Algorithms and Data Structures

Andrzej Pisarski

Plan of the lecture

- Shell Sort
- Partitioning
- Quicksort
- Binary Trees
  - Tree terminology
  - Basic Binary Tree operations
    - Finding a Node
    - Inserting a Node
  - Traversing the Tree
  - Finding maximum and minimum values
  - *Deleting a Node*
  - The efficiency of Binary Trees

Shell Sort

A1	A2	A3	A4	A5	A6	A7	A8

Array step={1,2,4}; step = 4

A1	A5

A2	A6

A3	A7

A4	A8

Shell Sort

A1	A2	A3	A4	A5	A6	A7	A8

Array step={1,2,4}; step = 2

A1	A3	A5	A7

A2	A4	A6	A8

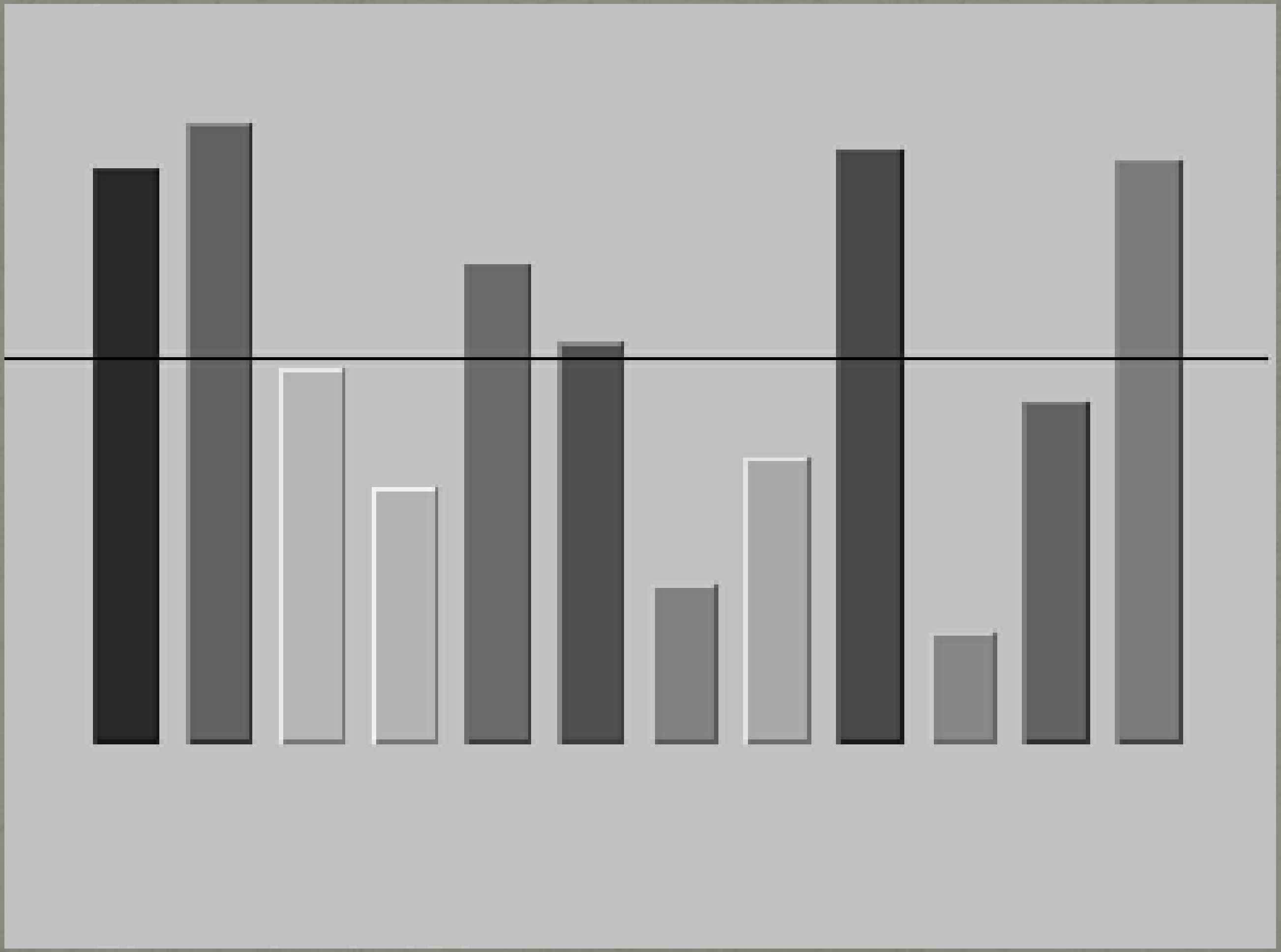
Shell Sort

A1	A2	A3	A4	A5	A6	A7	A8

Array step={1,2,4}; step = 1

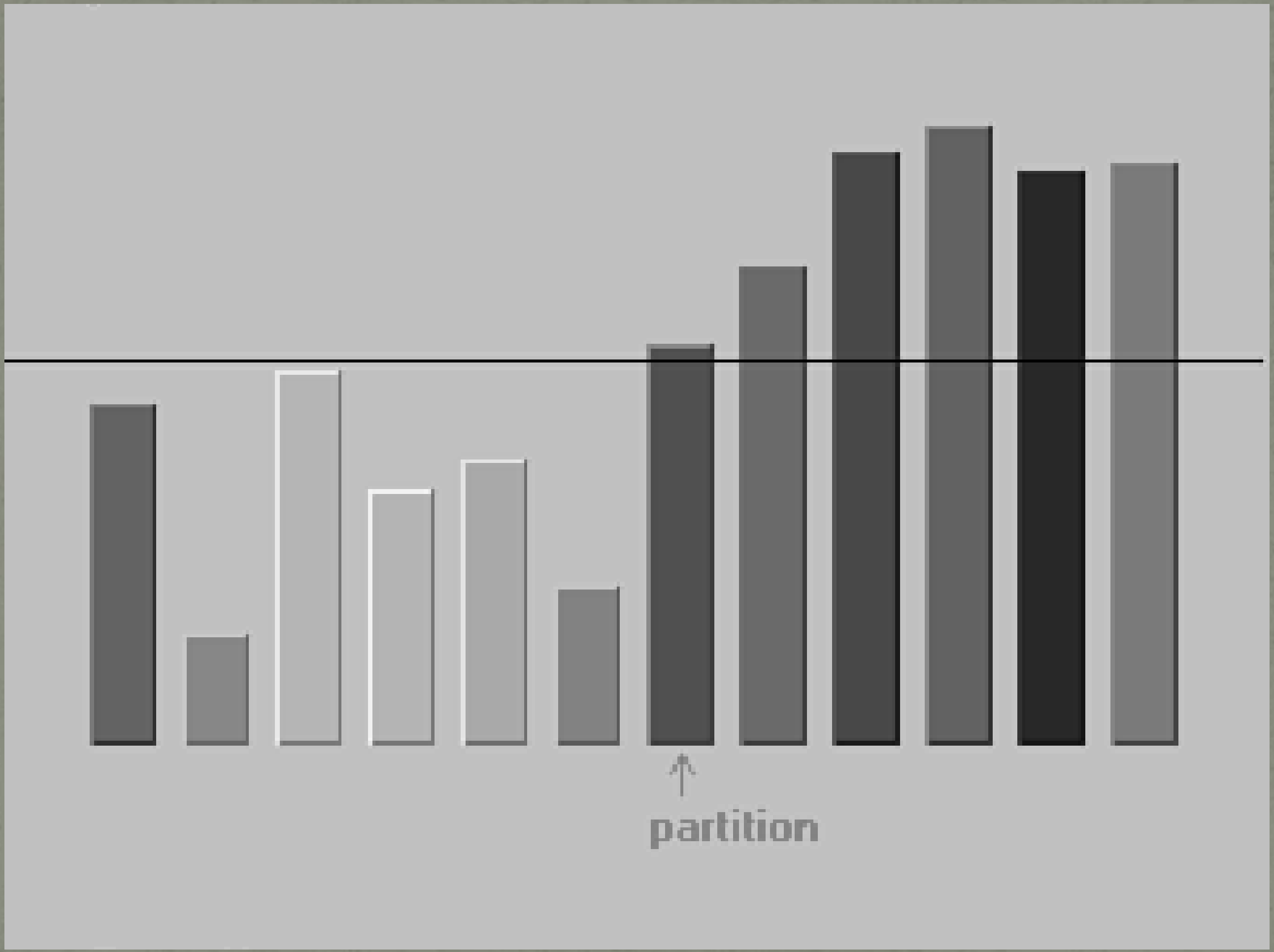
