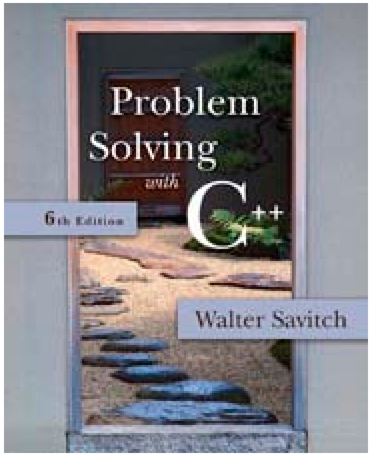
APS105: Lecture 35

Wael Aboelsaadat

wael@cs.toronto.edu

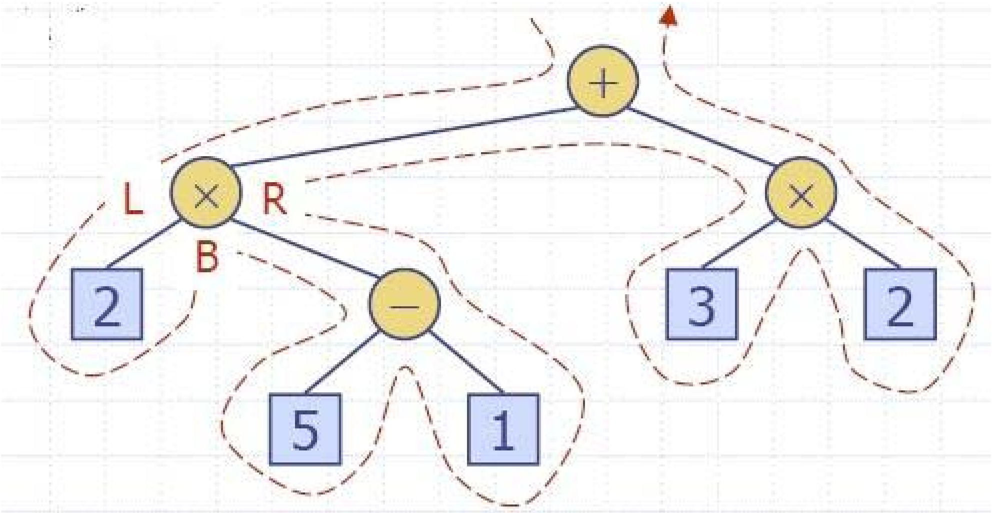
http://ccnet3.utoronto.ca/20079/aps105h1f/

Acknowledgement: These slides are a modified version of the text book slides as supplied by Addison Wesley



