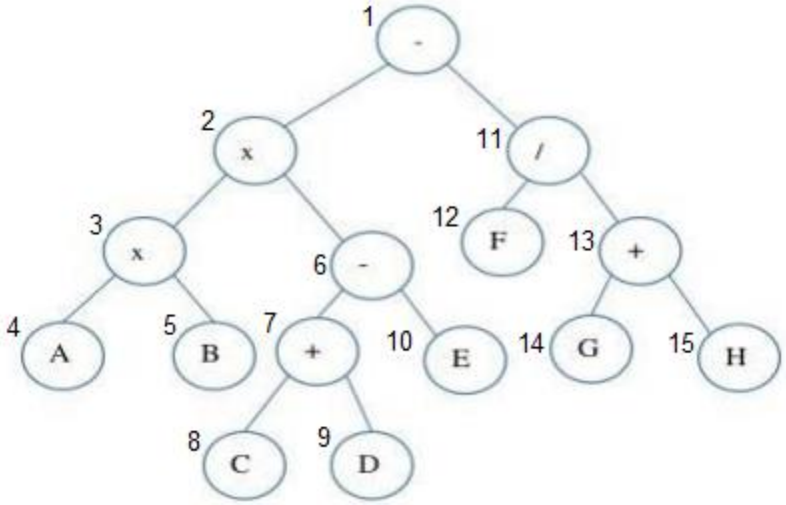
Trees
Burak Öztürk
21901841
Section 1
Assignment 2

