CS 202 – Homework 02

Title:Trees

Ali Eren Günaltılı

21801897

Assignment 2

PDF File for Question 1,2 and 4

10/11/2021

Question 1->

Prefix: - * * A B - + C D E / F + G H

Postfix: A B * C D + E - * F G H + / -

Infix: A*B*C+D-E-F/G+H

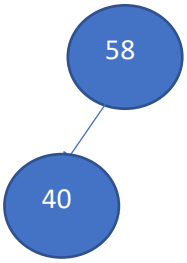
((A * B) * (C + D – E)) – (F / (G + H))

Question 2->

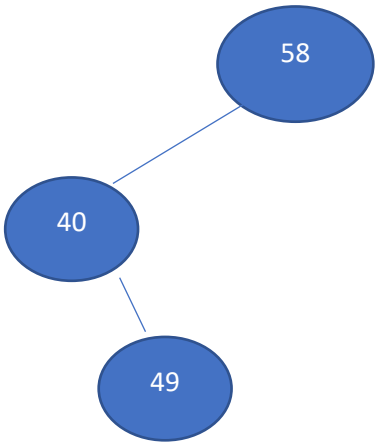
Binary Search Tree Implementation (bigger belongs to right side, smaller belongs to left side)

- **Inserting 58**

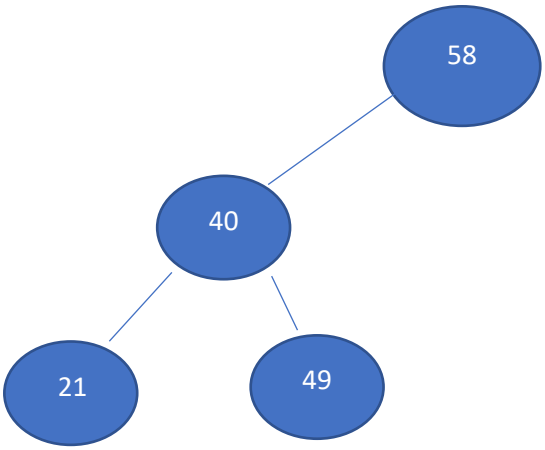
- **Inserting 40 (40 < 58 so add it to left**



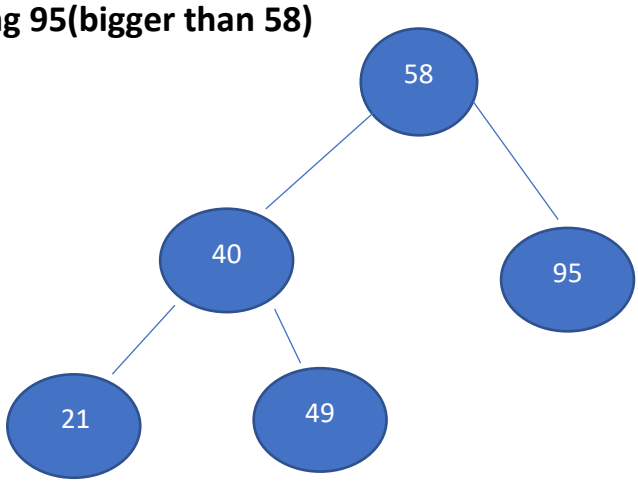
Inserting 49



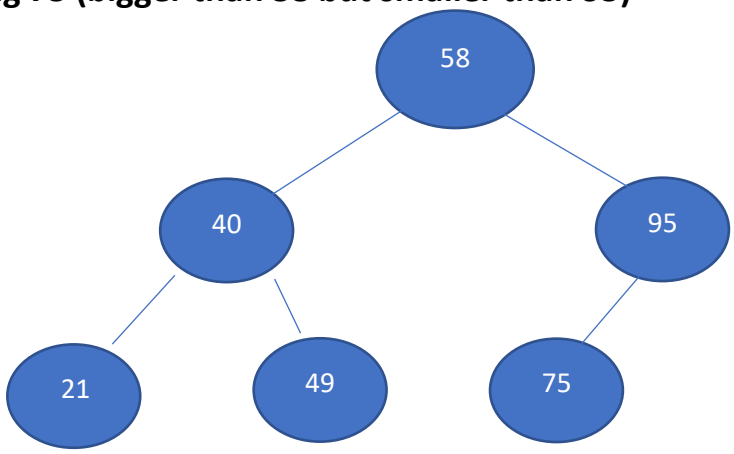
Inserting 21 (smaller than 58 and 40)



