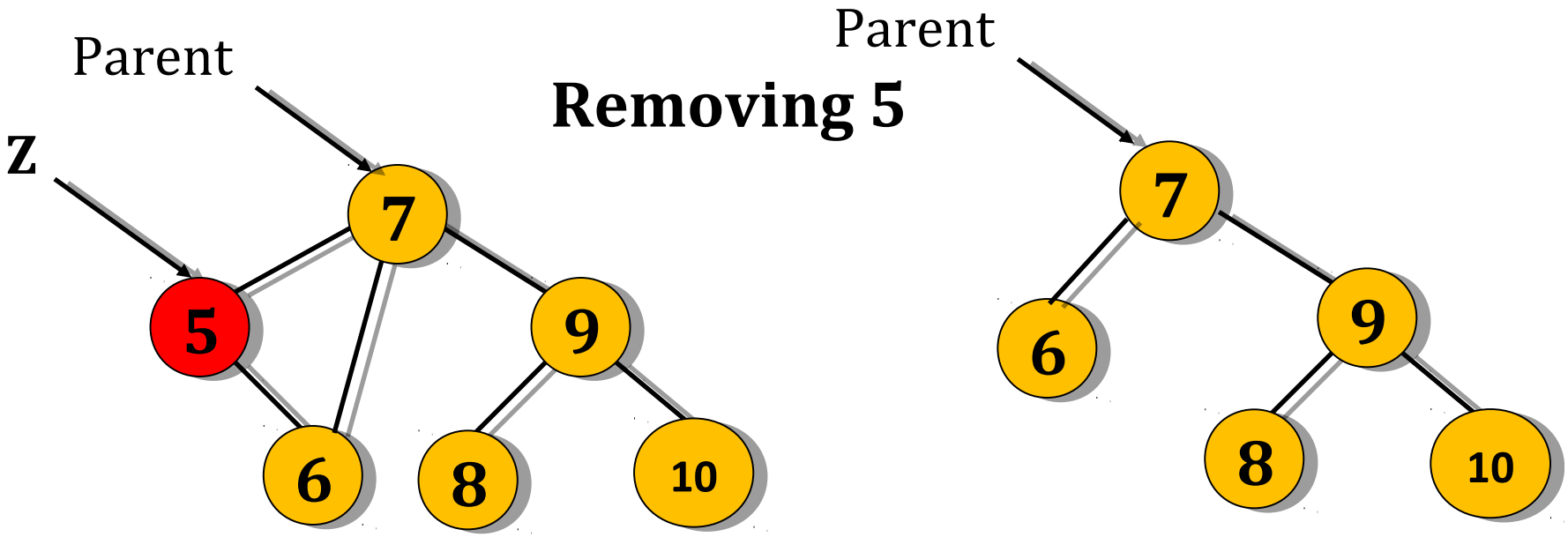
Case 2: removing a node with 2 SUBTREES
- replace the node's value with the max value in the left subtree
- delete the max node in the left subtree



Delete a value from the BST

Case 3: if the node has no left child

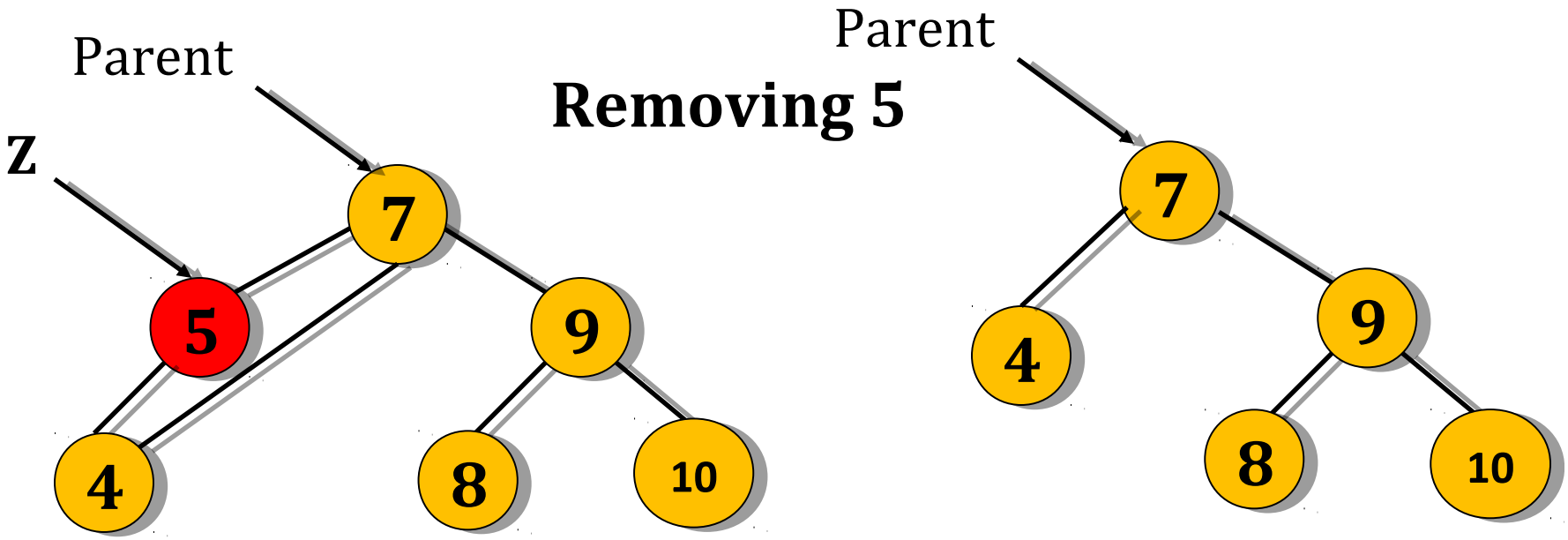
- link the parent of the node to the right (non-empty) subtree