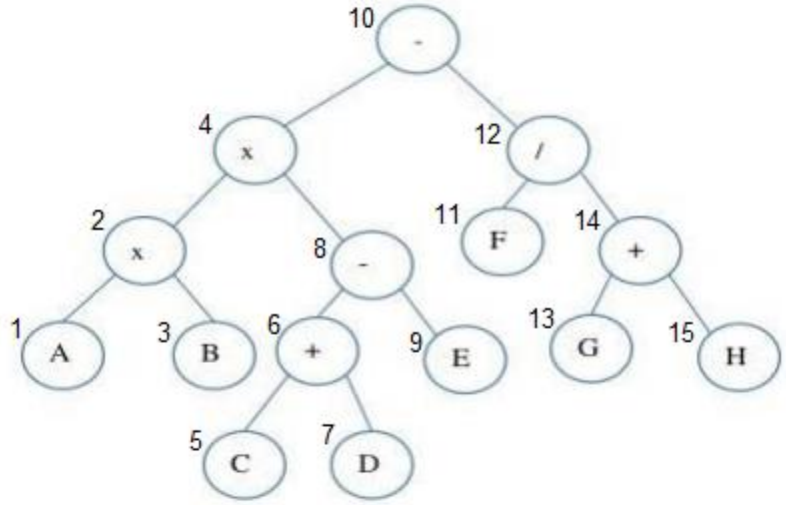
Question 1

Prefix Traversal



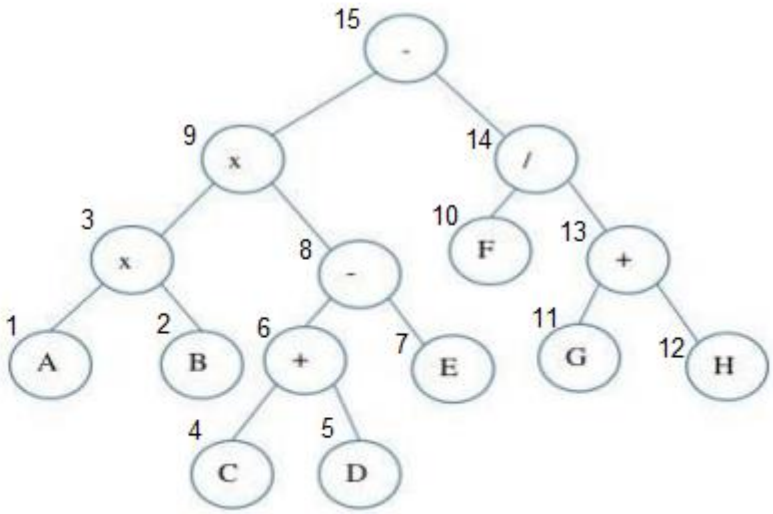
Expression: -xxAB-+CDE/F+GH

Infix Traversal



Expression: AxBxC+D-E-F/G+H

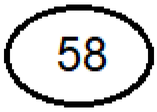
Postfix Traversal



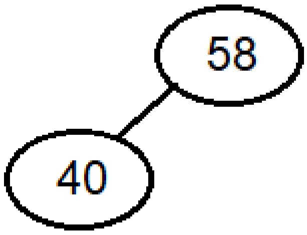
Expression: ABxCD+E-xFGH+/-

Question 2

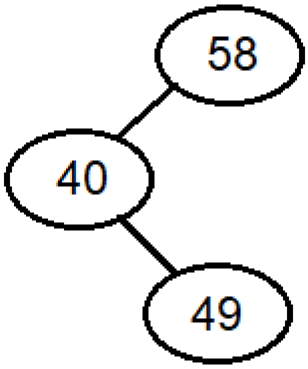
Insert 58



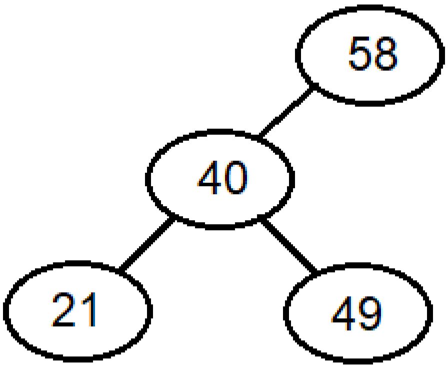
Insert 40



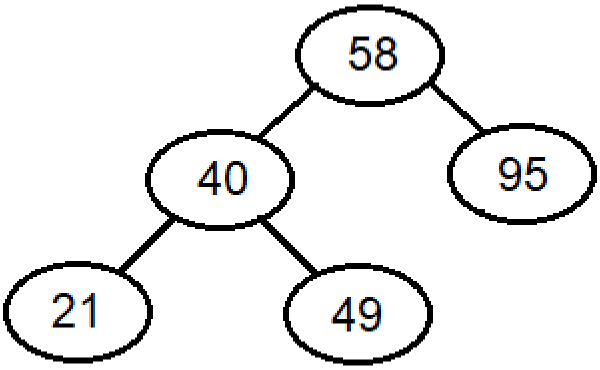
Insert 49



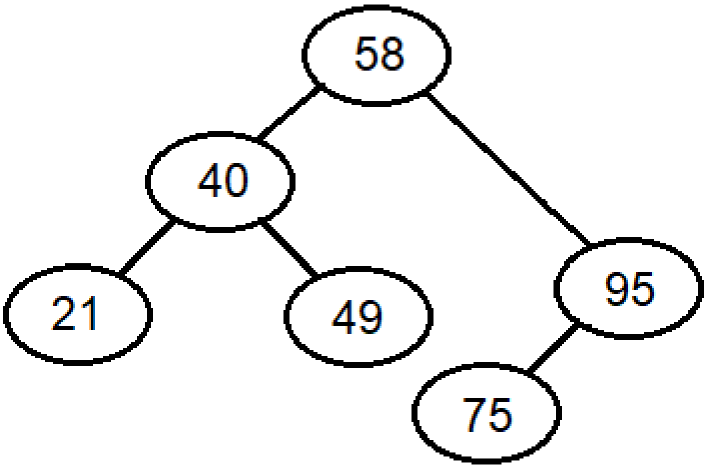
Insert 21



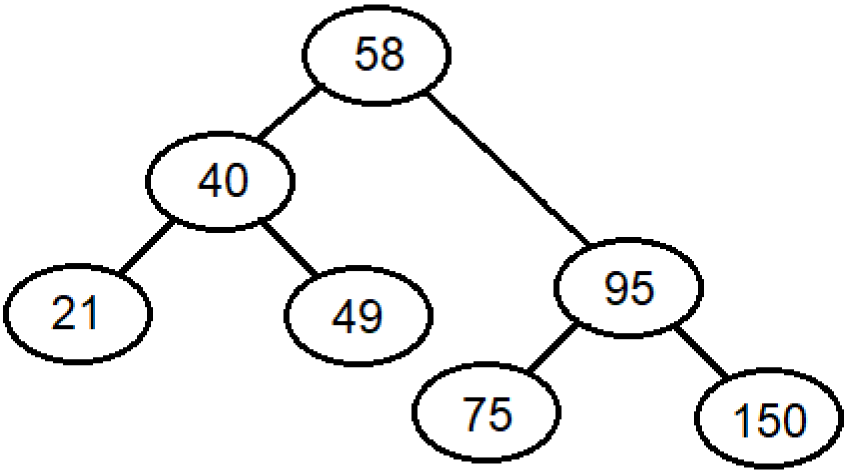
Insert 95



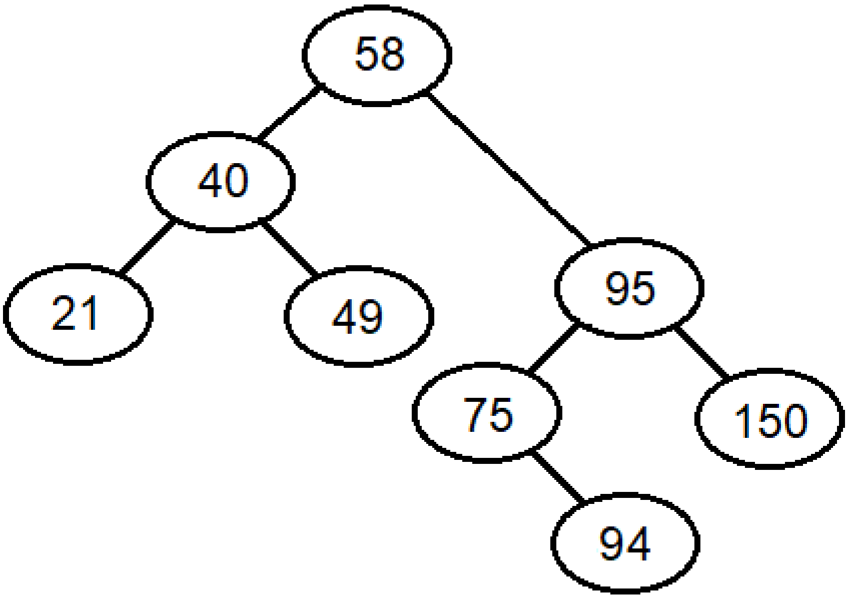
Insert 75



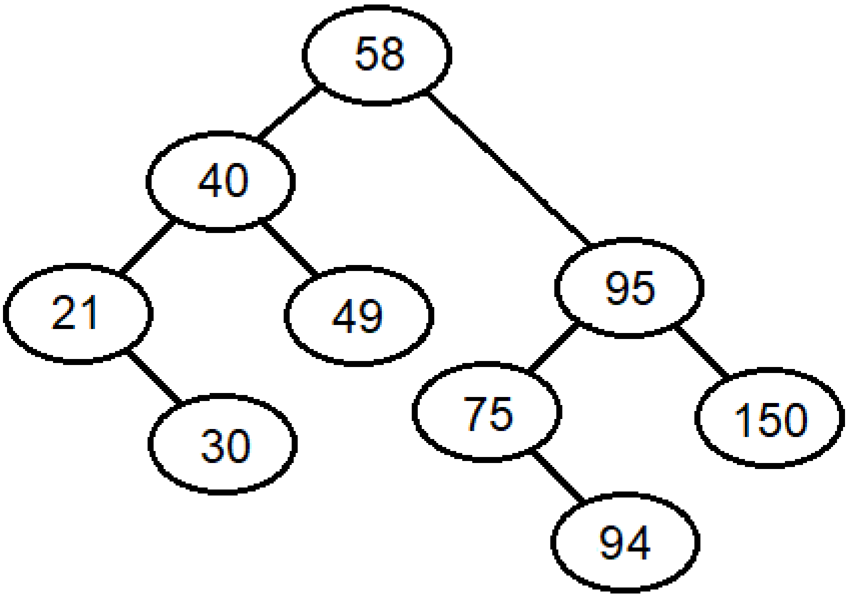
Insert 150



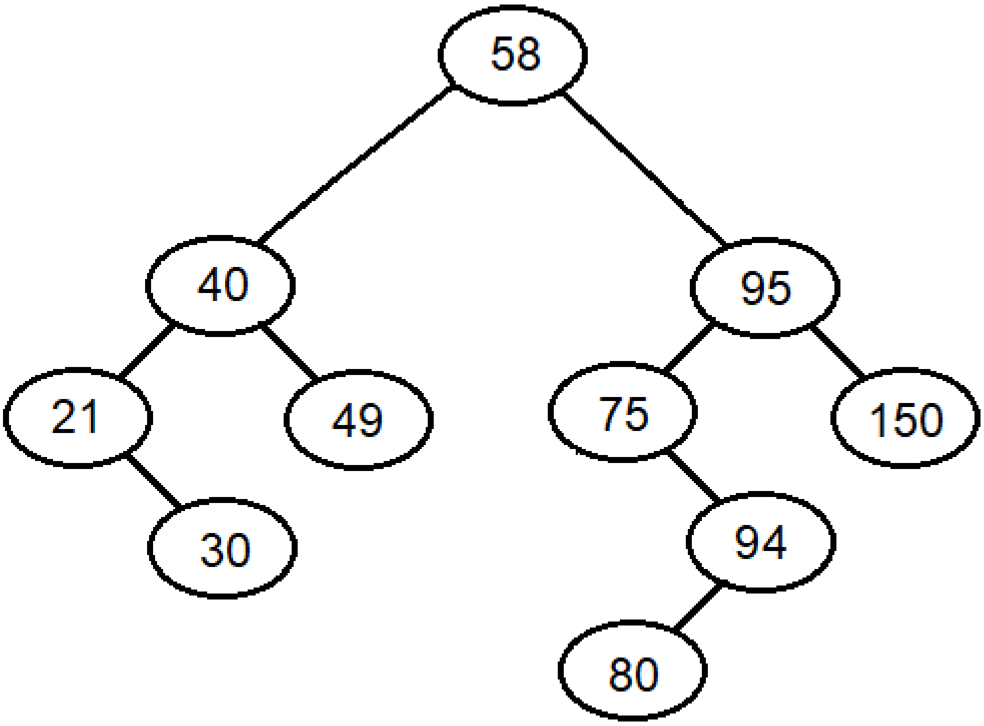