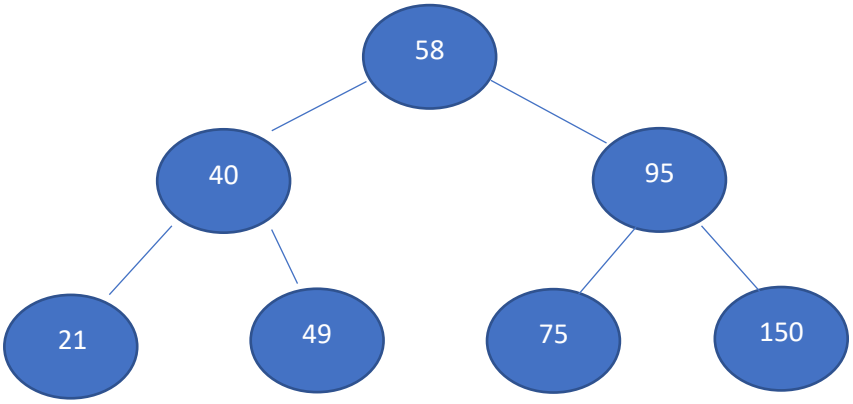
Inserting 95(bigger than 58)



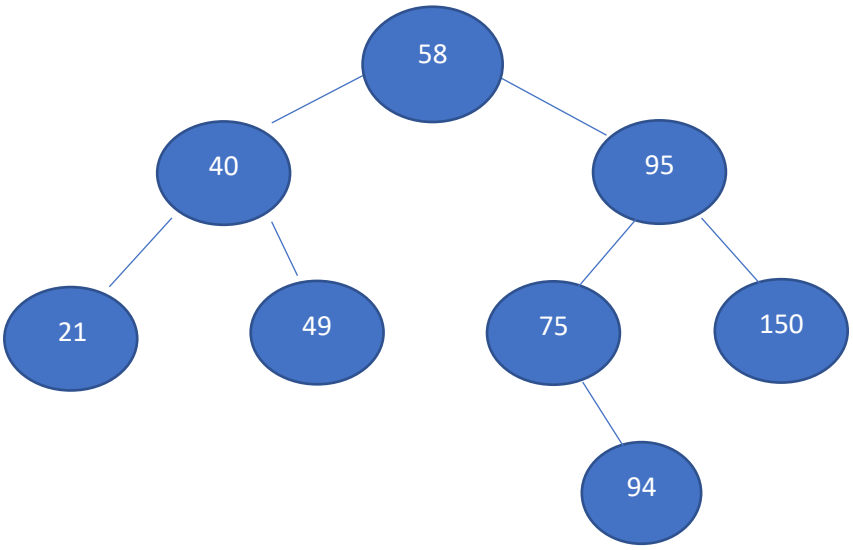
Inserting 75 (bigger than 58 but smaller than 95)



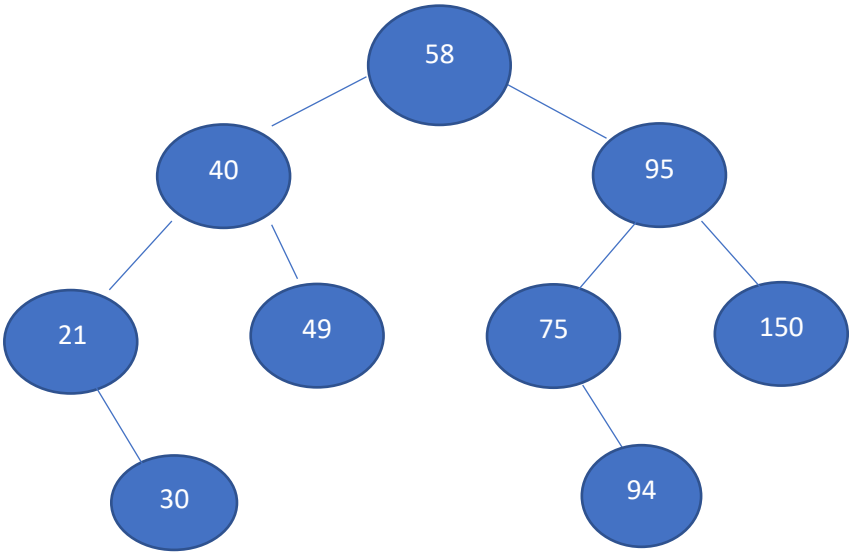
Inserting 150 (bigger than 58 and 95)



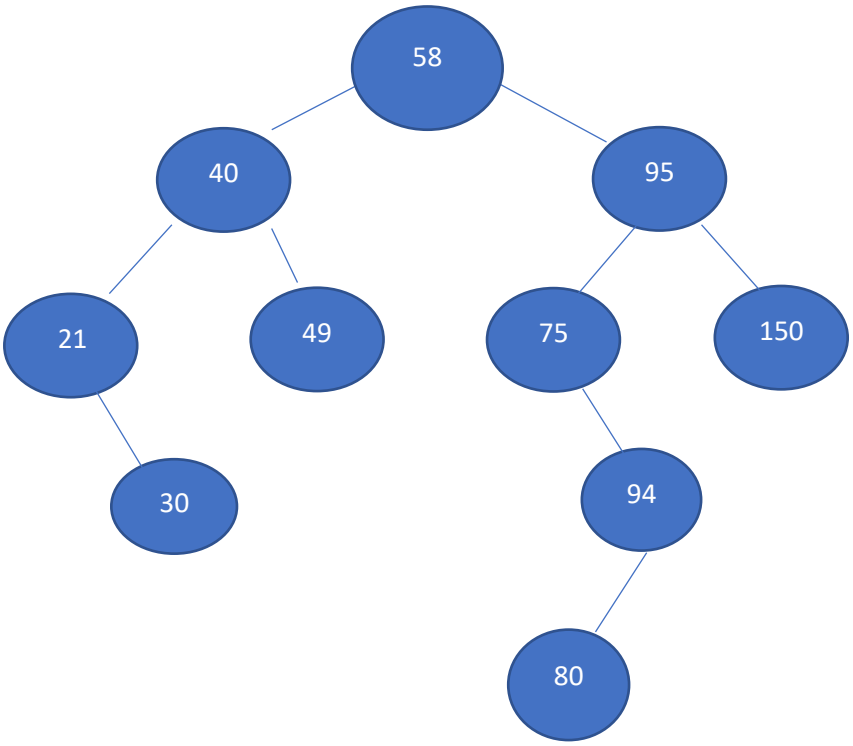
Inserting 94 (bigger than 58, smaller than 95 but bigger than 75 so it should be left child of 75)