Delete a value from the BST

Case 4: if the node has no right child

- link the parent of the node to the left (non-empty) subtree



BST Deletion Algorithm

- **TREE-DELETE (T, z)**

```
1.  if left [z] = NIL    .OR.    right[z] = NIL
2.      then y ← z
3.      else y ← TREE-SUCCESSOR (z)
4.  if left [y] ≠ NIL
5.      then x ← left[y]
6.      else x ← right [y]
7.  if x ≠ NIL
8.      then p[x] ← p[y]
9.  if p[y] = NIL
10.     then root [T] ← x
11.     else if y = left [p[y]]
12.         then left [p[y]] ← x
13.         else right [p[y]] ← x
14. if y ≠ z
15.     then key [z] ← key [y]
16.         if y has other field, copy them, too
17. return y
```

Analysis of BST Operations

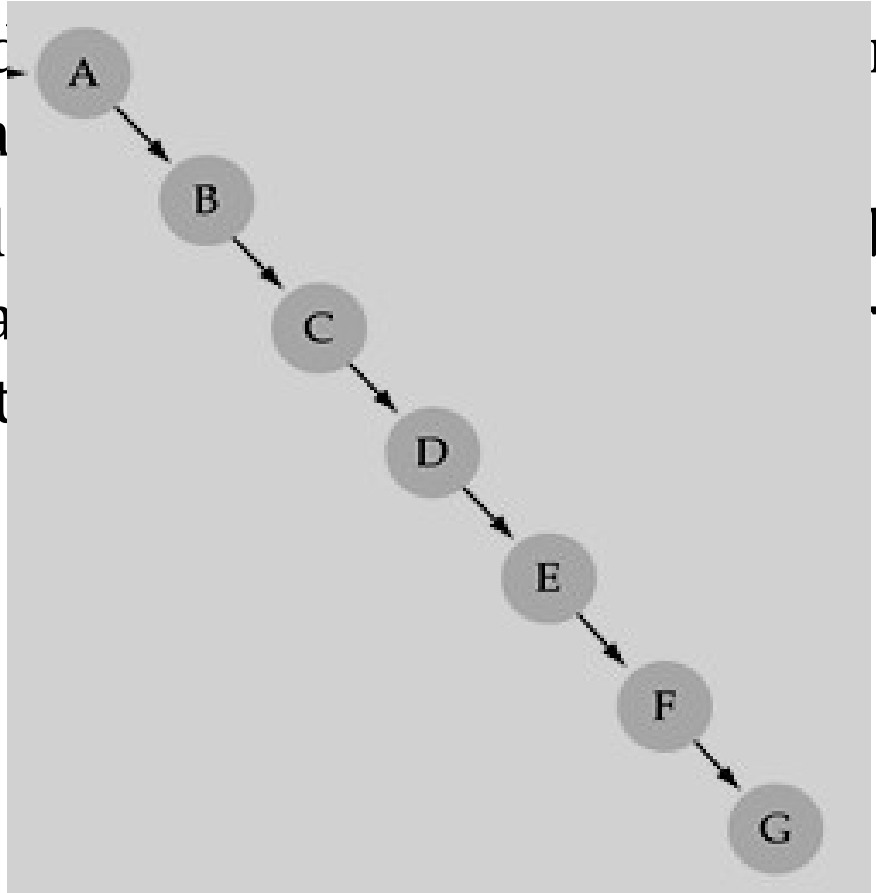
- The complexity of operations **search**, **insert** and **remove** in BST is O(h) , where h is the height.
- When the tree is balanced then it is O(log n). The updating operations cause the tree to become unbalanced.
- The tree can degenerate to a linear shape and the operations will become O (n)

Applications for BST

- Binary Search Tree: Used in *many* search applications where data is constantly entering/leaving, such as the map and set objects in many languages' libraries.
- Binary Space Partition: Used in almost every 3D video game to determine what objects need to be rendered.
- Binary Tries: Used in almost every high-bandwidth router for storing router-tables.
- Heaps: Used in implementing efficient priority-queues. Also used in heap-sort.
- Huffman Coding Tree:- used in compression algorithms, such as those used by the .jpeg and .mp3 file-formats.
- GGM Trees - Used in cryptographic applications to generate a tree of pseudo-random numbers.
- Syntax Tree: Constructed by compilers and (implicitly) calculators to parse expressions.

