CS 202

Fundamental Structures of Computer Science II

FALL 2021-2022

HOMEWORK 2

NAME: Ferhat

SURNAME: Korkmaz

ID: 21901940

SECTION: 01

ASSIGNMENT: Binary Search Trees

Question 1-)

- If we traverse the tree by using preorder traversal, we obtain the following prefix expression

  -xxAB-+CDE/F+GH

- If we traverse the tree by using inorder traversal, we obtain the following infix expression

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

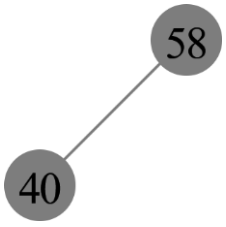
- If we traverse the tree by using postorder traversal, we obtain the following postfix expression

  ABxCD+E-xFGH+/-

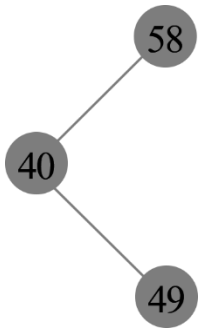
Question 2-)

An empty BST

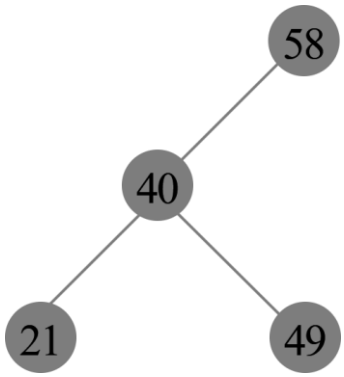
Inserting 58 to the root.



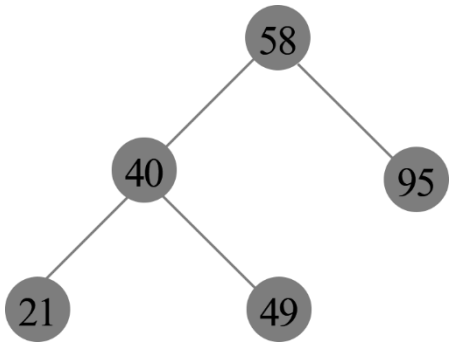
Inserting 40 to the left child of 58.



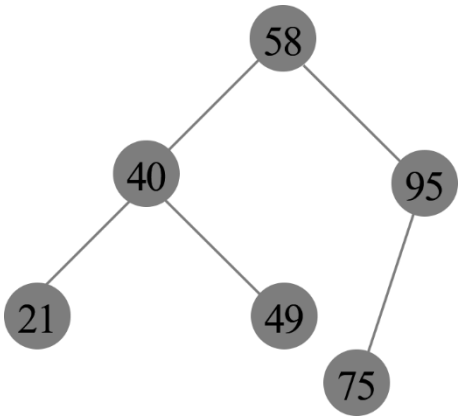
Inserting 49 to the right child of 40.



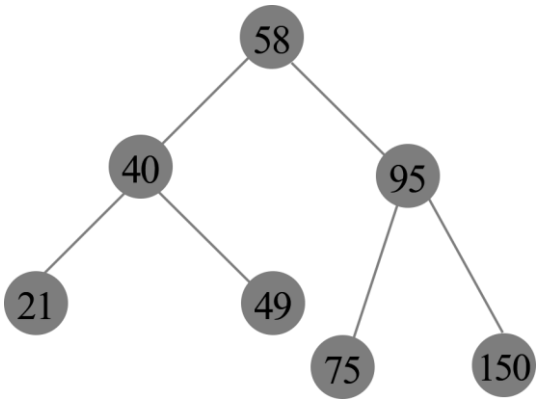
Inserting 21 to the left child of 40.