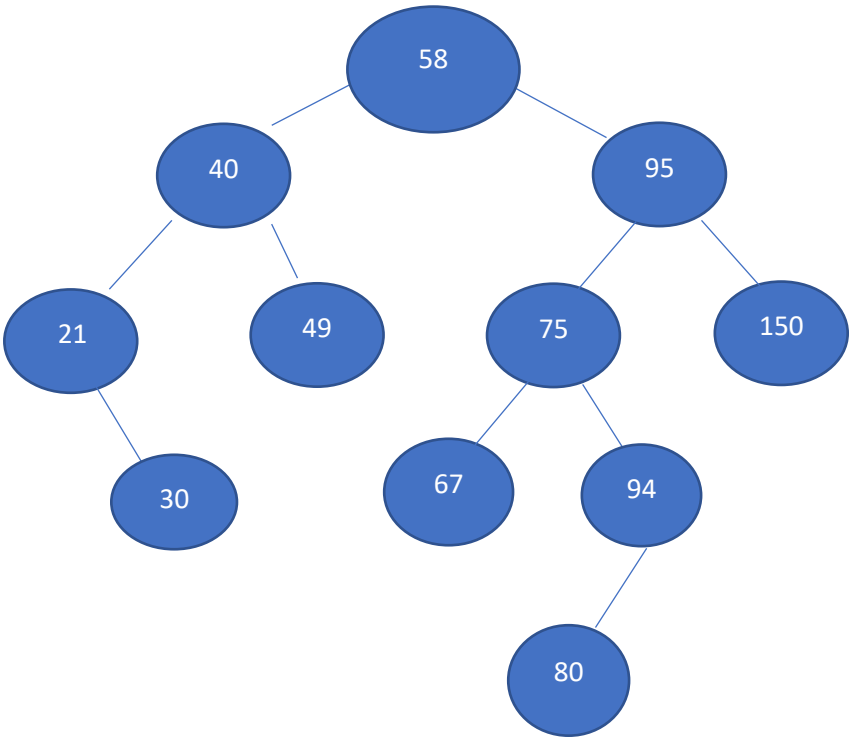
Inserting 30 (smaller than 58 and 40, but bigger than 21)



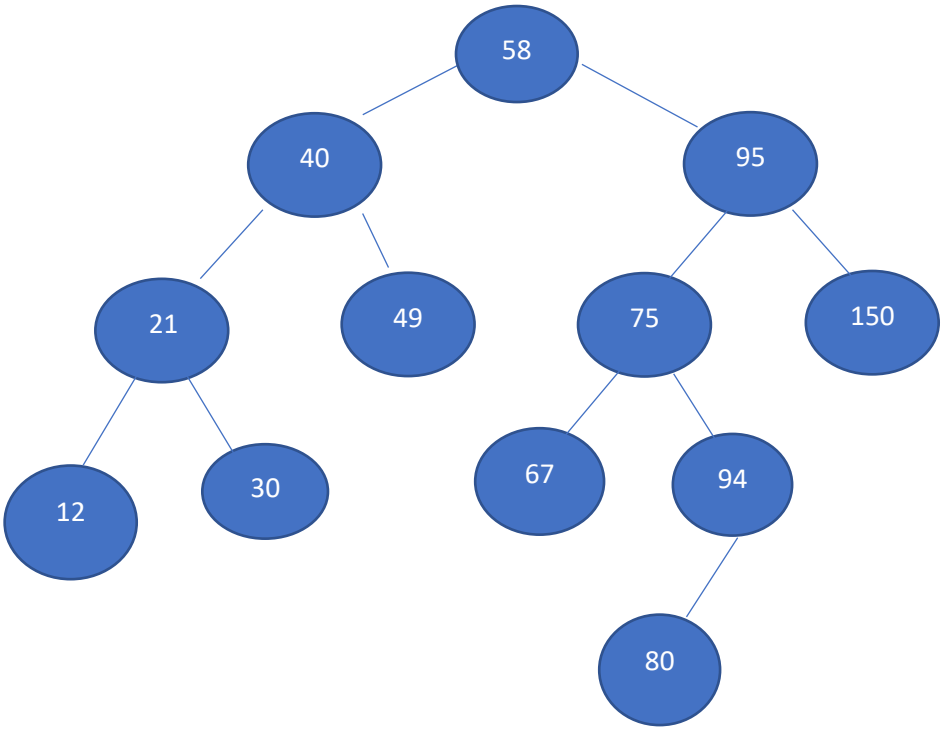
Inserting 80(bigger than 58, smaller than 95, bigger than 75, smaller than 94)



