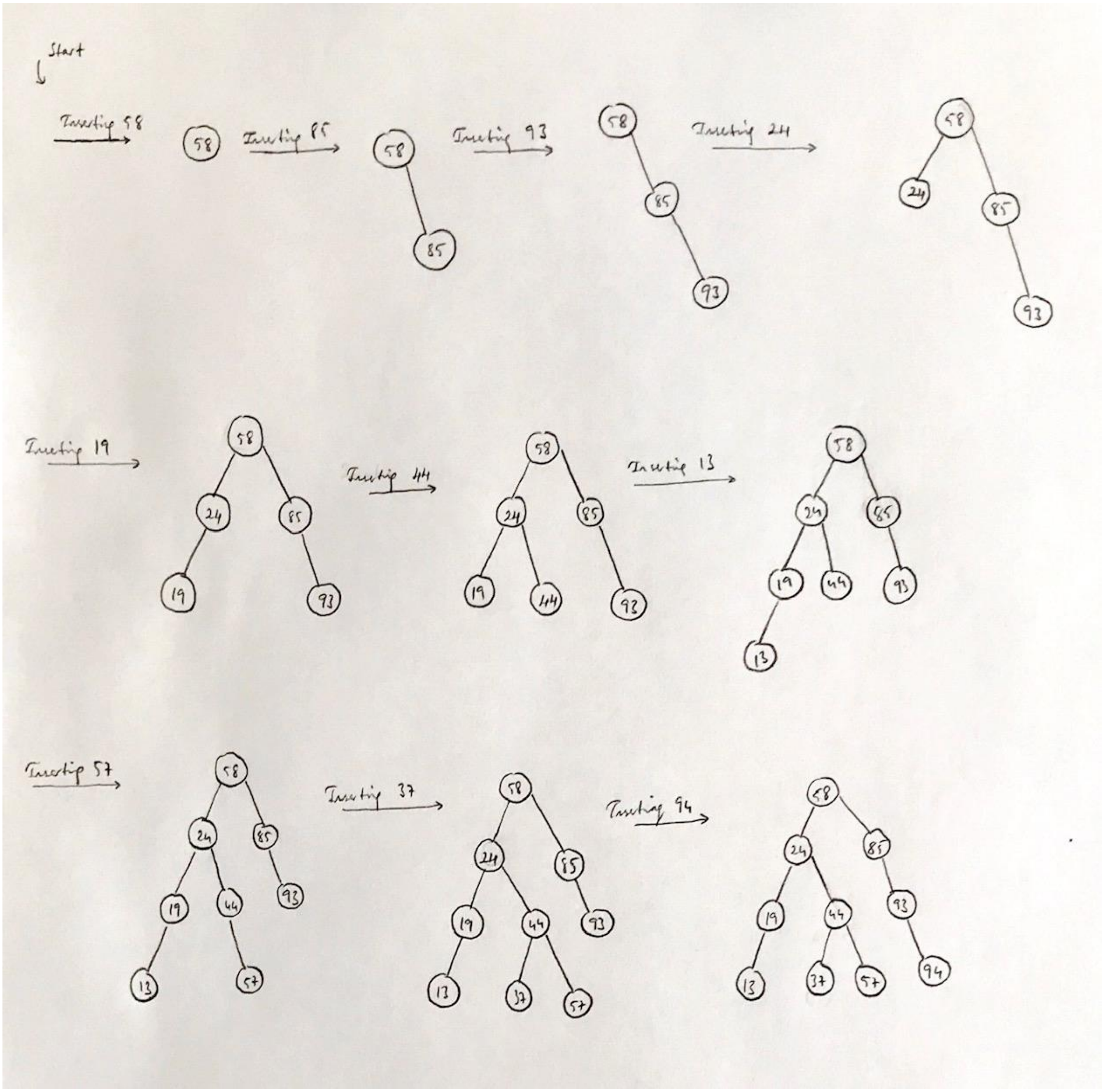
Name/Surname: Ümit Yiğit Başaran
ID: 21704103
Section: 2
Assignment: 2