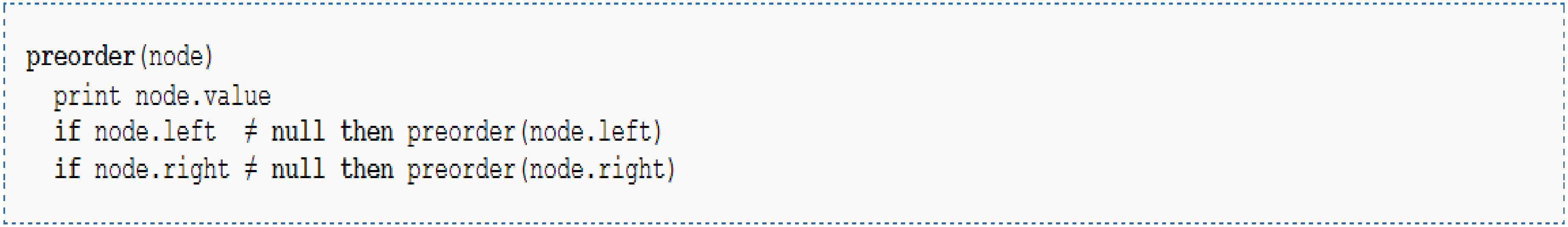
Revision: pointers & recursion

Binary Tree Traversal

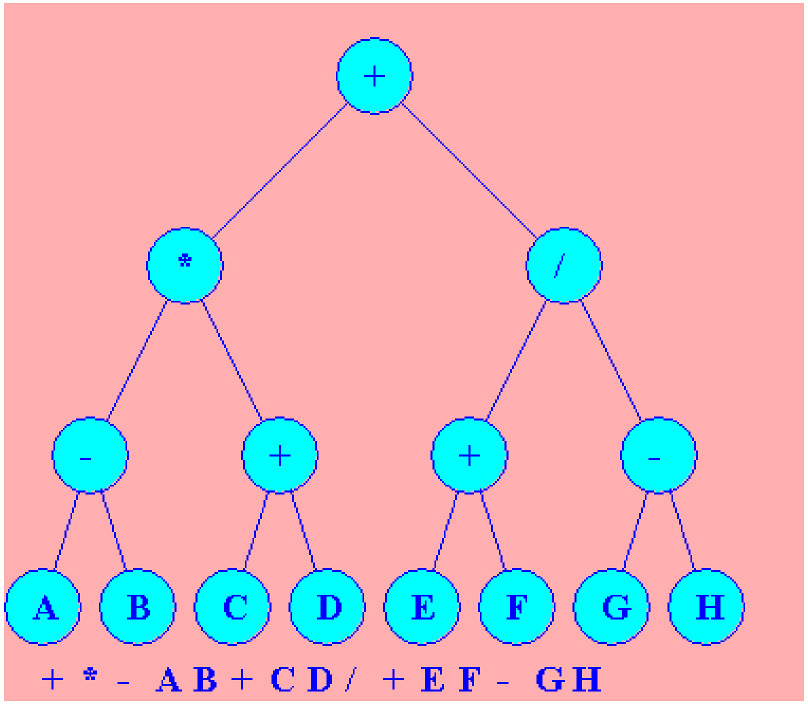


http://www.cosc.canterbury.ac.nz/mukundan/dsal/BTree.html
http://en.wikipedia.org/wiki/Tree_traversal

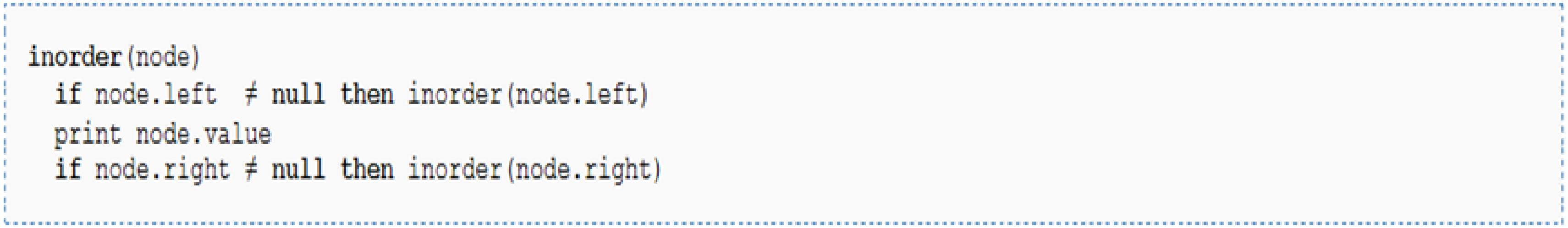
Binary Tree Traversal – pseudo code



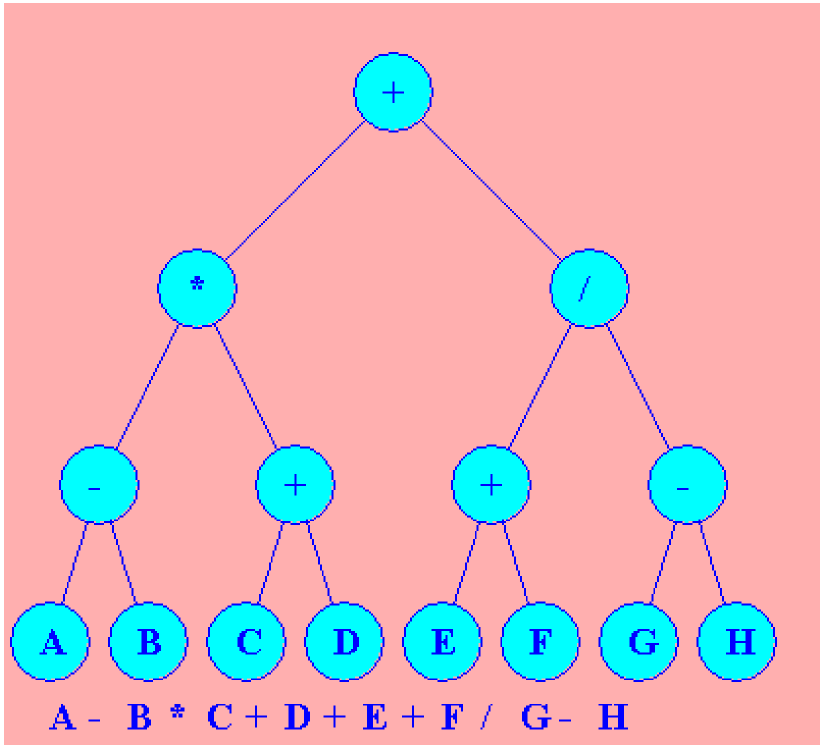
```
preorder(node)
  print node.value
  if node.left  ≠ null then preorder(node.left)
  if node.right ≠ null then preorder(node.right)
```