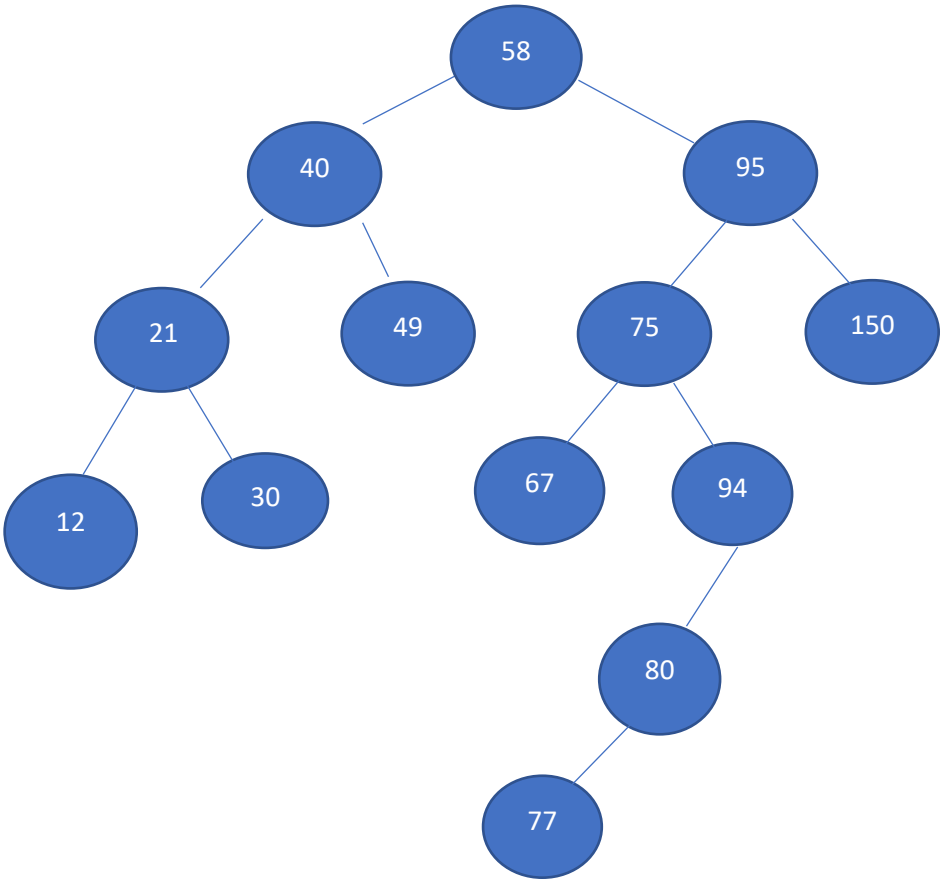
Inserting 67 (bigger than 58, smaller than 95, smaller than 75)



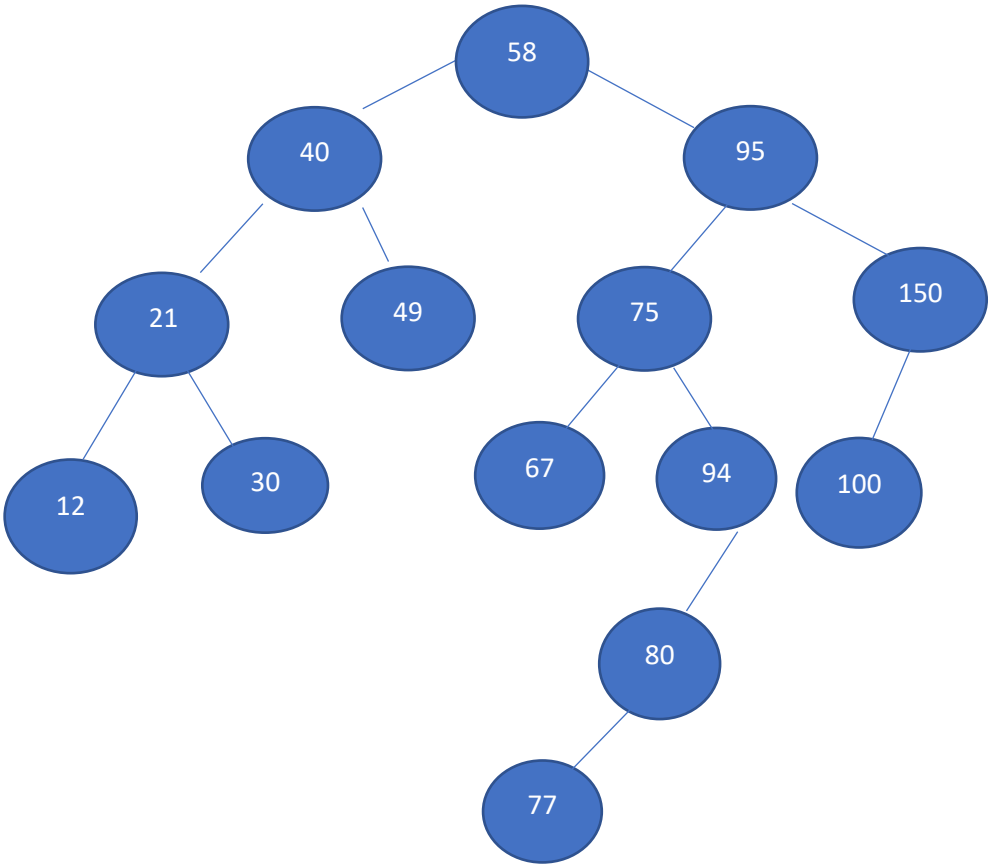
Inserting 12(smaller than 58, 40 and 21)