What is a Degenerate BST?

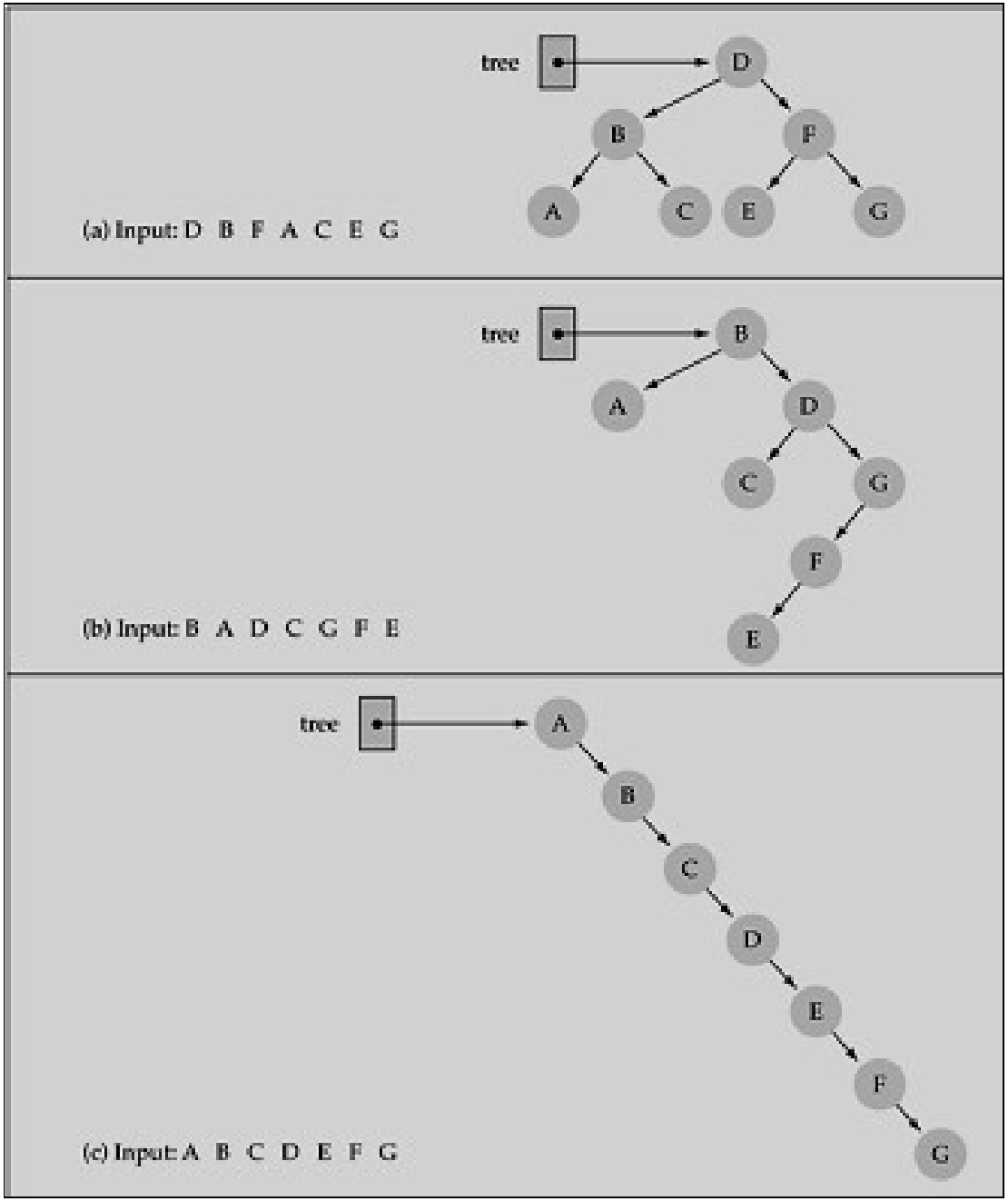
- A degenerate binary search tree is one where most or all of the nodes contain only one sub node.
- It is unbalanced rmance degrades to tha
- If your add nod balancing, then you can ea ee by feeding it data t



Does the order of inserting elements into a tree matter?

- Yes, certain orders might produce very unbalanced trees or degenerated trees!
- Unbalanced trees are not desirable because search time increases!
- Advanced tree structures, such as **red-black trees, AVL trees**, guarantee balanced trees.

Does the order of inserting elements into a tree matter?



Better Search Trees

Prevent the degeneration of the BST :

- A BST can be set up to maintain balance during updating operations (insertions and removals)
- Types of ST which maintain the optimal performance in other words balanced trees:
  - splay trees
  - AVL trees
  - 2-4 Trees
  - Red-Black trees
  - B-trees

References

- Introduction to Algorithms by Thomas H. Cormen and others
- Binary Search Tree by Penelope Hofsdal
- Rada Mihalcea

http://www.cs.unt.edu/~rada/CSCE3110

MD. Shakhawat Hossain
Student of Computer Science & Engineering Dept.
University of Rajshahi