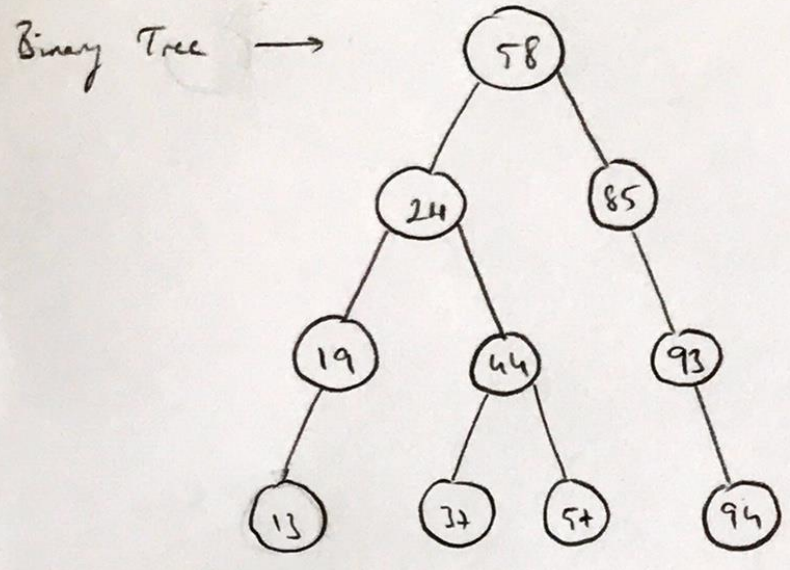
Question 1

(a) Insert 58, 85, 93, 24, 19, 44, 13, 57, 37, 94 into an empty binary search tree (BST) in the given order. Show the resulting BSTs after every insertion.



Name/Surname: Ümit Yiğit Başaran
ID: 21704103
Section: 2
Assignment: 2

(b) What are the preorder, inorder, and postorder traversals of the BST you have after (a)?



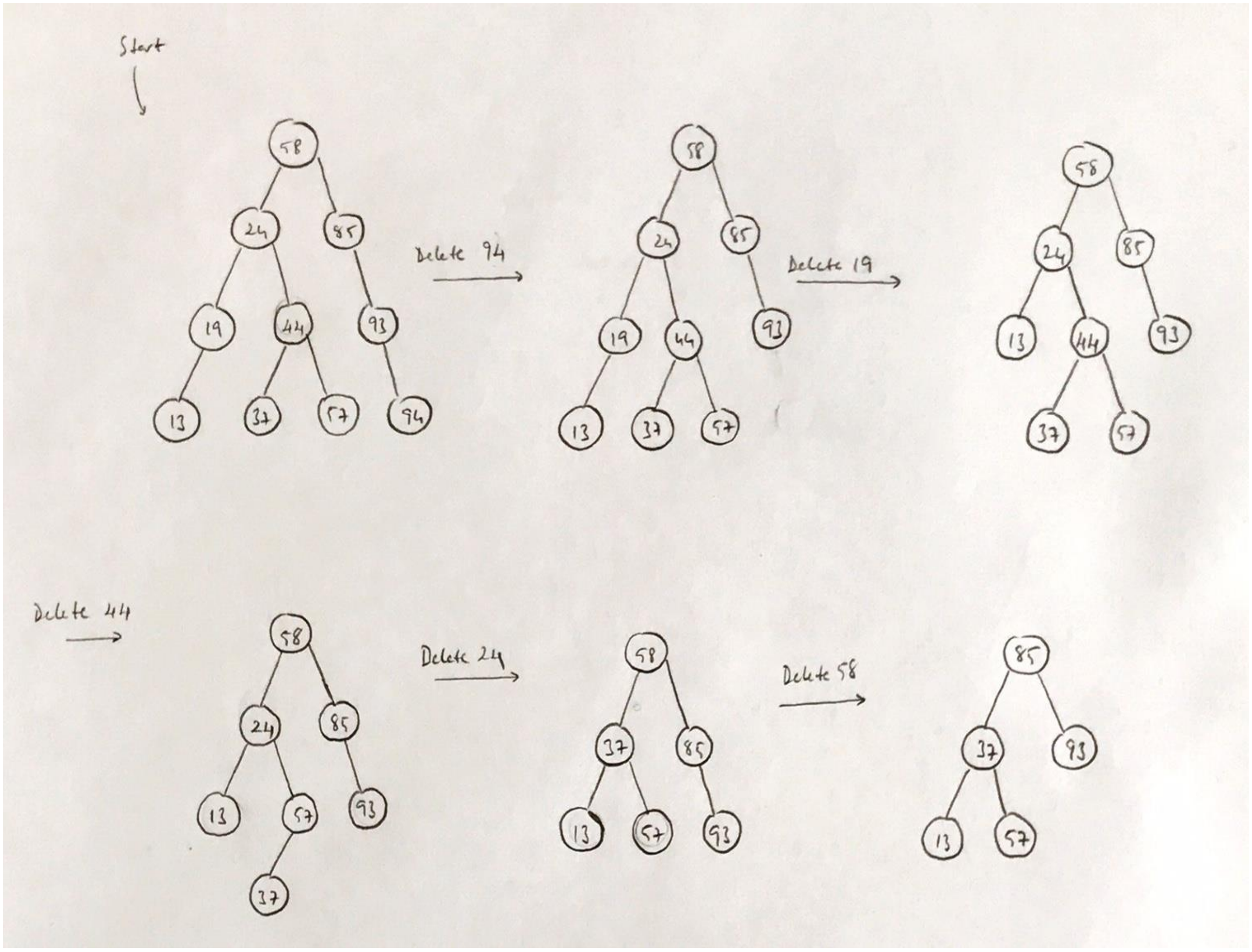
Preorder → 58, 24, 19, 13, 44, 37, 57, 85, 93, 94