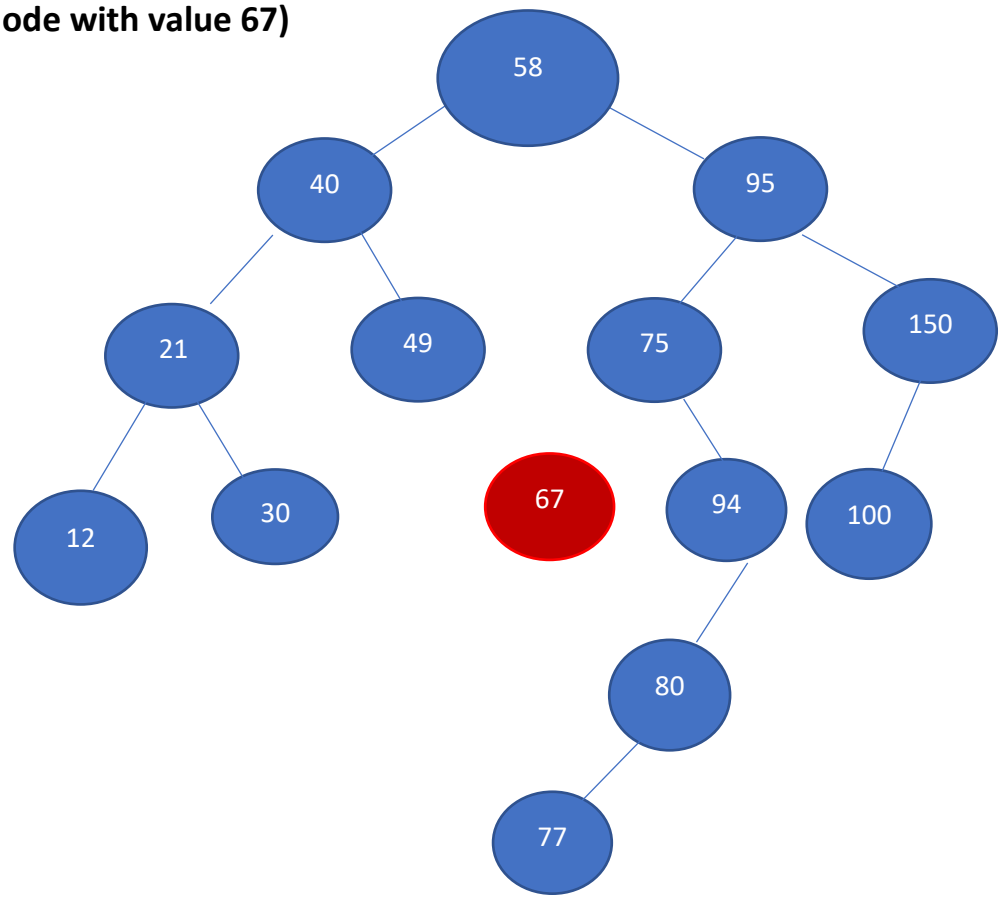
Inserting 77 (bigger than 58, smaller than 95, bigger than 75, smaller than 94 and 80)