A1	A2	A3	A4	A5	A6	A7	A8

Partitioning



(*example from R. Lafore book*)

Partitioning



(*example from R. Lafore book*)

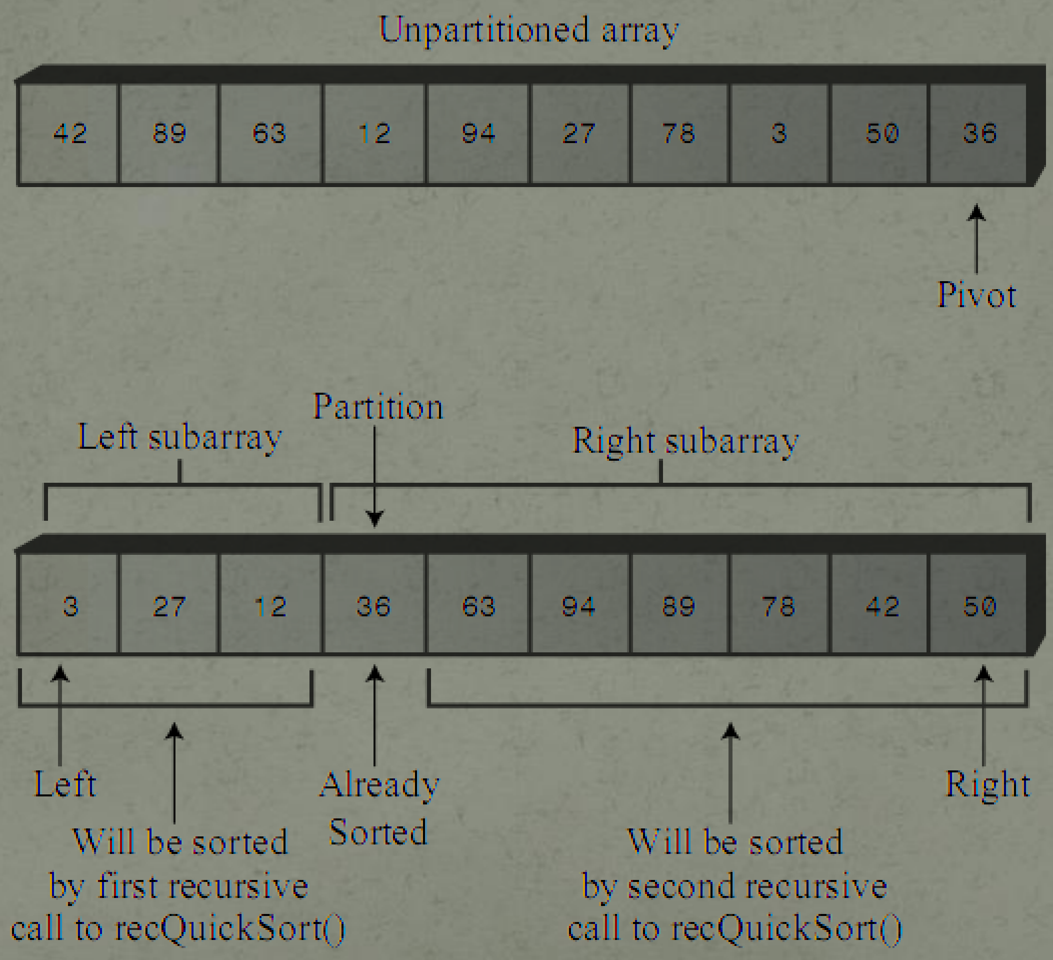
Quicksort

```
void recQuickSort(int left, int right)
   {
   if(right-left <= 0)          //if size is 1,
       return;                  //   it's already sorted
   else                         //size is 2 or larger
      {
                                           //partition range
      int partition = partitionIt(left, right);

    recQuickSort(left, partition-1);   //sort left side
    recQuickSort(partition+1, right);  //sort right side
    }
 }
```

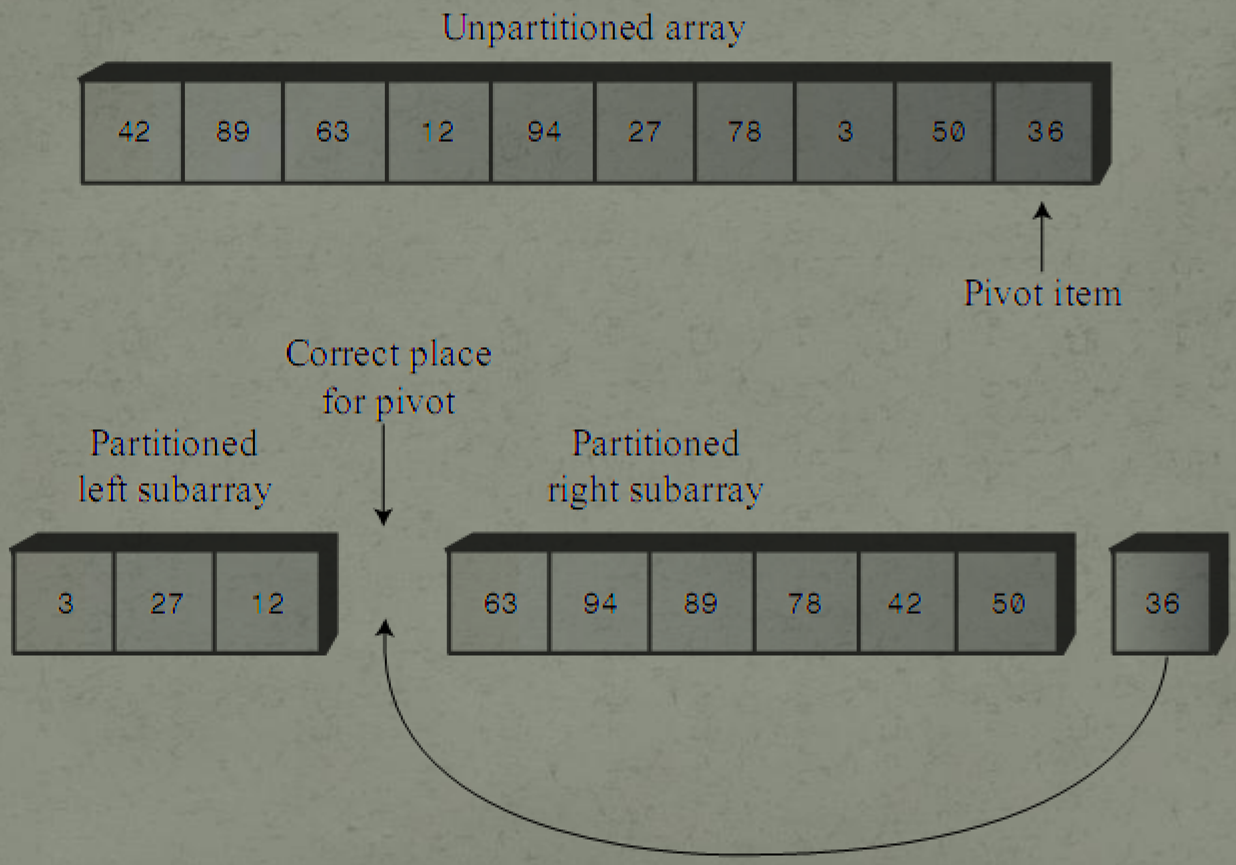
(*example from R. Lafore book*)