Insert 94



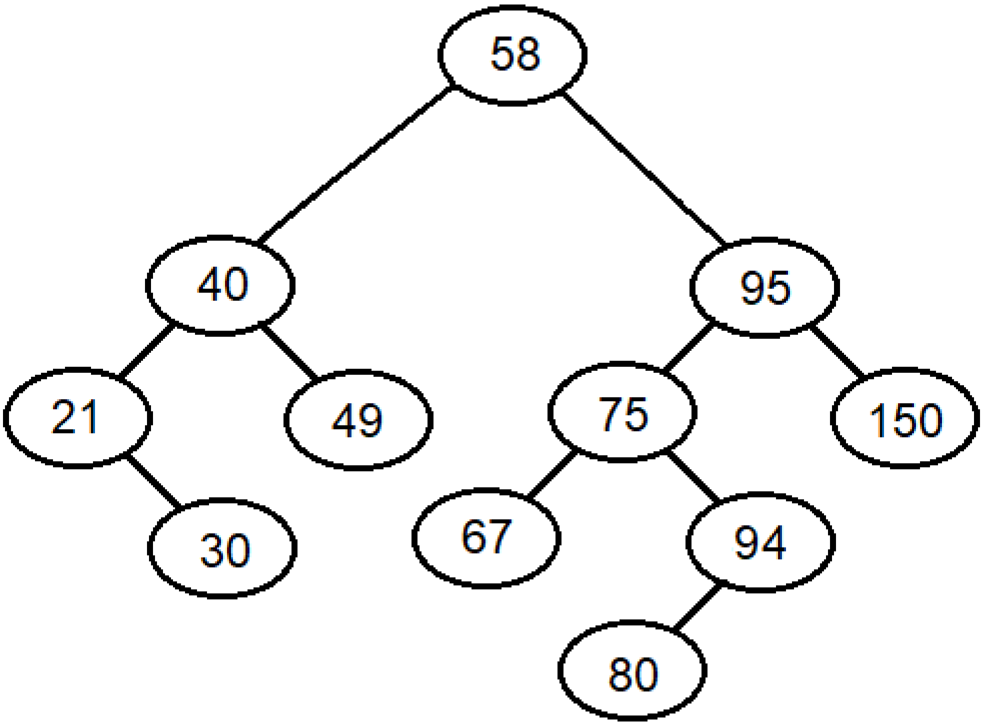
Insert 30



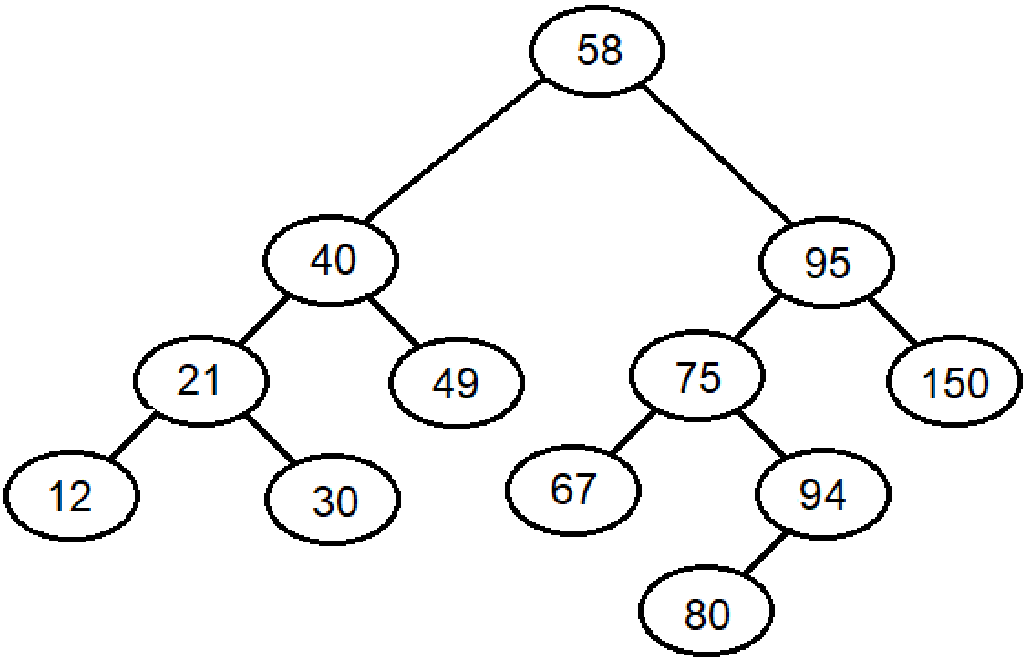
Insert 80



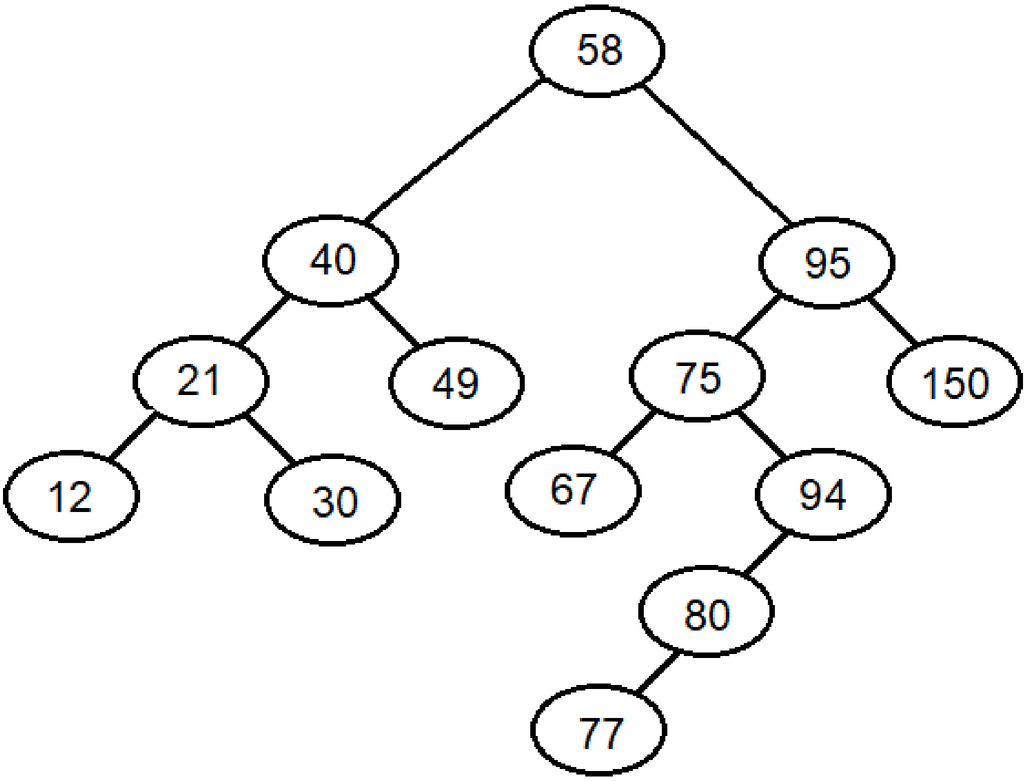
Insert 67



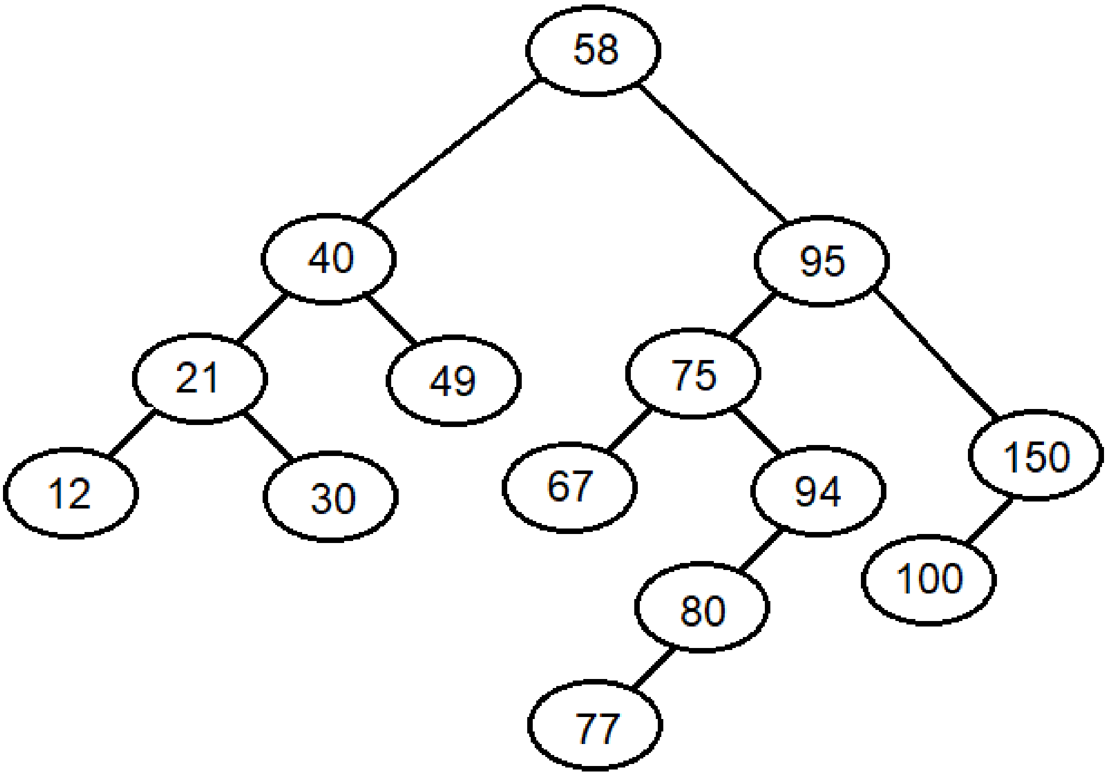
Insert 12



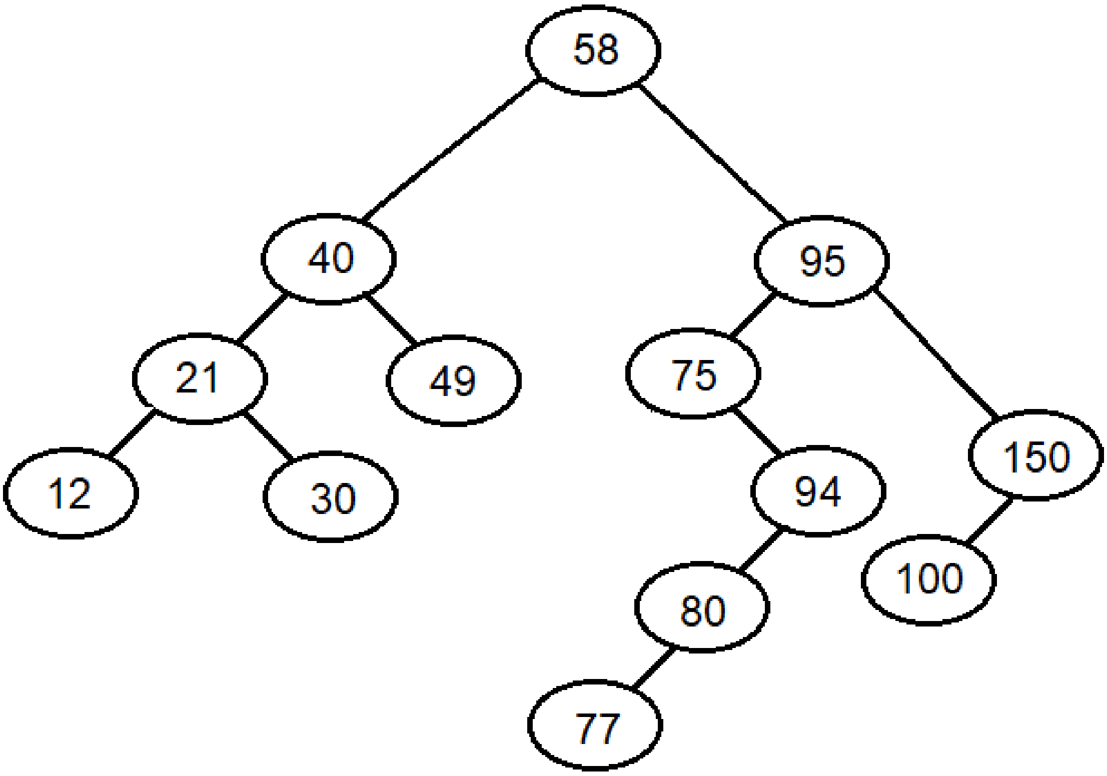
Insert 77



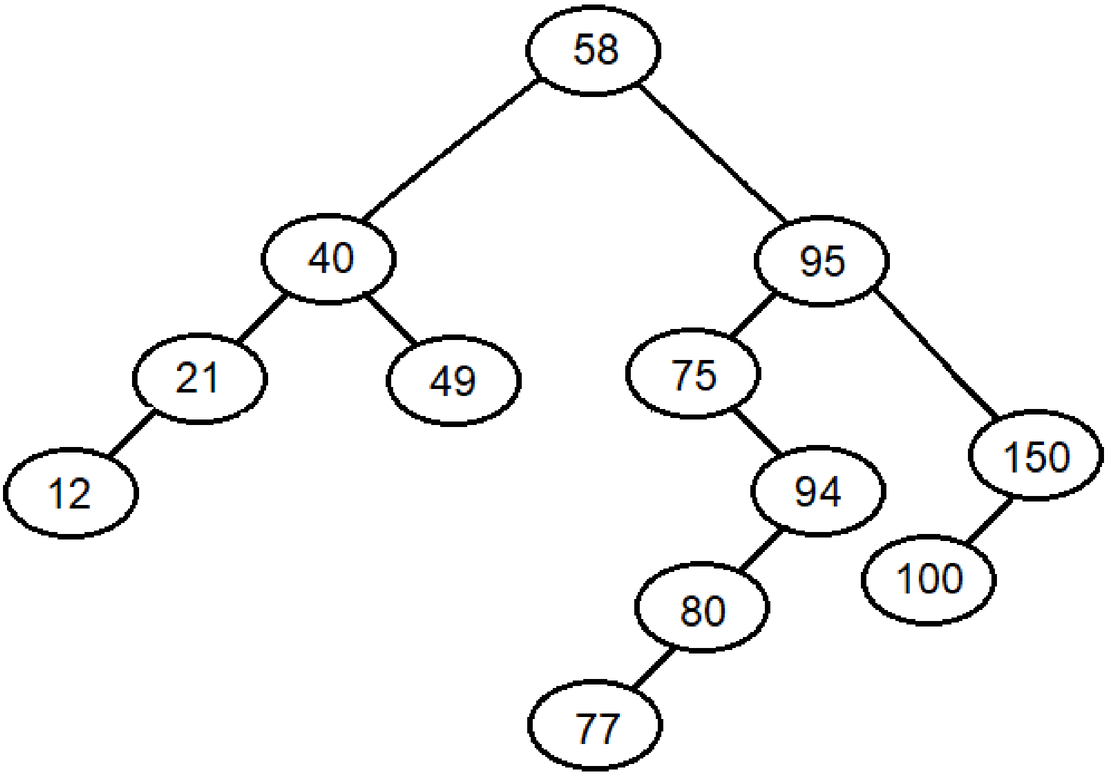
Insert 100



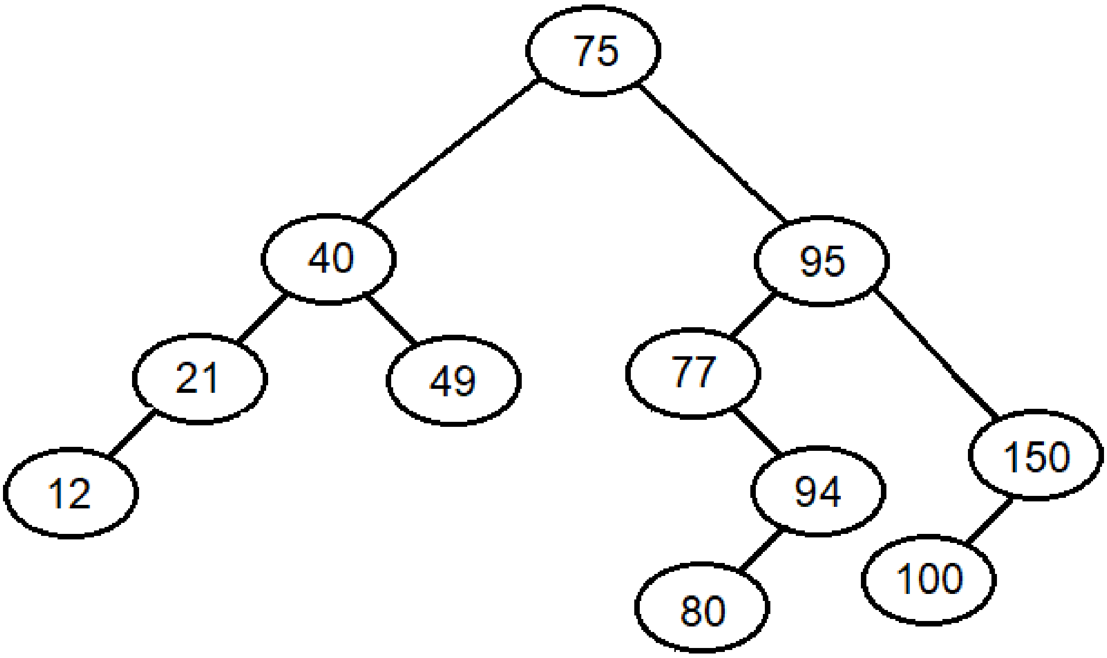
Delete 67


58
40
95
21
49
75
150
12
30
94
100
80
77

Delete 30


58
40
95
21
49
75
150
12
94
100
80
77

Delete 58



Question 4

addNgram function’s complexity

addNgram function from my implementation only calls searchTreeInsert function from BinarySearchTree class. It passes a KeyedItem instance with searchKey as ngram.

KeyedItem constructor has complexity O(n), n being length of ngram string.