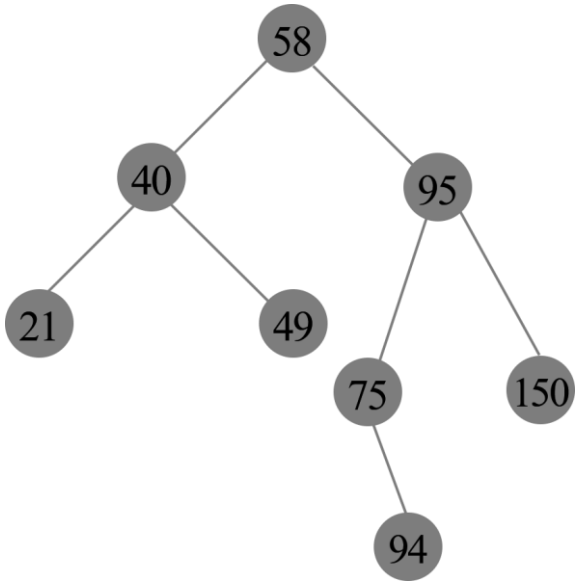
Inserting 95 to the right child of 58.



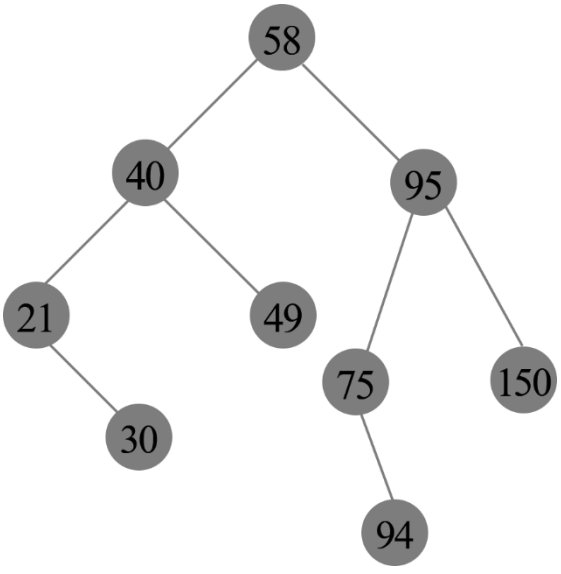
Inserting 75 to the left child of 95.



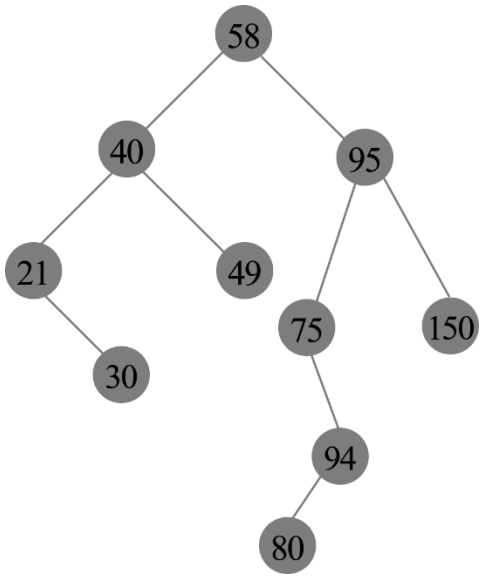
Inserting 150 to the right child of 95.



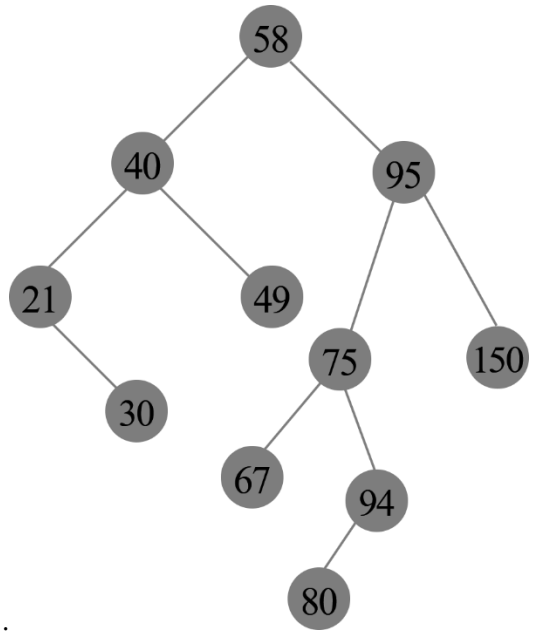
Inserting 94 to the right child of 75.



Inserting 30 to the right child of 21.