Quicksort



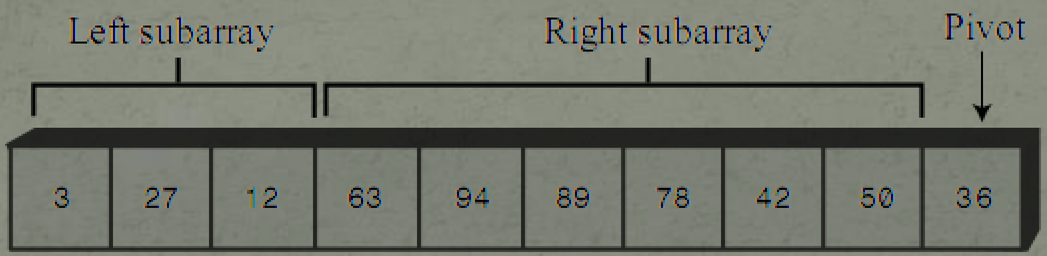
(*example from R. Lafore book*)

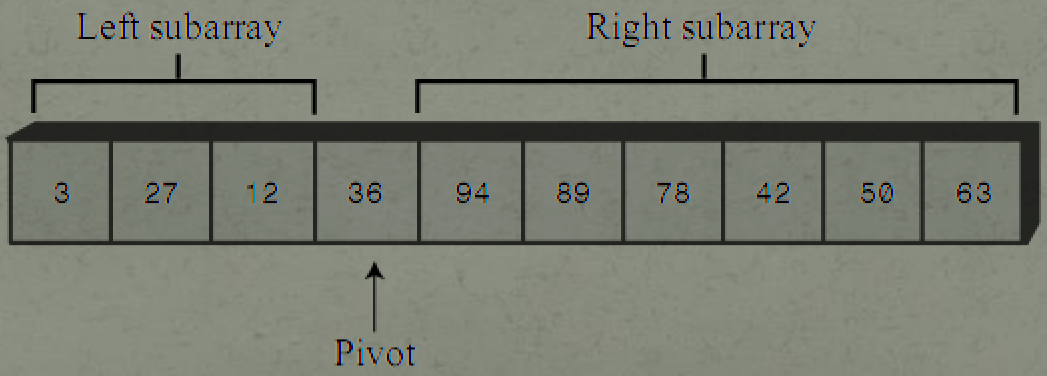
Quicksort



(*example from R. Lafore book*)

Quicksort





(*example from R. Lafore book*)

Quicksort: efficiency

- Quicksort operating in O(N*logN) time (for sorting in memory is the fastest method in majority simulations)

(*example from R. Lafore book*)

Binary Trees

Tree terminology

- Path – sequence of nodes: walking from node to node along the edges
- Root – the node at the top of the tree
- Parent –node above chosen one (except the root)
- Child – a node being a „child” of the parent node (conected by line)
- Leaf – a node without a children,
- Subtree – a node considerate to be a root of sub-tree.
- Visiting – action during visiting a node (checking the key value, displaying it, etc.); without action it is merely passing over a node.

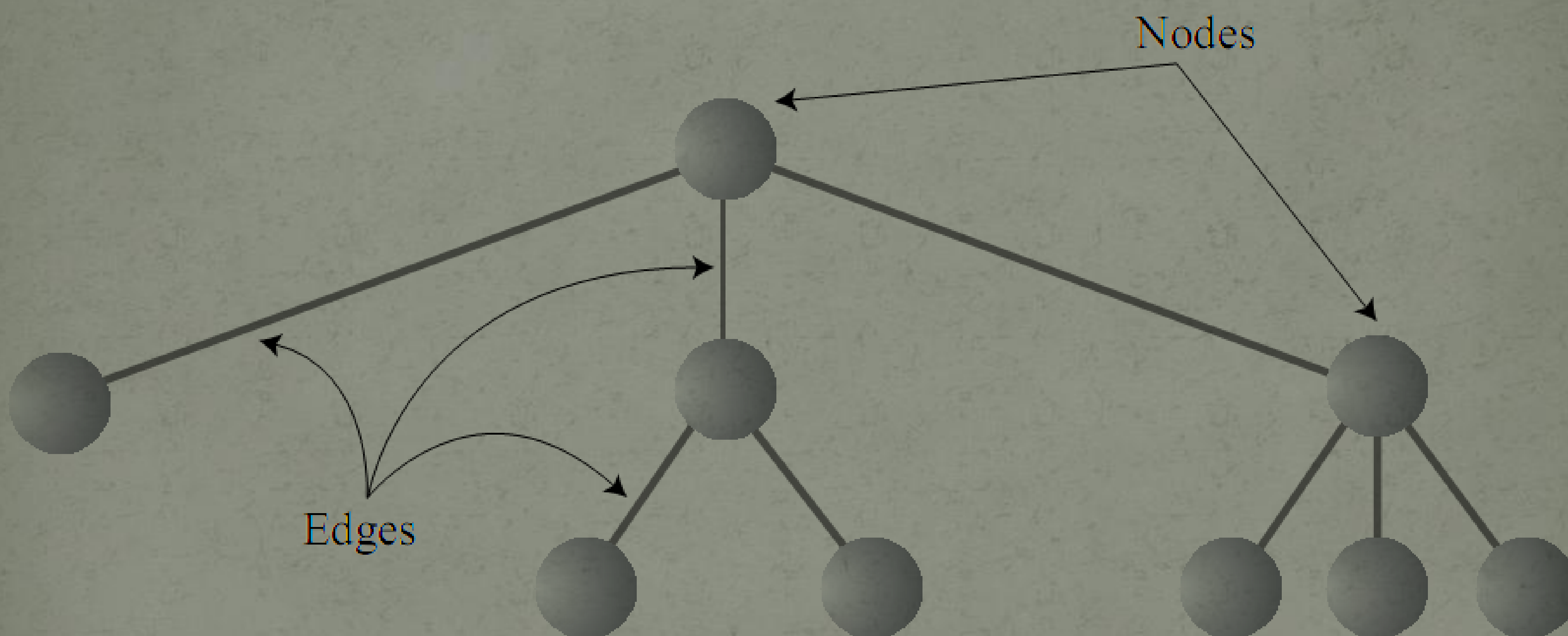
(*example from R. Lafore book*)