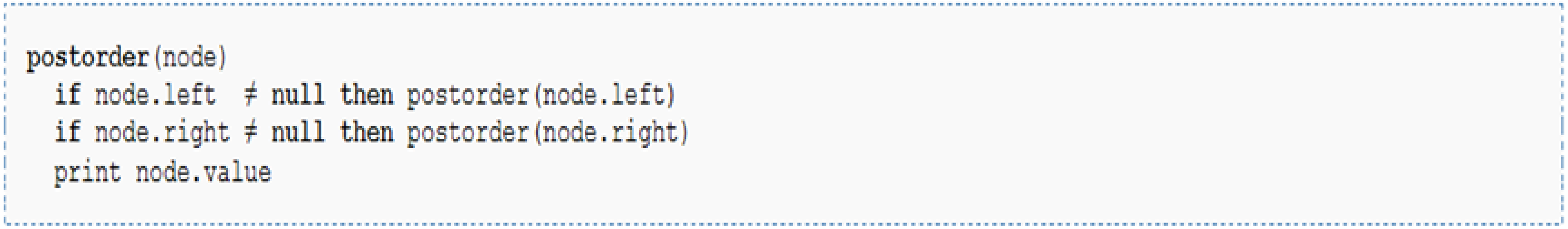
Binary Tree Traversal – pseudo code



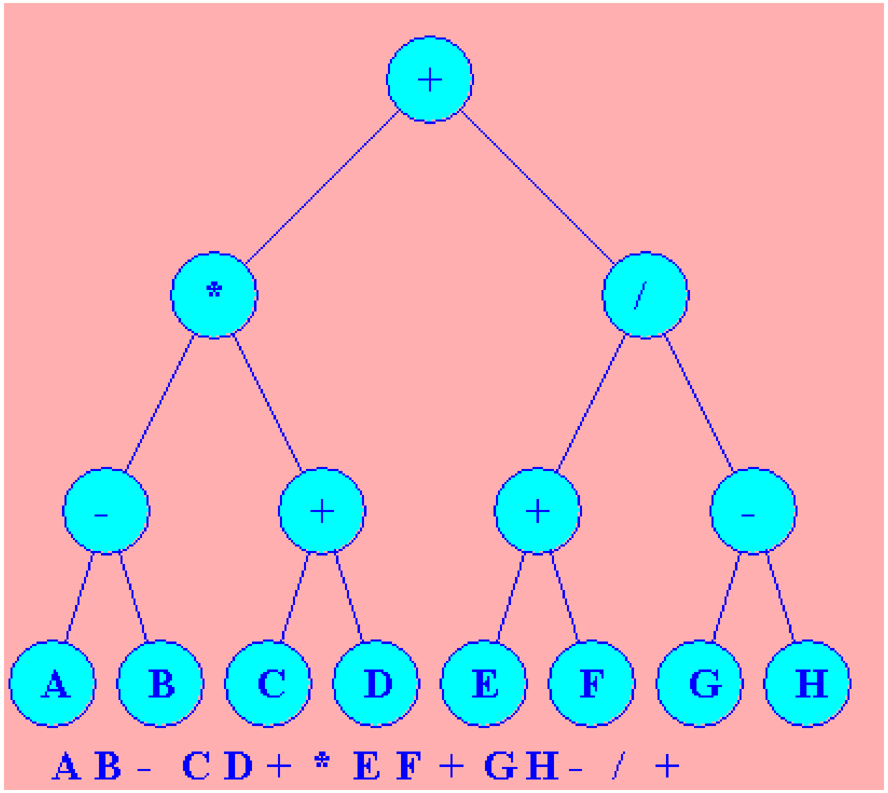
```
inorder(node)
  if node.left  ≠ null then inorder(node.left)
  print node.value
  if node.right ≠ null then inorder(node.right)
```