searchTreeInsert function does nothing with KeyedItem instance and calls insertItem function and passes the instance directly to it. There searchTreeInsert function has complexity O(1).

insertItem function is a recursive function and it takes two arguments; a pointer to a TreeNode and a KeyedItem instance to add upon BST. Every function call has four possibilities:

1. Current node is empty. This option’s complexity is O(1) since it does a simply comparison and it calls TreeNode constructor which has O(1) complexity.
2. Current node has same searchKey with the KeyedItem instance’s searchKey. This option’s complexity is O(1) as well since it does a simply addition only.
3. Current node’s searchKey is alphabetically smaller than the KeyedItem instance’s searchKey. This option calls stringComparison function that has O(n) complexity (n being length of searchKey strings) and calls insertItem function again with current node’s left child as first argument.
4. Current node’s searchKey is alphabetically bigger than the KeyedItem instance’s searchKey. This option calls stringComparison function that has O(n) (n being length of searchKey strings) complexity and calls insertItem function again with current node’s right child as first argument.

addNgram’s worst-case happens when the string-to-add is the last string to be added to the BST and the BST has strictly one-child nodes except the only leaf node. If we tree height is L, then there will be L-1 option (3,4)’s and one option (1,2). Therefore complexity is O((L-1) * O(n)) = O(L*n), n being searchKey’s length.