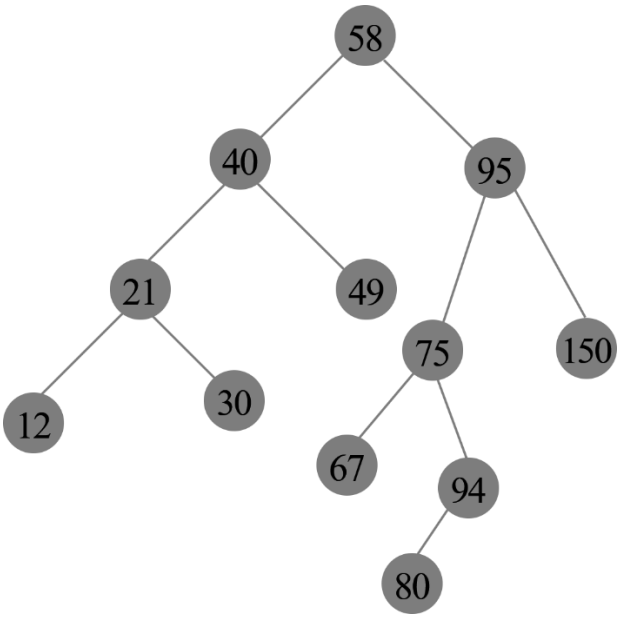


Inserting 80 to the left child of 94

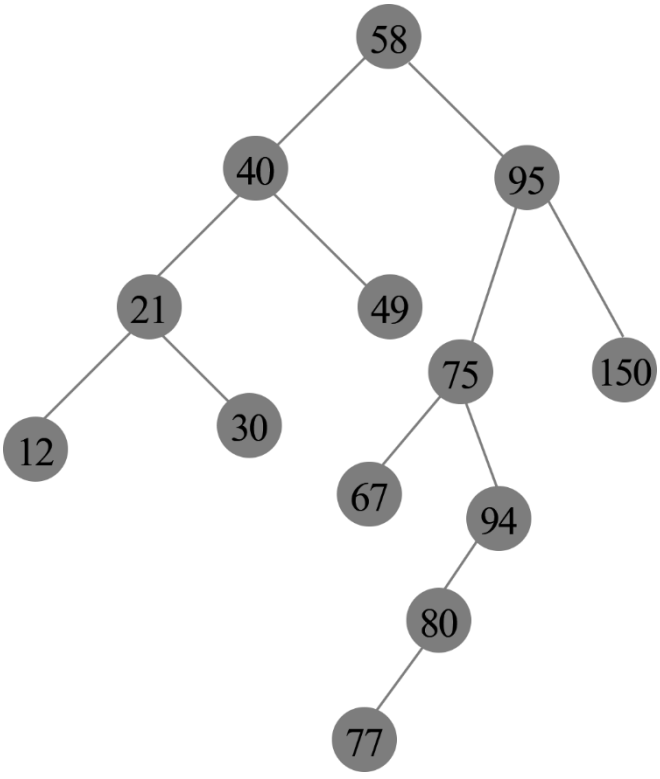


.

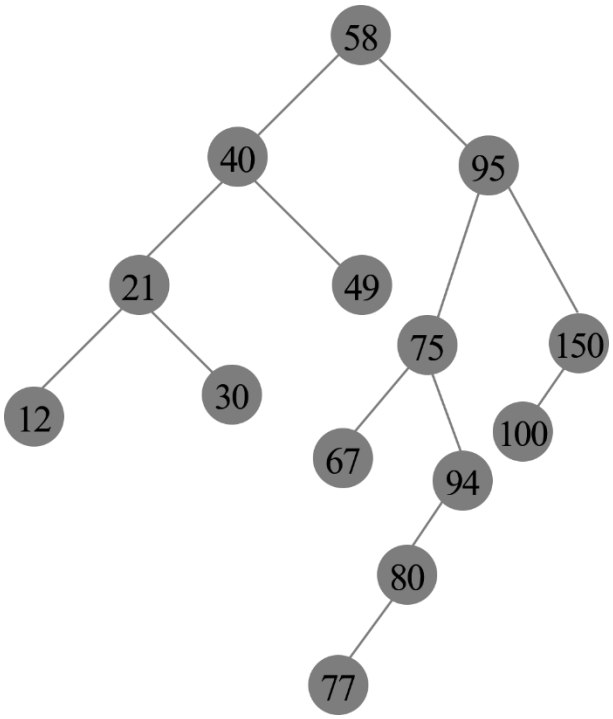
Inserting 67 to the left child of 75.