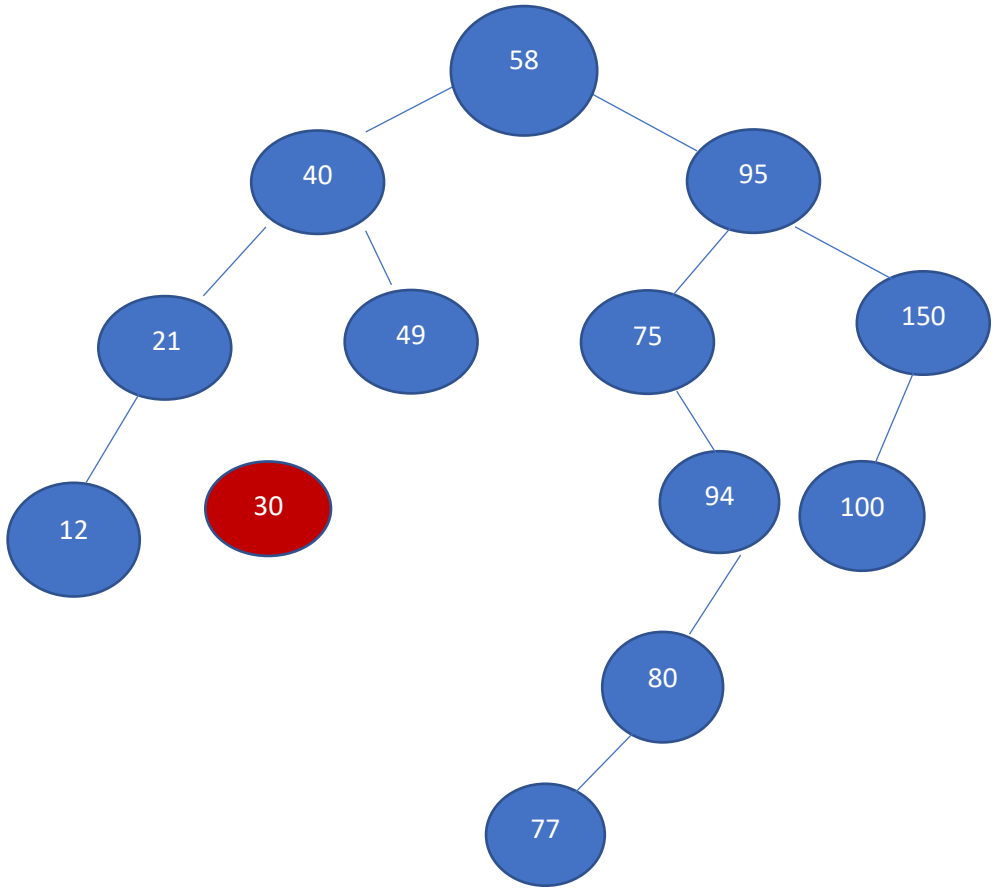
Inserting 100 (bigger than 58, bigger than 95 , smaller than 150)



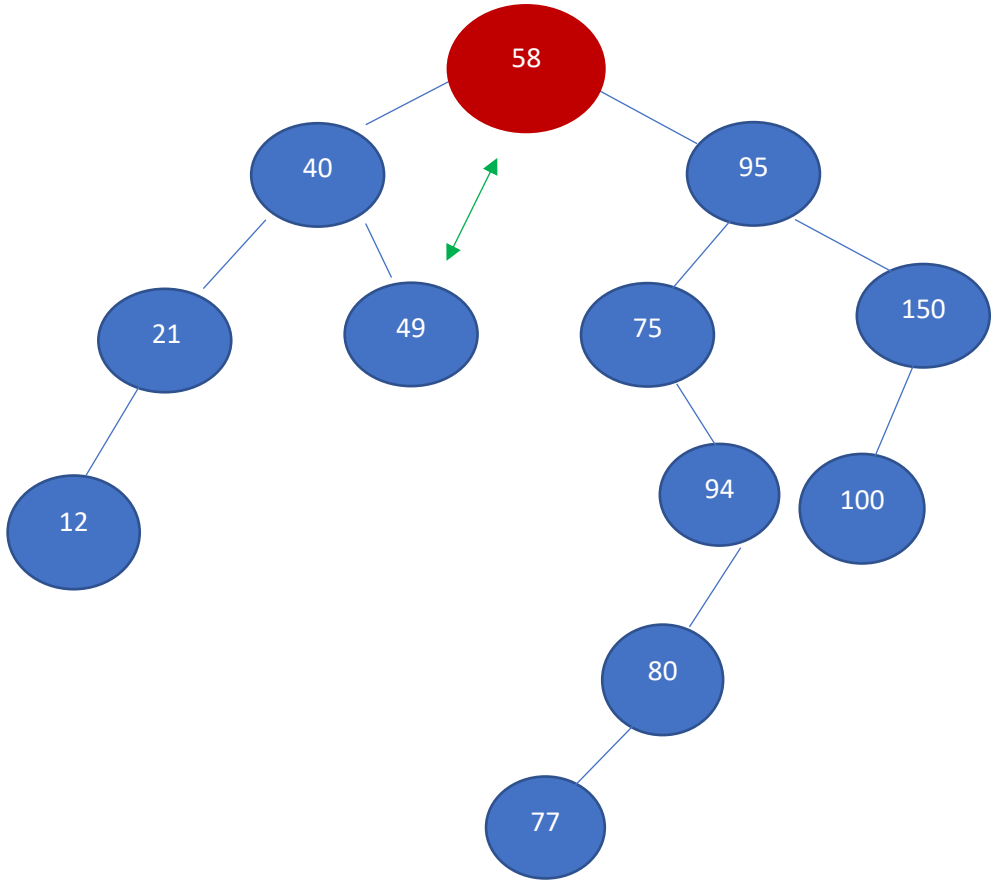
Deleting 67 (finding 67 and delete the connection between 75. Then delete node with value 67)