Tree terminology

- Traversing – to visit all nodes is some order (inorder, preorder, postorder)
- Levels – level refers to a number which tell us how many generations of nodes is from the root (to particular node)
- Keys – key value(s) stored in the node
- Binary trees – a node can have no more than two children

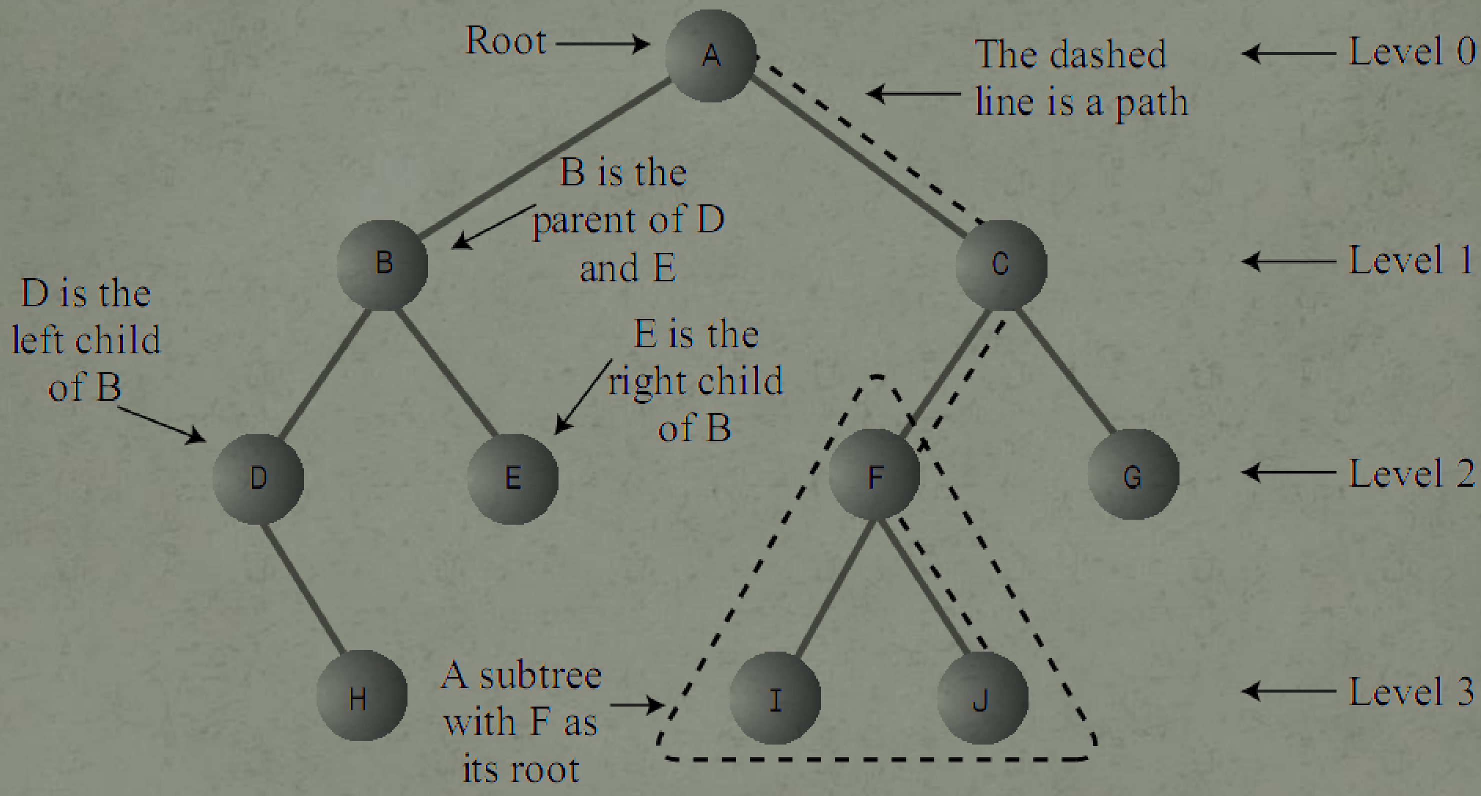
(*example from R. Lafore book*)

Tree terminology



(*example from R. Lafore book*)

Tree terminology

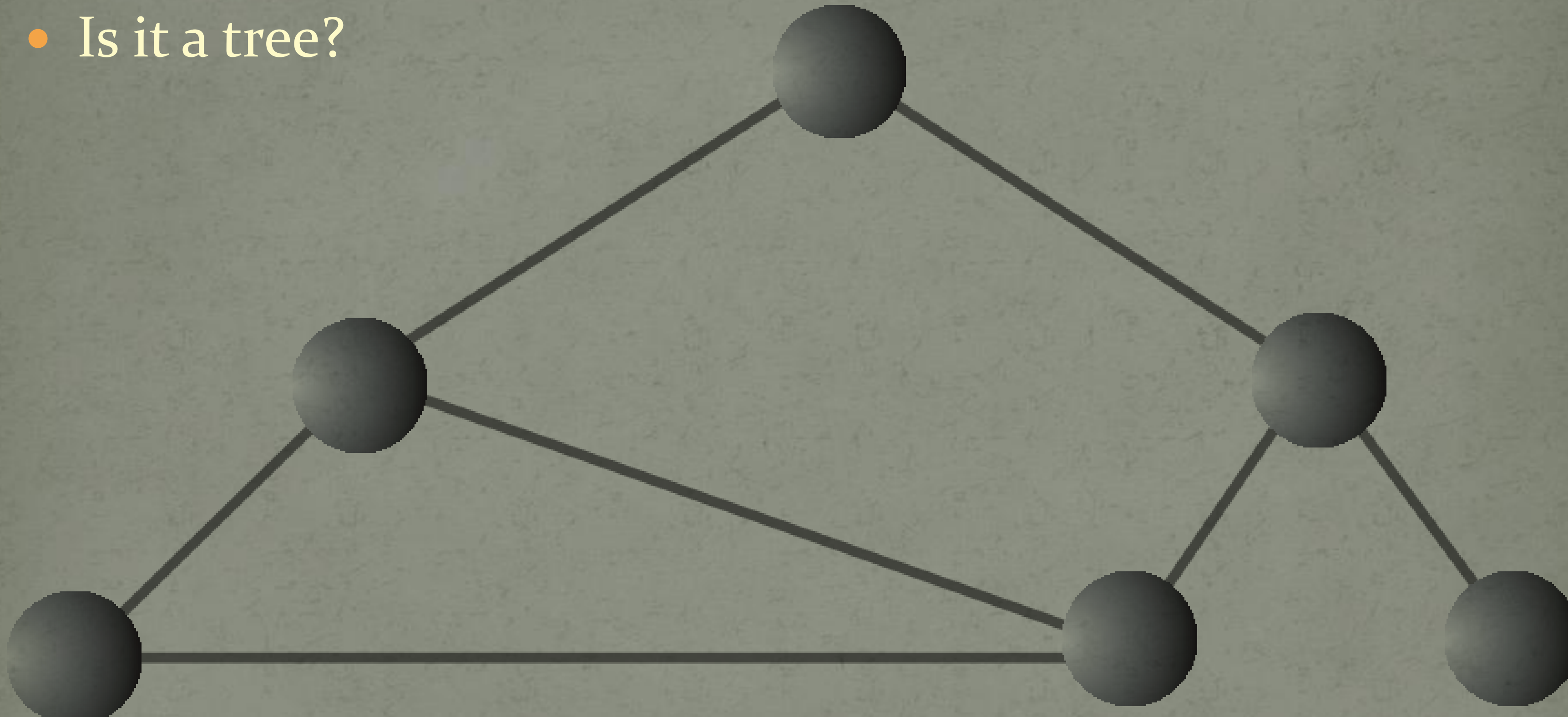


H, E, I, J, and G are leaf nodes

(example from R. Lafore book)