Inorder → 13, 19, 24, 37, 44, 57, 58, 85, 93, 94

Postorder → 13, 19, 37, 57, 44, 24, 94, 93, 85, 58

Name/Surname: Ümit Yiğit Başaran
ID: 21704103
Section: 2
Assignment: 2

(c) Delete 94, 19, 44, 24, 58 from the BST you have after (a) in the given order show the resulting BSTs after every deletion.



Name/Surname: Ümit Yiğit Başaran
ID: 21704103
Section: 2
Assignment: 2

Question 3

In calculateEntropy function firstly I find the total number of the given classes and by using the formula of the entropy I calculate the entropy of the given sample space. Here this function has $O(numClass)$ time complexity because to calculate the entropy function needs to travel all the items in the given classCount array.

In calculateInformationGain function I define parameters that I use for the calculate the information gain depending on the given formula. Firstly, I calculate the entropy of the parent sample space and to provide the appropriate parent sample space I use the usedSample array. Then by the given featureId parameter I divide the sample to sample left child that contains only "true" data and sample right child that contains only "false" data. I retrieve their sizes and I calculate the probabilities of these child samples. Then I calculate the entropies of the child samples and calculate the whole information gain based on given featureId parameter. Here this function has helper functions "calculateHP", "retrieveSizeLeftRight", and "calculateLeftRight". In "calculateHP" function I calculate the entropy of the parent. In "retrieveSizeLeftRight" function I retrieve the left and right sizes of the child samples. In "calculateLeftRight" I calculate the entropies of the child samples. The complexity of the calculateInformationGain function is $O(numSamples)$ because in this function, it traverses all the samples and calculate the information gain.

In DecisionTreeNode class I define leftChilPtr that points to the left child of the node, rightChildPtr that points to the right child of the node, item that holds the value of the node and a leaf property that represents whether a node is a leaf or not. Additionally, I write setNodeItem function that set the node to the given parameter and isLeaf function that returns the leaf property of the node. Here all the functions and implementations have $O(1)$ time complexity. Destructor of this class checks the left and right children and if they exist then delete them individually.

DecisionTree class has a private variable DecisionTreeNode pointer and by using this pointer it constructs a decision tree. Destructor of this class checks the root and if it is not NULL then deletes it.