Binary Tree Traversal – pseudo code

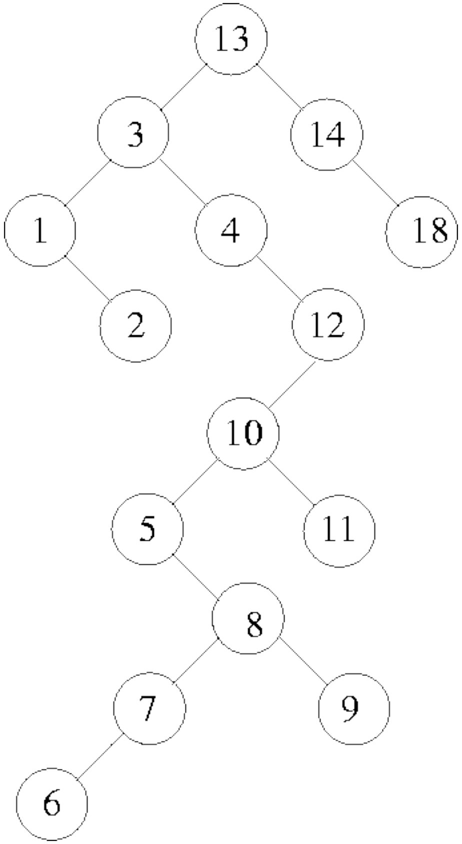


```
postorder(node)
  if node.left  ≠ null then postorder(node.left)
  if node.right ≠ null then postorder(node.right)
  print node.value
```



Binary Tree: insertion – exercise…

this tree is obtained by inserting the values:
13, 3, 4, 12, 14, 10, 5, 1, 8, 2, 7, 9, 11, 6, 18 in that order, starting from an empty tree.



Note: Check Handouts section in course website for a starter .c file to do this revision exercise

Binary Tree: search (pseudo code)

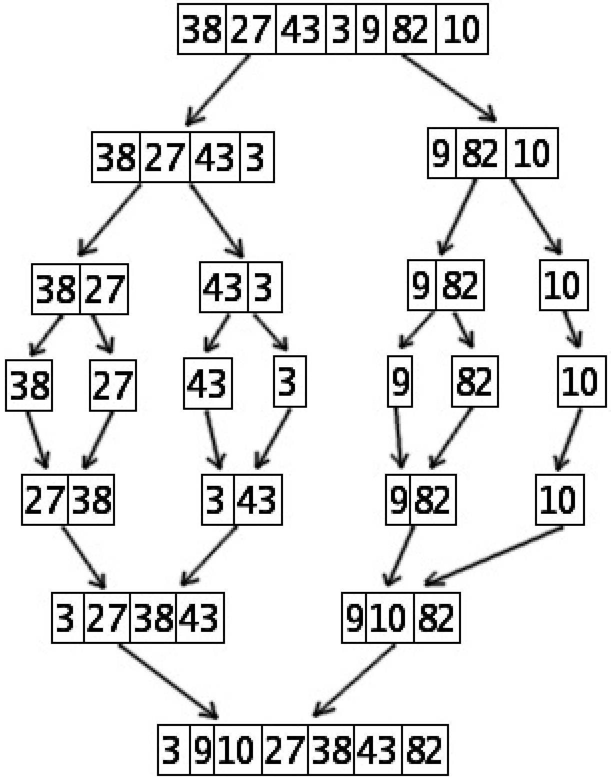
```
NodePtr searchBinaryTree(const NodePtr T, NodePtr &v, int nValue)
{
    if (v.value == nValue)
            return v;               // found!
    else
            if (v.value < nValue )
                    searchBinaryTree(T, T.ptrLeft, nValue);   // search left subtree
            else
                    searchBinaryTree(T, T.ptrRight, nValue); // search right subtree

}
```

Revision: merge sort

Merge sort trace

Conceptually, merge sort works as follows:

