**1. What is the estimated depth of a Decision Tree trained (unrestricted) on a one million instance training set?**

Ans-The estimated depth of a decision tree trained on a one million instance training set can vary depending on the complexity of the problem and the features used for training the tree. However, a decision tree trained on a large dataset can potentially become very deep and complex, leading to overfitting and poor generalization.

In practice, to prevent overfitting, the depth of decision trees is often limited by setting a maximum depth parameter during training. Typically, the depth of decision trees used in practice ranges from 10 to 100, depending on the size and complexity of the dataset, and the desired trade-off between model complexity and performance.

**2. Is the Gini impurity of a node usually lower or higher than that of its parent? Is it always lower/greater, or is it usually lower/greater?**

Ans-The Gini impurity of a node in a decision tree is usually lower than or equal to that of its parent. This is because the decision tree algorithm aims to split the data in a way that minimizes impurity and increases homogeneity in each split. By splitting the data into two or more subsets based on the feature that best separates the data, the impurity of the resulting subsets is typically lower than that of the parent node.

However, there can be cases where the impurity of a node is higher than that of its parent. This can occur if the split results in two subsets that are less homogeneous than the original set, or if the split is based on a feature that is not informative for the classification task. In such cases, the decision tree algorithm may choose to stop splitting and create a leaf node to avoid overfitting the data.Overall the Gini impurity of a node is not always lower than its parent, it is usually lower or equal due to the algorithm's goal of minimizing impurity in each split.

**3. Explain if its a good idea to reduce max depth if a Decision Tree is overfitting the training set?**

Ans- Yes, it is generally a good idea to reduce the maximum depth of a decision tree if it is overfitting the training set. Overfitting occurs when the decision tree captures the noise or random variations in the training data, leading to poor generalization performance on new, unseen data.

Reducing the maximum depth of a decision tree is a common method to prevent overfitting and improve its generalization performance. By reducing the maximum depth, the decision tree is forced to learn simpler and more general rules that can be applied to new data. This helps to reduce the complexity of the model and limit its capacity to fit the noise in the training data.However, it's important to find the right balance between underfitting and overfitting when choosing the maximum depth.

**4. Explain if its a good idea to try scaling the input features if a Decision Tree underfits the training set?**

Ans-Scaling input features is generally not necessary for decision trees since the algorithm is insensitive to monotonic transformations of the input features. However, if a decision tree is underfitting the training set, meaning it has high bias and low variance, it may be helpful to scale the input features to improve its performance.

**5. How much time will it take to train another Decision Tree on a training set of 10 million instances if it takes an hour to train a Decision Tree on a training set with 1 million instances?**

Ans- The time it takes to train a decision tree on a training set is dependent on several factors such as the complexity of the problem, the number of features, and the computing resources available. However, assuming that the complexity of the problem and the number of features are similar, we can estimate the time it takes to train a decision tree on a training set of 10 million instances based on the time it takes to train a decision tree on a training set with 1 million instances.

**6. Will setting presort=True speed up training if your training set has 100,000 instances?**

Ans-Setting the presort=True parameter in scikit-learn's DecisionTreeClassifier or DecisionTreeRegressor can be used to speed up training time in certain cases, but it is not always guaranteed to do so. The presort parameter is used to pre-sort the data before fitting the decision tree, which can be helpful when the dataset is small or when the tree is shallow.

For larger datasets, pre-sorting can actually slow down training time, as the time required to sort the data may exceed the time saved during the tree-building process. Therefore, setting presort=True for a training set with 100,000 instances may not necessarily speed up training time, as this dataset size is considered moderate to large.

**7. Follow these steps to train and fine-tune a Decision Tree for the moons dataset:**

**a. To build a moons dataset, use make moons(n samples=10000, noise=0.4).**

**b. Divide the dataset into a training and a test collection with train test split().**

**c. To find good hyperparameters values for a DecisionTreeClassifier, use grid search with cross-validation (with the GridSearchCV class). Try different values for max leaf nodes.**

**d. Use these hyperparameters to train the model on the entire training set, and then assess its output on the test set. You can achieve an accuracy of 85 to 87 percent.**

Ans-Here's an example code implementation of the above steps:

from sklearn.datasets import make\_moons

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.tree import DecisionTreeClassifier

# Step a: Generate moons dataset

X, y = make\_moons(n\_samples=10000, noise=0.4)

# Step b: Split the dataset into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step c: Grid search for hyperparameters

param\_grid = {'max\_leaf\_nodes': [2, 4, 8, 16, 32, 64, 128]}

grid\_search = GridSearchCV(DecisionTreeClassifier(random\_state=42), param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)

# Step d: Train model on entire training set and evaluate on test set

tree\_clf = DecisionTreeClassifier(random\_state=42, \*\*grid\_search.best\_params\_)

tree\_clf.fit(X\_train, y\_train)

accuracy = tree\_clf.score(X\_test, y\_test)

print("Accuracy:", accuracy)

**8. Follow these steps to grow a forest:**

**a. Using the same method as before, create 1,000 subsets of the training set, each containing 100 instances chosen at random. You can do this with Scikit-ShuffleSplit Learn's class.**

**b. Using the best hyperparameter values found in the previous exercise, train one Decision Tree on each subset. On the test collection, evaluate these 1,000 Decision Trees. These Decision Trees would likely perform worse than the first Decision Tree, achieving only around 80% accuracy, since they were trained on smaller sets.**

**c. Now the magic begins. Create 1,000 Decision Tree predictions for each test set case, and keep only the most common prediction (you can do this with SciPy's mode() function). Over the test collection, this method gives you majority-vote predictions.**

**d. On the test range, evaluate these predictions: you should achieve a slightly higher accuracy than the first model (approx 0.5 to 1.5 percent higher). You've successfully learned a Random Forest classifier!**