Tree terminology

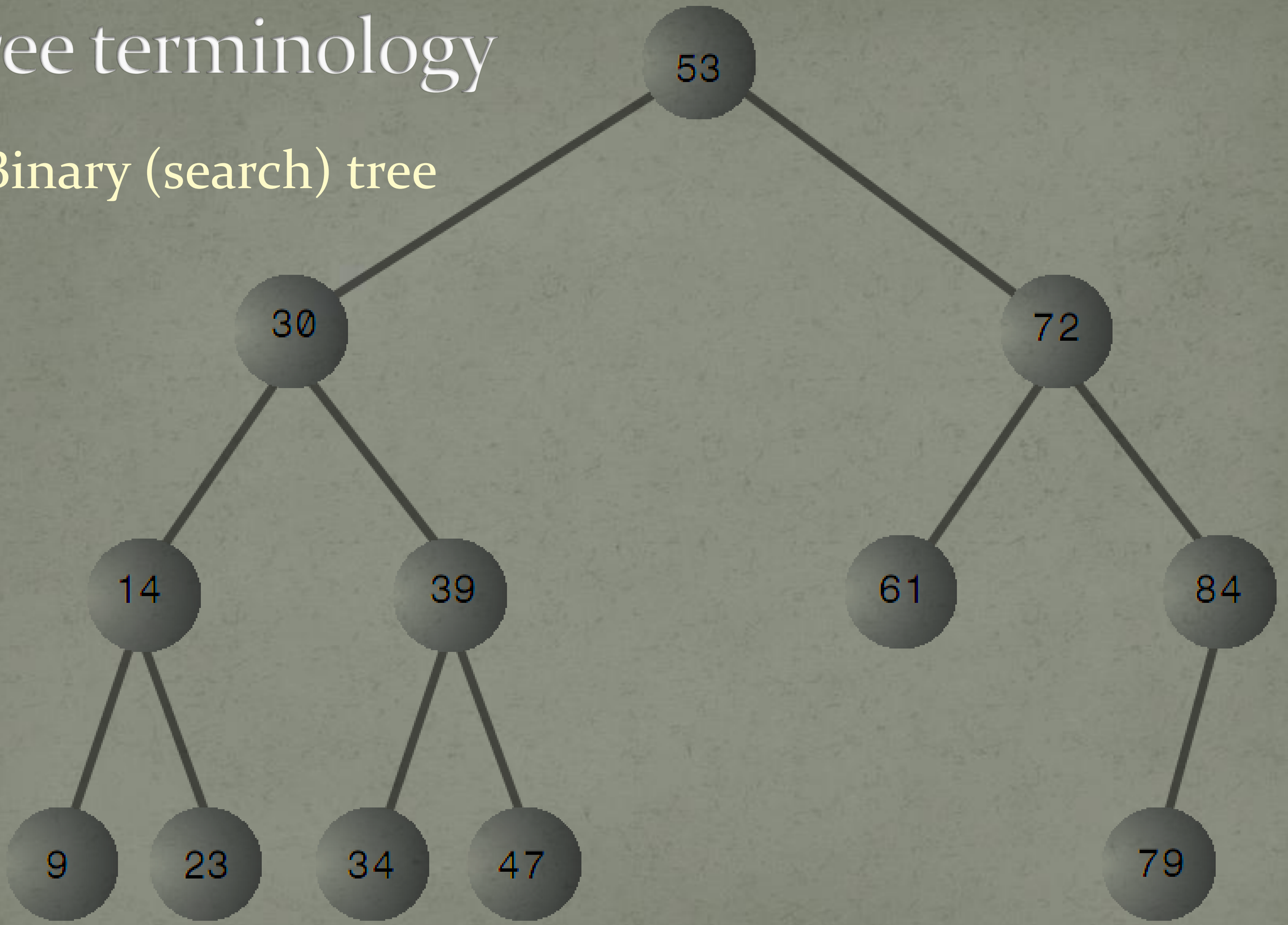
- Is it a tree?



(*example from R. Lafore book*)