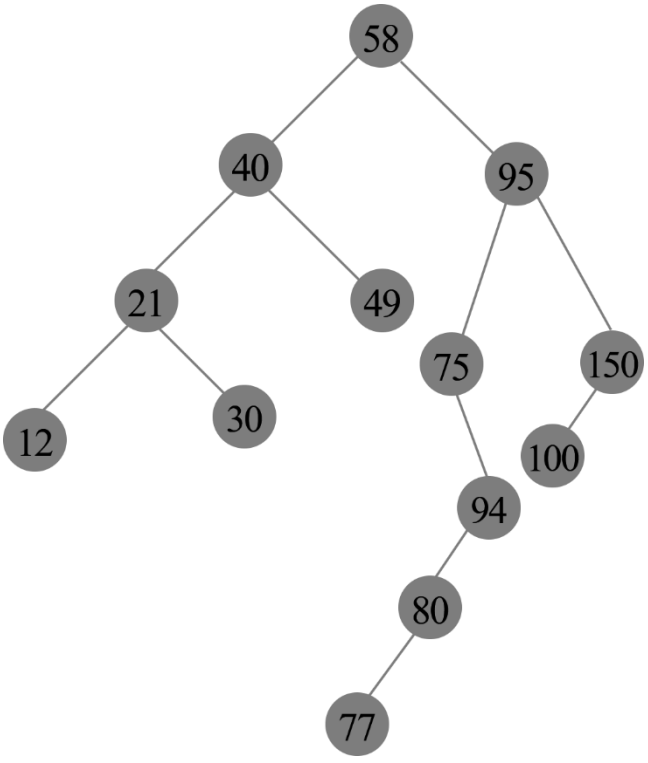
Inserting 12 to the left child of 21.



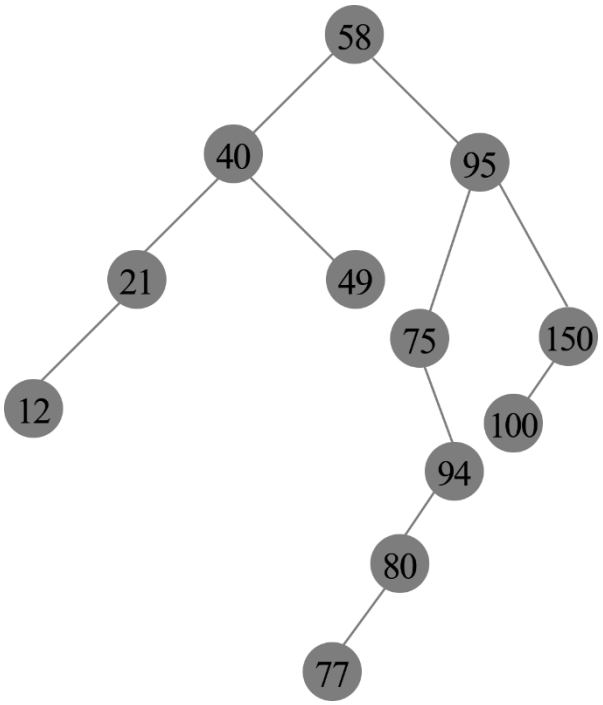
Inserting 77 to the left child of 80.



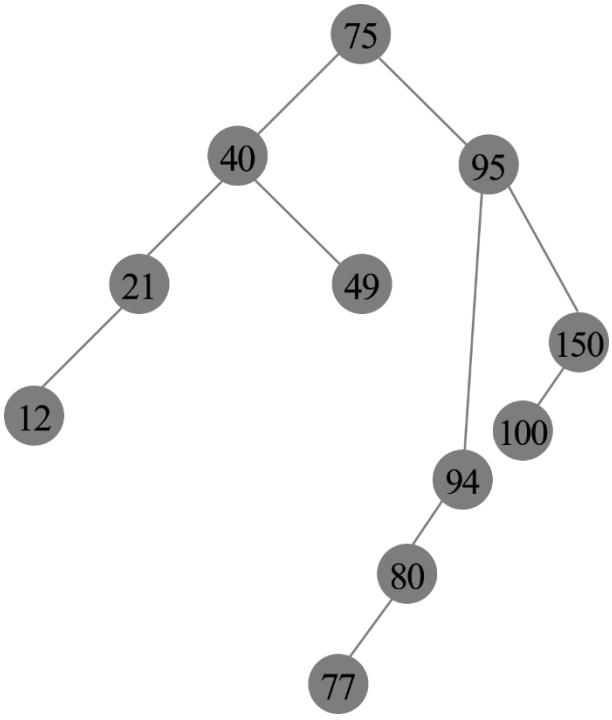
Inserting 100 to the left child of 150.