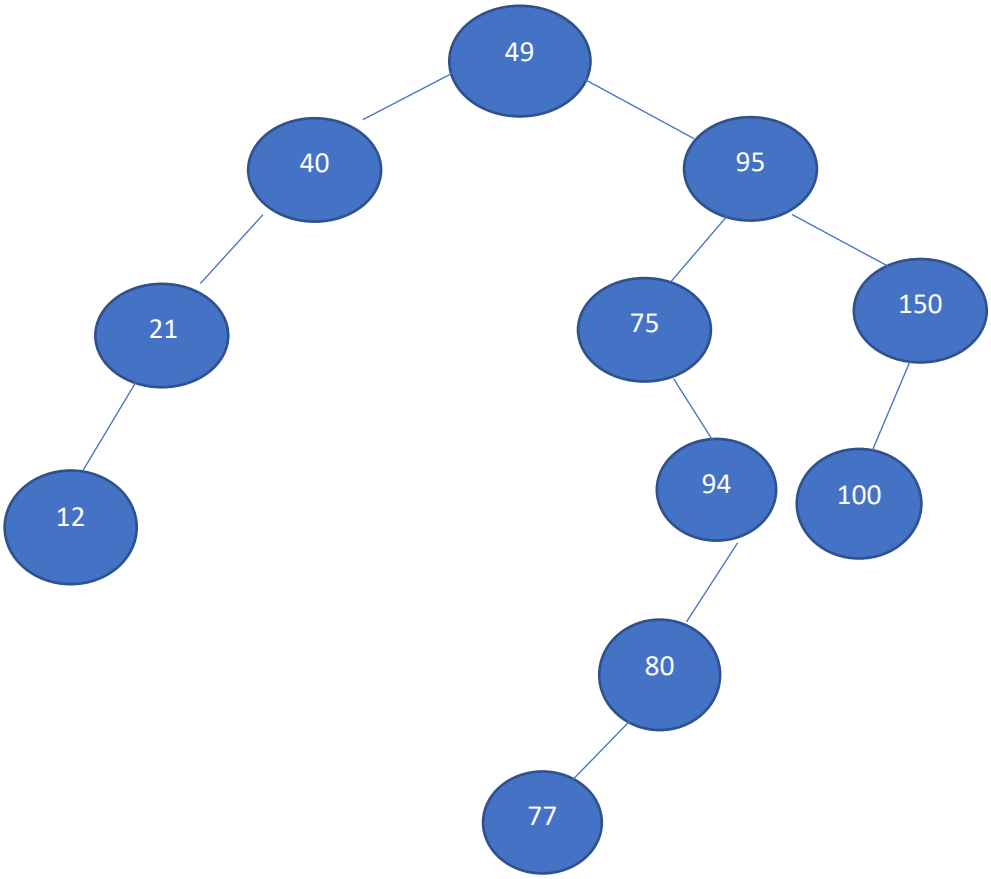
Deleting 30 (finding 30, 30 is a leaf so only thing to do is break the connection between its parent)



Deleting 58 (58 is the root so when we before delete it to not lost connections of the tree, we have to put other node as a root. We choose the root value as a biggest value of a left sub tree of our tree. New choosen root is node with value 49 as it is the biggest one in left subtree. Make 49's left and right same as 58 before deletion. After we saved the connections 58 can be deleted safely.



Last version of our BST



Question 4 and Execution of my program on Dijkstra

```
[eren.gunaltili@dijkstra CS202HW2]$ g++ CS202_Hw02.cpp NgramTree.cpp -o fun
[eren.gunaltili@dijkstra CS202HW2]$ ./fun te.txt 4

Total 4-gram count: 6
"ampl" appers 1 time(s)
"hise" appers 1 time(s)
"mple" appers 1 time(s)
"samp" appers 1 time(s)
"text" appers 1 time(s)
"this" appers 2 time(s)

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

Total 4-gram count: 6

Total 4-gram count: 8
"aatt" appers 1 time(s)
"ampl" appers 1 time(s)
"hise" appers 1 time(s)
"mple" appers 1 time(s)
"samp" appers 3 time(s)
"text" appers 1 time(s)
"this" appers 2 time(s)
"zinc" appers 1 time(s)

4-gram tree is complete: No
4-gram tree is full: No
---------------------------
[eren.gunaltili@dijkstra CS202HW2]$
```

```
==14347==
==14347== HEAP SUMMARY:
==14347==     in use at exit: 0 bytes in 0 blocks
==14347==   total heap usage: 42 allocs, 42 frees, 10,168 bytes allocated
==14347==
==14347== All heap blocks were freed -- no leaks are possible
==14347==
==14347== Use --track-origins=yes to see where uninitialised values come from
==14347== For lists of detected and suppressed errors, rerun with: -s
==14347== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

My addNGram