Tree terminology

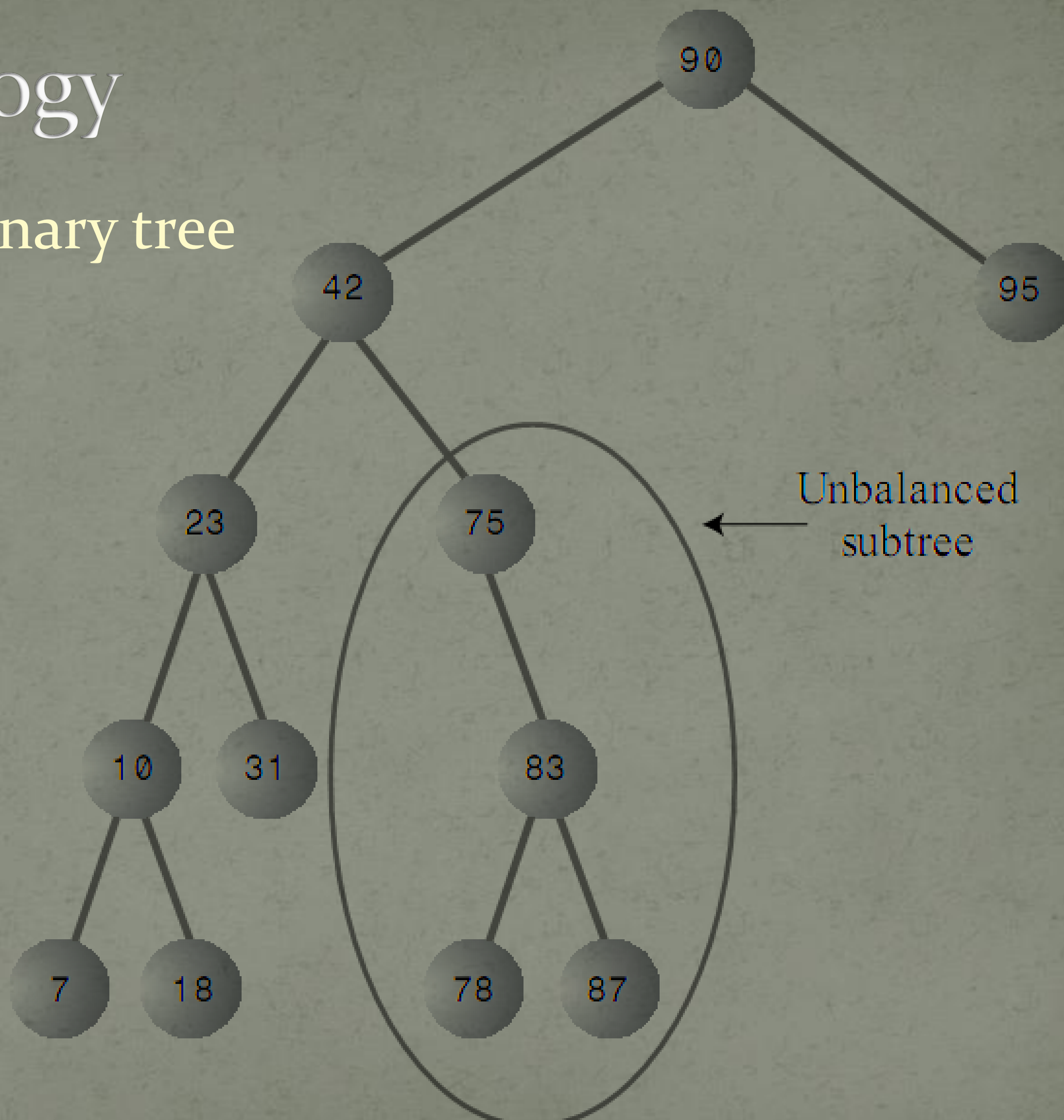
- Binary (search) tree



(example from R. Lafore book)

Tree terminology

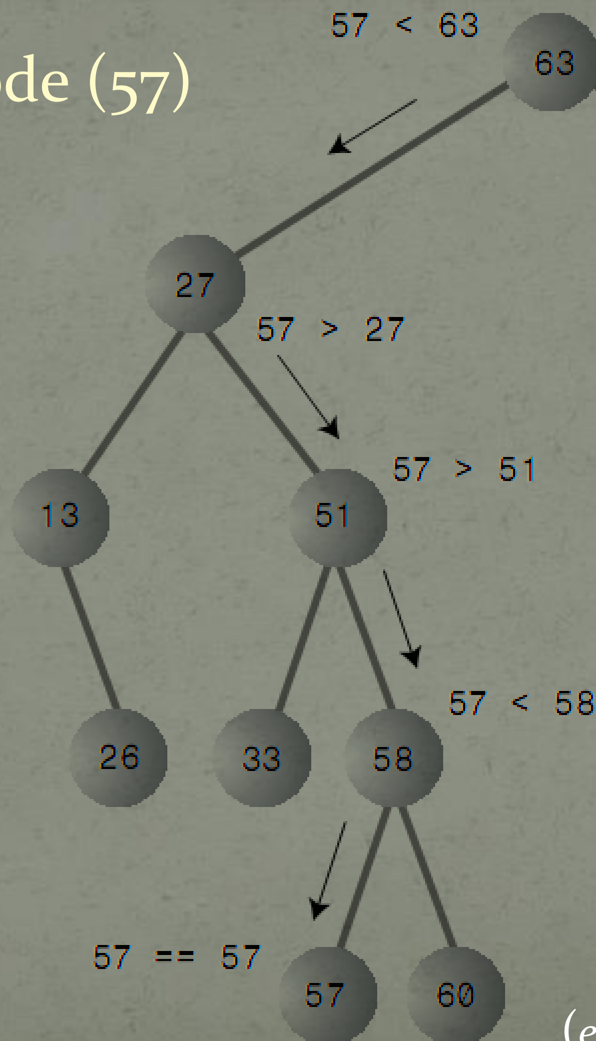
- An unbalanced binary tree



(*example from R. Lafore book*)

Basic Binary Tree operations

- Finding a Node (57)



(example from R. Lafore book)

Basic Binary Tree operations

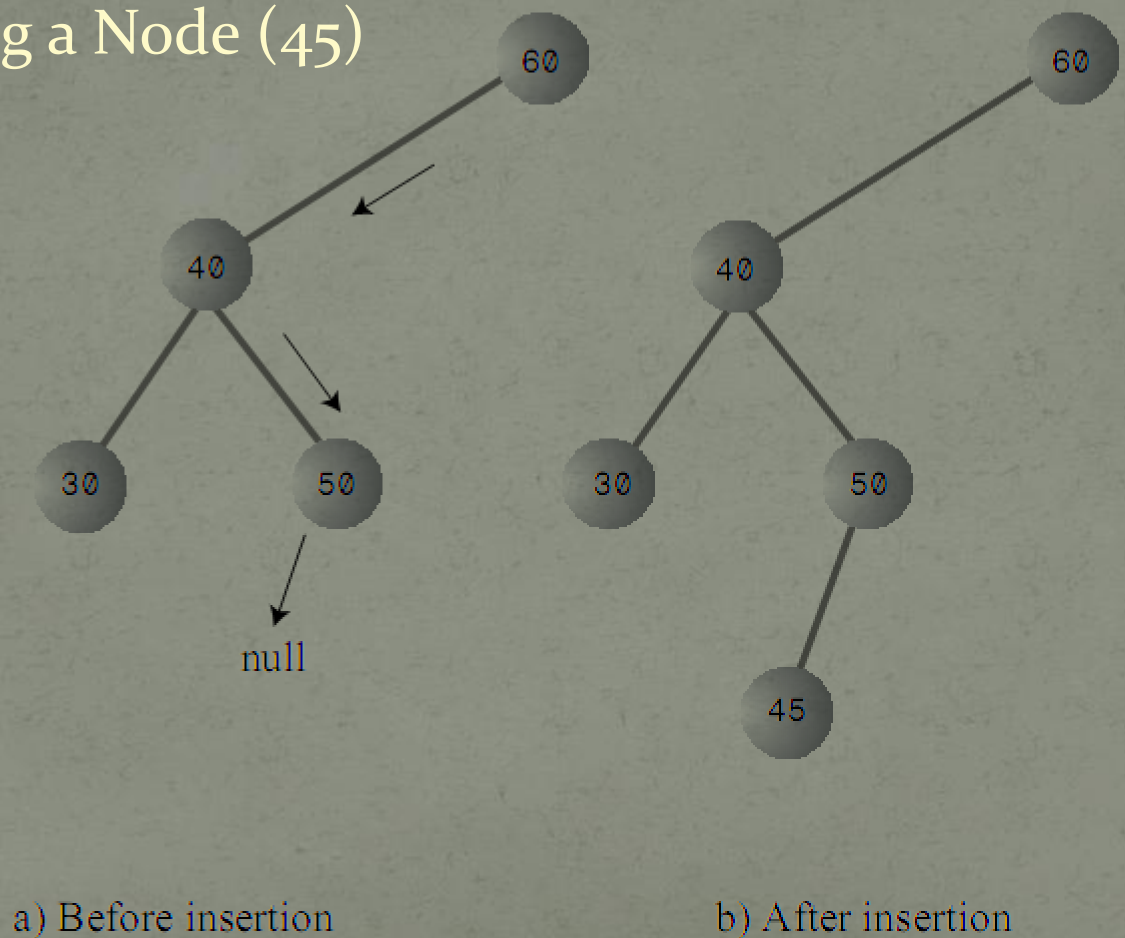
- Finding a Node (57)

```
Node* find(int key)              //find node with given key
   {                             //(assumes non-empty tree)
   Node* pCurrent = pRoot;                //start at root
   while(pCurrent->iData != key)          //while no match,
      {
      if(key < pCurrent->iData)           //go left?
         pCurrent = pCurrent->pLeftChild;
      else                                //or go right?
         pCurrent = pCurrent->pRightChild;
      if(pCurrent == NULL)                //if no child,
         return NULL;                     //didn't find it
      }
   return pCurrent;                       //found it
   }  //end find()
```