operator<< function’s complexity

operator<< function from my implementation calls inorderTraverse function of BST instance that NgramTree holds and passes it’s output to ostream out.

inorderTraverse is an in-between function that calls inorderTraverse_ and returns it’s result.

inorderTraverse_ is a recursive method and works by recursively calls itself onto every child that current node has. In every case, inorderTraverse_ calls representation function of every node on the tree once. representation function has O(1) complexity and it is being n times (n being the number of nodes on the tree), therefore operator<< function has complexity O(n).

Sample Output:

Tester Main:

```
#include "NgramTree.h"
int main( int argc, char **argv ) {
    NgramTree tree;

    string fileName( argv[1] );
    int n = atoi( argv[2] );

    tree.generateTree( fileName, n );

    cout << "\nTotal " << n << "-gram count: " << tree.getTotalNgramCount() <<
endl;
    cout << tree << endl;

    cout << n << "-gram tree is complete: " << (tree.isComplete() ? "Yes" : "No")
<< endl;

    // Before insertion of new n-grams
    cout << "\nTotal " << n << "-gram count: " << tree.getTotalNgramCount() <<
endl;

    tree.addNgram( "samp" );
    tree.addNgram( "samp" );
    tree.addNgram( "zinc" );
    tree.addNgram( "aatt" );

    cout << "\nTotal " << n << "-gram count: " << tree.getTotalNgramCount() <<
endl;
    cout << tree << endl;
    cout << n << "-gram tree is complete: " << (tree.isComplete() ? "Yes" : "No")
<< endl;
    cout << n << "-gram tree is full: " << (tree.isFull() ? "Yes" : "No") << endl;

    return 0;
}
```

testFile.txt

```
this is sample text and thise is all
```

Output:

```
Total 4-gram count: 7
"ampl" appears 1 time(s)
"hise" appears 1 time(s)
"mple" appears 1 time(s)
"samp" appears 1 time(s)
"text" appears 1 time(s)
"this" appears 2 time(s)

4-gram tree is complete: No

Total 4-gram count: 7

Total 4-gram count: 11
"aatt" appears 1 time(s)
"ampl" appears 1 time(s)
"hise" appears 1 time(s)
"mple" appears 1 time(s)
"samp" appears 3 time(s)
"text" appears 1 time(s)
"this" appears 2 time(s)
"zinc" appears 1 time(s)

4-gram tree is complete: No
4-gram tree is full: No
```