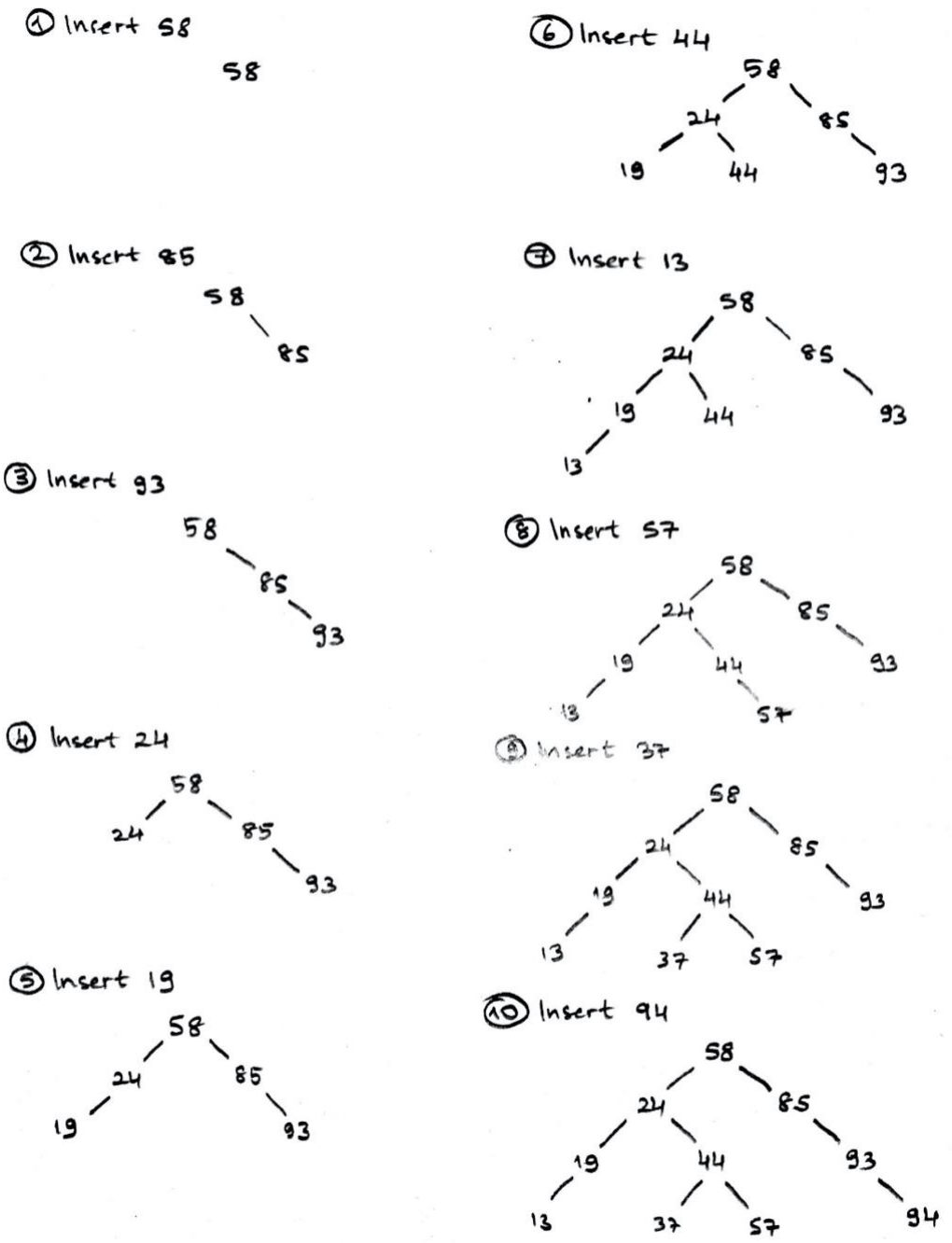
Arda Akça Büyük
21802835
Section 3

CS202 – HW2

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.



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

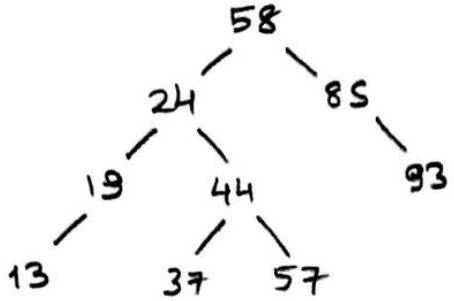
preorder : 58 24 19 13 44 37 57 85 93 94

inorder : 13 19 24 37 44 57 58 85 93 94

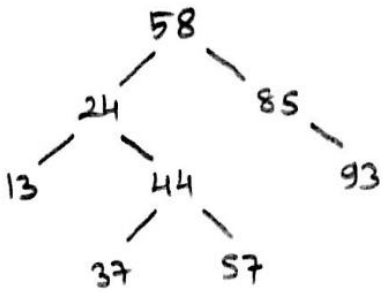
postorder : 13 19 37 57 44 24 94 93 85 58

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.