(*example from R. Lafore book*)

Basic Binary Tree operations

- Inserting a Node (45)



(*example from R. Lafore book*)

Basic Binary Tree operations

- Inserting a Node (45)

```
void insert(int id, double dd) //insert new node
    {
    Node* pNewNode = new Node;              //make new node
    pNewNode->iData = id;                   //insert data
    pNewNode->dData = dd;
    if(pRoot==NULL)                         //no node in root
        pRoot = pNewNode;
    else                                    //root occupied
        {
        Node* pCurrent = pRoot;             //start at root
        Node* pParent;
        while(true)                         //(exits internally)
            {
            pParent = pCurrent;
            if(id < pCurrent->iData)        //go left?
                {
                pCurrent = pCurrent->pLeftChild;
                if(pCurrent == NULL)        //if end of the line,
                    {                       //insert on left
                    pParent->pLeftChild = pNewNode;
                    return;
                    }
                }  //end if go left
            else                            //or go right
                {
```

(*example from R. Lafore book*)

Basic Binary Tree operations

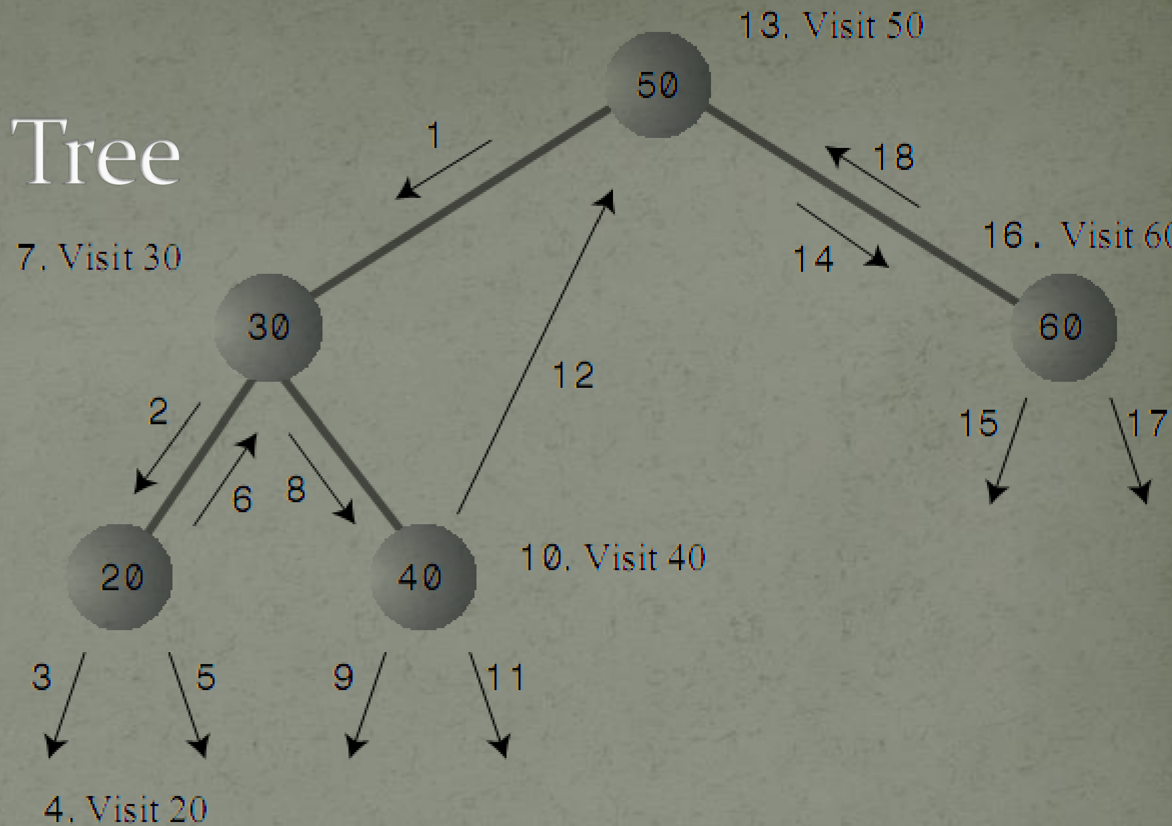
- Inserting a Node (45)

```
        else                                    //or go right?
            {
            pCurrent = pCurrent->pRightChild;
            if(pCurrent == NULL)                //if end of the line
                {                               //insert on right
                pParent->pRightChild = pNewNode;
                return;
                }
            }  //end else go right
        }  //end while
    }  //end else not root
}  //end insert()
```

(*example from R. Lafore book*)

Traversing the Tree

- inOrder()
- preOrder()
- postOrder()

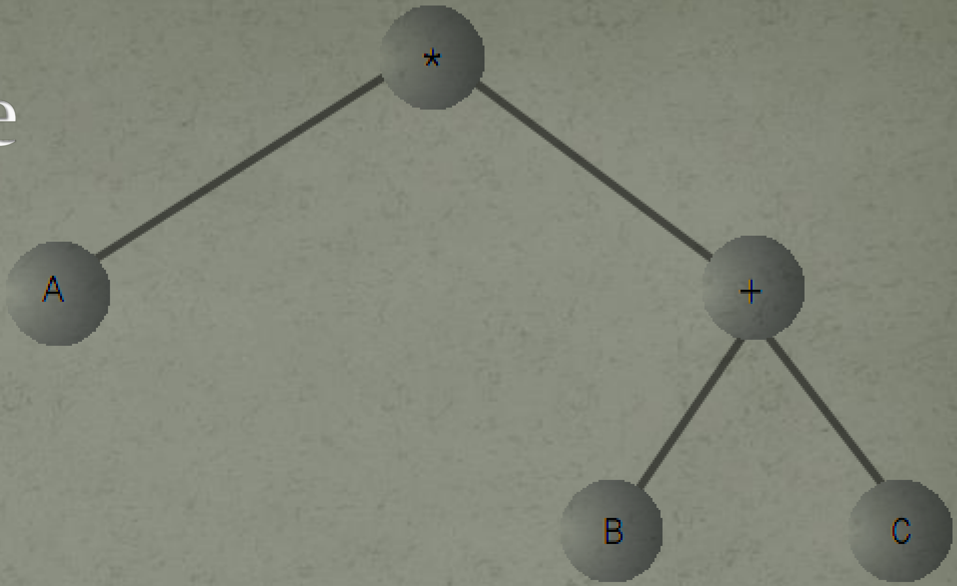



```
void inOrder(Node* pLocalRoot)
   {
   if(pLocalRoot != NULL)
      {
      inOrder(pLocalRoot->pLeftChild);     //left child
      cout << pLocalRoot->iData << " ";    //display node
      inOrder(pLocalRoot->pRightChild);    //right child
      }
   }
```

(*example from R. Lafore book*)

Traversing the Tree

- inOrder()
- preOrder()
- postOrder()