Deleting 67 from the tree. 67 was a leaf so we just removed it.



Deleting 30 from the tree. 30 was a leaf so we just removed it.



Deleting 58 from the tree. Since 58 had two children, we had to find inorder successor of 58. Inorder successor is found in the following way: find left-most of the right subtree of the root that is going to be deleted. Therefore, 75 was the inorder successor of the node containing 58. Inorder successor is transferred into the 58's location and 95 is connected directly to the node containing 94 since the node containing 75 had only one child, which is 94.

Question 4-)

Part 1-) Analyzing worst case running time complexity of addNgram() function

```
42  void NgramTree::addNgram(string ngram)
43  {
44      NgramNode *current = root;
45      NgramNode **currentsParent = NULL;
46      bool alreadyExist = false;
47
48      //cout << "Currently Adding: " << ngram << endl;
49      if (current == NULL)
50      {
51          root = new NgramNode();
52          (root)->item = ngram;
53          root->visited = false;
54          root->occurrence = 1;
55      } else
56      {
57          while (current != NULL)
58          {
59              if (ngram.compare(current->item) > 0) // if it is alphabetically greater than the current node's string
60              {
61                  currentsParent = &(current->rightChild);
62                  current = current->rightChild;
63              } else if (ngram.compare(current->item) <
64                         0)  // if it is alphabetically smaller than the current node's string
65              {
66                  currentsParent = &(current->leftChild);
67                  current = current->leftChild;
68              } else // if it already exists
69              {
70                  (current->occurrence)++;
71                  alreadyExist = true;
72                  break;
73              }
74          }
75          if (!alreadyExist)
76          {
77              *currentsParent = new NgramNode();
78              (*currentsParent)->item = ngram;
79              (*currentsParent)->occurrence = 1;
80
81          }
82      }
83  }
```

Figure 1: My addNgram() function

For this function, worst case occurs when the string argument that is inserted to binary search tree has the lexicographically highest value and the tree is already sorted in lexicographic order, like an linear 1D array. For that case, the string argument will be inserted in the last position of the tree whose length is n. So the time complexity will be denoted in *O(n)* for the worst case of addNgram() function.

Part 2-) Analyzing worst case running time complexity of operator<< function

```
183  void NgramTree::leftShiftOperatorHelper(NgramNode *root,
184                                          string &result)    //Recursive call happens here. With inorder traversal
185  {
186      if (root != NULL)
187      {
188          leftShiftOperatorHelper(root->leftChild, result);
189          result.append("\"");
190          result.append(root->item);
191          result.append("\" appears ");
192          stringstream temp;
193          string tempStr;
194          temp << root->occurrence;
195          temp >> tempStr;
196          result.append(tempStr);
197          result.append(" time(s)");
198          result.append("\n");
199          leftShiftOperatorHelper(root->rightChild, result);
200      }
201  }
202
203  ostream &operator<<(ostream &out, NgramTree tree)
204  {
205      string result = "";
206      NgramTree *temp = new NgramTree(tree);      --> Copy Constructor: O(n)
207      tree.leftShiftOperatorHelper(temp->root, result);   --> Inorder traversal: O(n)
208      out << result;
209      //NgramTree::destroyTree(temp->root);
210      delete temp;                                --> Destructor: O(n)
211      return out;
212  }
```

Figure 2: My operator<< override and its helper method.