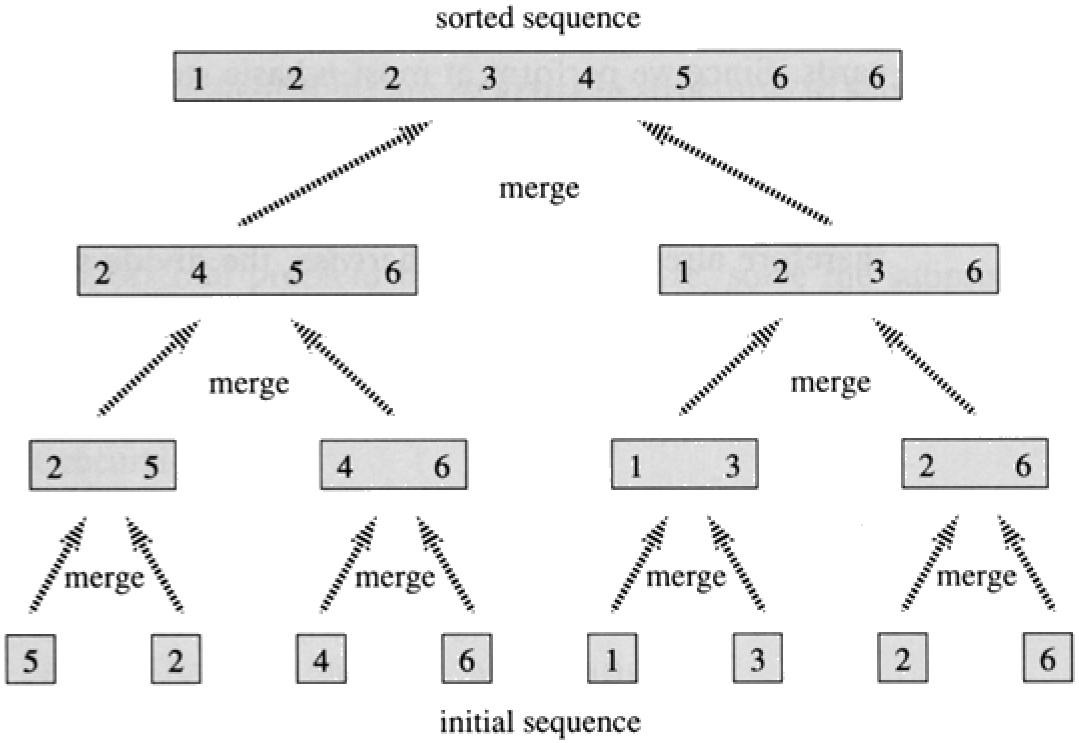
- Divide the unsorted list into two sublists of about half the size
- Divide each of the two sublists recursively until we have list sizes of length 1, in which case the list itself is returned
- Merge the two sorted sublists back into one sorted list.



Merge sort (trace #2)

Conceptually, merge sort works as follows:

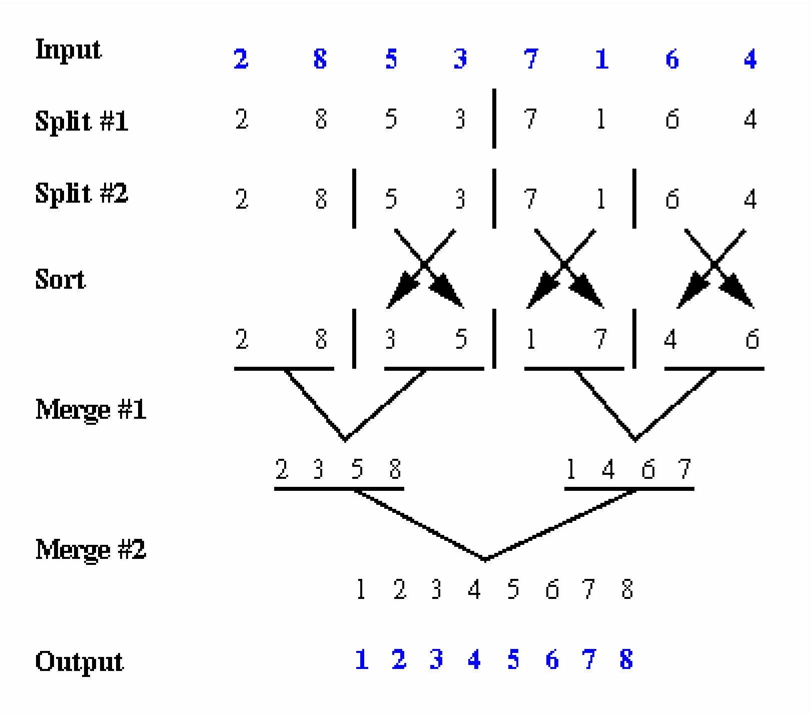
Divide the unsorted list into two sublists of about half the size
Divide each of the two sublists recursively until we have list sizes of length 1, in which case the list itself is returned
Merge the two sorted sublists back into one sorted list.



Merge sort (trace #3)

Conceptually, merge sort works as follows:

Divide the unsorted list into two sublists of about half the size
Divide each of the two sublists recursively until we have list sizes of length 1, in which case the list itself is returned
Merge the two sorted sublists back into one sorted list.



Input 2 8 5 3 7 1 6 4

Split #1 2 8 5 3 | 7 1 6 4

Split #2 2 8 | 5 3 | 7 1 | 6 4

Sort

2 8 | 3 5 | 1 7 | 4 6

Merge #1

2 3 5 8 1 4 6 7

Merge #2

1 2 3 4 5 6 7 8

Output 1 2 3 4 5 6 7 8