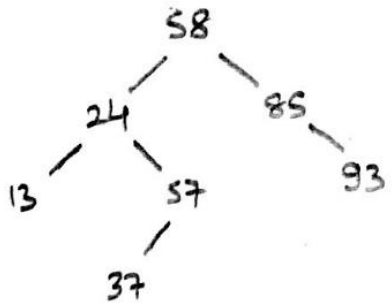
① Delete 94



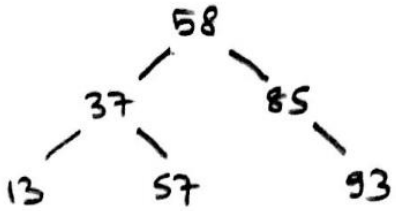
② Delete 19



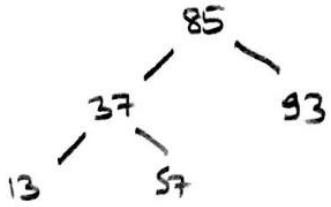
③ Delete 44



④ Delete 24



⑤ Delete 58



Question 3

1. `double calculateEntropy(const int* classCounts, const int numClasses);`

Since this function has two for loops that both iterate ***numClasses*** times and statements that are $O(1)$ the time complexity is $O(numClasses)$

2. `double calculateInformationGain(const bool** data, const int* labels, const int numSamples, const int numFeatures, const bool* usedSamples, const int featureId);`

This function has several loops and statements. There are two loops iterating ***numSamples*** times and one loop iterating ***numClasses*** times. After these loops, ***calculateEntropy*** is called three times in total for parent node, left node and right node. As we sum up these, we end up with

$$2 * O(numSamples) + 4 * O(numClasses) = O(numSamples + numClasses)$$

Since ***numSamples*** $\gg$ ***numClasses***, overall time complexity of this function is $O(numSamples)$.

3. `int chooseFeature(const bool** data, const int* labels, const int numSamples, const int numFeatures, const bool* usedSamples, bool* availableFeatures);`

This function calls ***calculateInformationGain*** for each feature, so overall time complexity is $O(numSamples * numFeatures)$

4. `void split(const bool **data, const int *labels, const int numSamples, const int numFeatures, DecisionTreeNode *node, bool *usedSamples, bool *availableFeatures, const int feature);`

This function is the helper function of ***train*** that splits the nodes recursively to construct the binary tree. It calls the ***chooseFeature*** function, which is $O(numSamples * numFeatures)$, twice for the left node and the right node. Although there are several linear loops in the function, they do not contribute to the time complexity of the function since they are independent loops. At the end, this function calls itself twice for the left node and the right node until there are no features left or the node is purified. In the worst case, there are ***numFeatures*** nodes in the tree at the end, which means the split function is called ***numFeatures*** times. Therefore, time complexity of this function is $O(numSamples * numFeatures^2)$. If we let $numSamples = n$, and $numFeatures = m$, time complexity of this function is $O(nm^2)$.

5. `void train(const bool **data, const int *labels, const int numSamples, const int numFeatures);`

This function initializes the necessary arrays and the root node, then calls the ***split*** function. Therefore its time complexity is the same as ***split***, which is $O(nm^2)$.

6. `void train(const string fileName, const int numSamples, const int numFeatures);`

This function parses the file into a 2d array with a nested loop of $numSamples * numFeatures$, after that, it calls the ***train*** function above. So, overall time complexity of this function is $O(nm^2 + nm) = O(nm^2)$.

7. `int predict(const bool *data);`

This function has one loop that iterates through the decision tree, which has an average height of $log(numFeatures)$. Therefore, overall time complexity of this function is $O(log(numFeatures))$. However, depending on the tree (if height of the tree is ***numFeatures***), this function's time complexity might be $O(numFeatures)$ which is the worst case time complexity for this function.

8. `double test(const bool** data, const int *labels, const int numSamples);`

This function calls ***predict*** function for each sample in the test dataset. Therefore, time complexity of this function is $O(numSamples * log(numFeatures))$. Worst case might also occur in this function if the tree has a height of ***numFeatures***, which is $O(numSamples * numFeatures)$.

9. `double test(const string fileName, const int numSamples);`

This function first parses a textfile into a 2d double array with a nested loop that iterates $numSamples * numFeatures$ times. After that, it calls the ***test*** function above to make the calculations. In the end, we end up with

$$O(numSamples * numFeatures) + O(numSamples * numFeatures)$$

or (depending on height of the tree)

$$O(numSamples * numFeatures) + O(numSamples * log(numFeatures))$$

Regardless of the equation we ended up with, time complexity of this function is $O(numSamples * numFeatures)$

10. `void print(DecisionTreeNode *node, int level);`

This function traverses through all nodes of the tree and prints tabs according to node levels. Let the number of nodes be N. Time complexity of this function is $O(N * log(N))$ in average and best cases (N is for traversing all nodes, $log(N)$ is for printing tabs). Worst case scenario time complexity is $O(N^2)$. Worst case occurs when the height of the tree is N.

11. `void print();`

This function only calls the helper function ***print***, which is explained above. Therefore, its time complexity is the same as the helper function ***print***.