```
//add nGram method is also use createTree method as cr
void NgramTree::addNgram(string ngram) {
    createTree(ngram, getN());
}
```

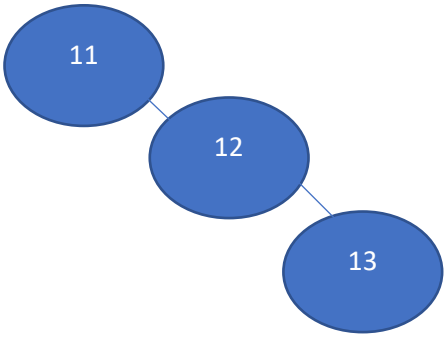
```
//Then puts it into right place.
void NgramTree::createTree(string str, int n) {
    if (str.length() >= n) {
        string tmp;
        bool flag = false;
        int k = 0;
        if (root == NULL) {
            root = new TreeNode(str.substr(0, n), NULL, NULL);
            root->count++;
            for (int i = 1; i <= str.length() - n; i++) {
                tmp = str.substr(i, n);
                flag = isAlreadyThere(root, tmp);
                if (!flag) {
                    insert(root, tmp);
                }
            }
        }
        else {
            TreeNode* head = root;
            for (int i = 0; i <= str.length() - n; i++) {
                tmp = str.substr(i, n);
                flag = isAlreadyThere(root, tmp);
                if (!flag) {
                    insert(head, tmp);
                }
            }
        }
    }
}
```

```
205
206 bool NgramTree::isAlreadyThere(TreeNode*& ptr, string str) {
207     if (ptr != NULL) {
208         if (ptr->item == str) {
209             ptr->count++;
210             return true;
211         }
212         else if (str < ptr->item) {
213             isAlreadyThere(ptr->leftChildPtr, str);
214         }
215         else {
216             isAlreadyThere(ptr->rightChildPtr, str);
217         }
218     }
219     return false;
220 }
221
```

```
void NgramTree::insert(TreeNode* ptr, string str) {
    TreeNode* tmp = NULL;
    TreeNode* r = NULL;
    while (ptr != NULL) {
        r = ptr;
        if (ptr->item == str) {
            return;
        }
        else {
            if (str > ptr->item) {
                ptr = ptr->rightChildPtr;
            }
            else {
                ptr = ptr->leftChildPtr;
            }
        }
    }
    tmp = new TreeNode(str, NULL, NULL);
    tmp->count++;
    if (tmp->item < r->item) {
        r->leftChildPtr = tmp;
        if (r->rightChildPtr == NULL) {
            internal++;
        }
    }
    else {
        r->rightChildPtr = tmp;
        if (r->leftChildPtr == NULL) {
            internal++;
        }
    }
}
```

My addNGram automatically calls createTree function. Inside the createTree function we directly go inside the for loop as we are using addNGram to add a word with size n. As the size is precise we visit the for loop once. Thus, we can consider it like a if statement with a complexity O(1). Main two functions that create complexity are isAlreadyThere and insert. isAlreadyThere pass the whole tree to check whether the string is there or not. It’s worst case scenerio

happens when the height equals to size. Thus it's wors case is O(N). Insertion is also similar, it is worst scenario happens when the height become equals to size.



We can give worst case scenario example from this Binary Search Tree. To find 13 we have to travel 3 times which is proportional to size of a tree. To insert 14 we also have to traverse 3 times just like searching.

Worst Case of addNGram = O(N) + O(N) = O(N). We only have to consider insert and isAlready there since other sentences have O(1) complexity.

Operator Overload <<operator

```
//overloading operator calls destructor directly. To eliminate that
ostream& operator<<(ostream& out, NgramTree tree) {
    return out;
}
```

```
//I use myIn traversal print method inside of my copy constr
NgramTree::NgramTree(const NgramTree& t) {
    copyItem(t.root, this->root);
    myin(this->root);
}
```

```
39      }
40
41
42      //copying every item of TreeNode to another
43      void NgramTree::copyItem(TreeNode* ptr, TreeNode*& newPtr) {
44          if (ptr != NULL) {
45              newPtr = new TreeNode(ptr->item, NULL, NULL);
46              newPtr->count = ptr->count;
47              newPtr->index = ptr->index;
48              copyItem(ptr->leftChildPtr, newPtr->leftChildPtr);
49              copyItem(ptr->rightChildPtr, newPtr->rightChildPtr);
50          }
51      }
```

```
86
87  void NgramTree::myin(TreeNode* ptr) {
88      if (ptr != NULL) {
89          myin(ptr->leftChildPtr);
90          cout << "\"" << ptr->item << "\"" << " appers " << ptr->count << " time(s) " << endl;
91          myin(ptr->rightChildPtr);
92      }
93  }
94
```

My operator overload function automatically calls the copy constructor as it tries to reach an reference of an object. In my copy constructor I create one copy of my tree to save real one. My copy constructor has a worst time complexity O(N). My real print function myIn has also worst case scenario when height become equals to size. It is also O(N).

Thus time complexity of <<operator = O(N) + O(N) = O(N).