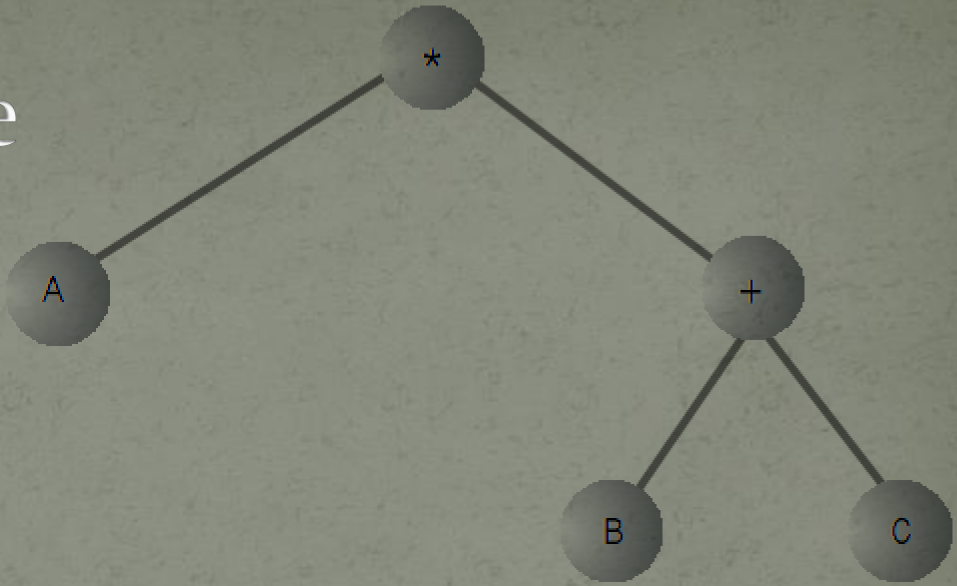

```
Infix: A * (B + C)
Prefix: *A + BC
Postfix: ABC + *
```

```
void preOrder(Node* pLocalRoot)
{
    if(pLocalRoot != NULL)
    {
        cout << pLocalRoot->iData << " ";    //display node
        preOrder(pLocalRoot->pLeftChild);    //left child
        preOrder(pLocalRoot->pRightChild);   //right child
    }
}
```

(*example from R. Lafore book*)

Traversing the Tree

- inOrder()
- preOrder()
- postOrder()




```
Infix: A * (B + C)
Prefix: *A + BC
Postfix: ABC + *
```

```
void postOrder(Node* pLocalRoot)
{
    if(pLocalRoot != NULL)
    {
        postOrder(pLocalRoot->pLeftChild);   //left child
        postOrder(pLocalRoot->pRightChild);  //right child
        cout << pLocalRoot->iData << " ";    //display node
    }
}
```

(*example from R. Lafore book*)

Finding maximum and minimum values

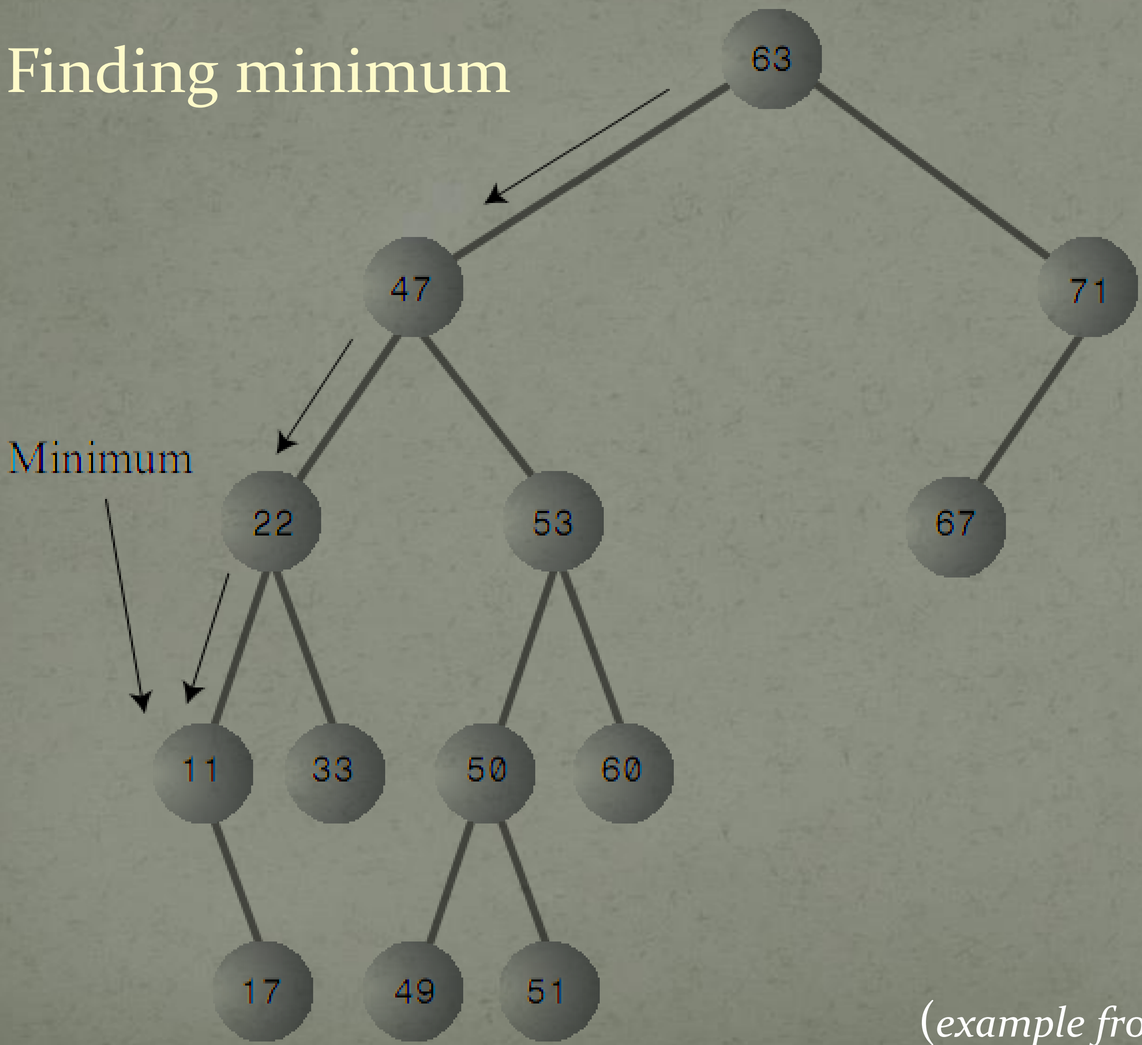
- Finding minimum

```
Node* minimum()      // returns node with minimum key value
    {
    Node* pCurrent, pLast;
    pCurrent = pRoot;                        //start at root
    while(pCurrent != NULL)                  //until the bottom,
        {
        pLast = pCurrent;                    //remember node
        pCurrent = pCurrent->pLeftChild;     //go to left child
        }
    return pLast;
    }
```

(*example from R. Lafore book*)

Finding maximum and minimum values

- Finding minimum



(*example from R. Lafore book*)

The efficiency of Binary Trees

- Operations like finding a particular node involve descending the tree from level 0 to level with search node. How long (many operations) it will take to do this? (for full tree)

(*example from R. Lafore book*)

The efficiency of Binary Trees

- During a search we need to visit a one node per level.

Number of Nodes	Number of Levels
1	1
3	2
7	3
15	4
31	5
...	...
1,023	10
...	...
32,767	15
...	...
1,048,575	20

N=2^L -1 //+1

N+1=2^L

$L=\log_2(N+1)$

O(log N)

(*example from R. Lafore book*)