Name/Surname: Ümit Yiğit Başaran
ID: 21704103
Section: 2
Assignment: 2

In train function I set usedSamples and usedFeatures arrays to all false. Then I wrote a helper function called trainTree, that takes decision tree node pointer as an extra parameter to traverse while constructing the tree, for train function and called it. trainTree function is a recursive function, and its base case occurs after it constructs at least one node and when the constructed node is a leaf.

```
//pseudo-code
If the node that will be constructed is NULL
    For every features of the data matrix
        If this feature was not used before
            Calculate the information gain
            If this information gain is greater than the past one
                Set maximum information gain as this information gain
                Set best feature id as this feature id

    Based on the maximum information gain value construct the node
    // Either leaf or not leaf
    // Use usedSample array to select the correct class value of the leaf

    Set usedSampleLeft and usedSampleRight arrays
    // Based on true and false values of this given features
    // Use usedSample array to select the index wanted to use

    If the constructed node is not a leaf // BASE CASE
        Then, first train tree for the left child of the tree // Recursion
        Second, train tree for the right child of the tree
```

The complexity of the train function can be differentiate based on the given data matrix. For average case, it has $O(numSamples * numFeatures * \log(numFeatures))$ time complexity. Also, the function which is also called as train that takes the name of the file, the number of the samples and the number of the features, it traverses of the whole given file and create data matrix and labels array then pass these values to the train function itself. Because this function traverses all the numbers in the file it has additional $O(numSamples * (numFeatures + 1))$ complexity. As a result because, because the complexity of reading integers from a file is less than the train function itself, train function called with a file name has $O(numSamples * numFeatures * \log(numFeatures))$ time complexity.

Name/Surname: Ümit Yiğit Başaran
ID: 21704103
Section: 2
Assignment: 2

In predict function I used a helper private function called predictTree and I directly called in my predict function. predictTree function is a recursive function, and its base case occurs when the node pointer reaches one of the leaves in the tree.

```
//pseudo-code
If the node is a leaf // BASE CASE
    Then, return the item of this node
If given index which is taken from the node of data array is true
    Then traverse leftChild // Recursion
Else
    Traverse rightChild
```

Therefore, if we take the decision tree's height as h then and the number of nodes as n then the complexity of this function is $O(\log(n)) = O(h)$. If we calculate with the given properties the function $O(\log(\log(numFeatures)))$ time complexity.

In test function, I used predict function to test the given data from train_data.txt and I use it numSamples times because of collecting all true and false predictions.

```
// pseudo-code
For every samples in the data matrix // numSamples
    Predict sample // predict function
    Check whether it is true or not
    If it is true then increase # of true predictions
Return (# of true predictions / # of all samples)
```

Therefore, the complexity of the test function is $O(numSamples * h) = O(numSamples * \log(n))$ where n is the number of nodes in the decision tree and h is the height of the tree. If we calculate with the given properties, the function $O(\log(numFeatures) * numSamples)$. Also, the function which is also called as test that takes the name of the file and the number of the samples, it traverses of the whole given file and create data matrix and labels array then pass these values to the test function itself. In order to construct data and labels, I create a private integer variable called tmpNumFeatures and it takes the numFeatures from the train function parameter. Because this function traverses all the numbers in the file it has additional $O(numSamples * (numFeatures + 1))$ complexity. As a result, because the complexity of reading integers from a file is greater than the test function itself, test function called with a file name has $O(numSamples * (numFeatures + 1))$ time complexity.

Name/Surname: Ümit Yiğit Başaran
ID: 21704103
Section: 2
Assignment: 2

In print function, I use a helper function called printTree function that takes one decision tree node and an integer called stage that adjusts the number of spaces in the printed tree.

```
// pseudo-code
// BASE CASE
If the node which will be printed is NULL
    Then do not print anything and return

From zero to stage print a tab
If the node is a leaf firstly print "class ="
Print the exact node's item

Print the left child of the node. // Recursion
Print the right child of the node // Recursion
```

Therefore, the complexity of the print function is depending on the decision tree. For the worst case, we need a full binary tree with n nodes, then, we have $O(nlogn)$ time complexity because if we take height of the tree as h then for every height of the tree we have $2^h * h$ operations. If we generalize that we have $\sum 2^h * h$ operations and it is equal to $O(nlogn)$ time complexity. For the average case we have $2h$ operations for each height than if we generalize that we have $O(log^2 n)$ time complexity. If we calculate this complexity with given data, then, we can write log $(numSamples)$ rather than h.