In this part, I needed a helper method in order to operate my traversal easier. My helper method will visit, no matter what, every node once since it is using in order traversal. Also, I am calling copy constructor and destructor here but it does not matter since they are all in *O(n)*. Also, I want to show that Inorder traversal has the Big-O notation *O(n)* by solving a recursion tree.

$$T(n) = 2T\left(\frac{n}{2}\right) + c, \text{ where c is a constant.}$$

Similarly, $$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + c \rightarrow T(n) = 4T\left(\frac{n}{4}\right) + 2c + c$$

…

…

$$T(n) = nT(1) + c \sum_{i=0}^{height(h)} 2^i$$

So, complexity of inorder traversal is $O(2^{h+1} - 1)$.

Note that $h = \log(n)$.

Therefore, the actual Big-O notation of operator<< is $O(2n - 1) \cong O(n)$.

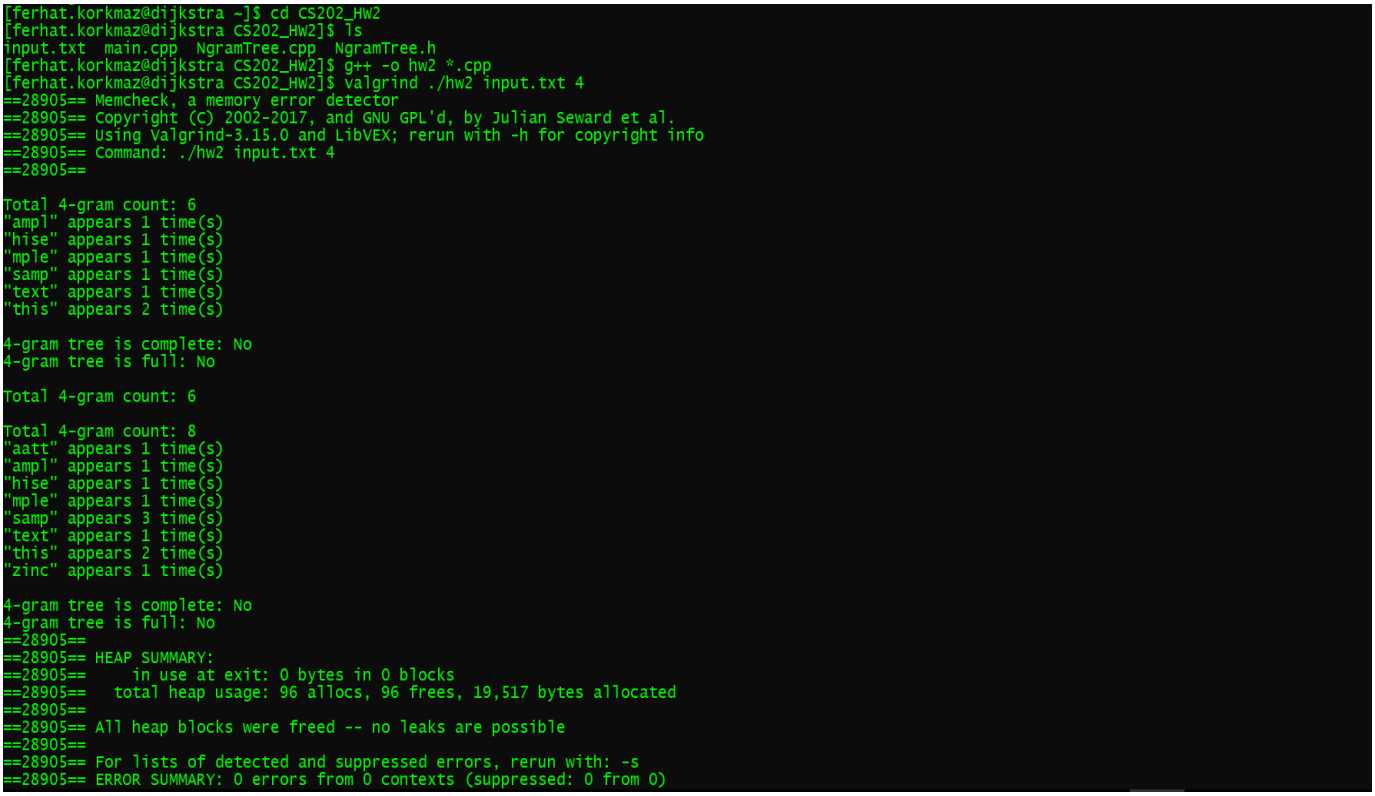


Figure 3: Sample output of my program in BCC Linux machine.