302 Project 2 W17

A Run-Time Estimate Of The Average Height Of A Binary Search Tree

Read the given theory.

Look at the given code.

Pick the correct answers for the two multiple choice questions. Simply type your multiple choice letter for each of the multiple choice questions in the Canvas text box.

Code the demo-driver class that uses the given BinarySearchTree class.

Specifications:

Program/project description at the top of your main method file.

Descriptive variable names.

Your name, date, in javadoc between import and main method header, just like NetBeans does.

Only one line in main: new Lab2().run(); --- assuming your project is called Lab2.

Thoroughly commented.

Your code reproduces similar output to the given output.

There is a loop, n, that varies the total of records added.

There is a loop, that averages the heights of 20 random trees.

There is a loop that uses the BST add(Math.random()) to add random numbers using the Math class.

This lab focuses on the height of a binary search tree, and provides run-time support for the claim, in Chapter 10, that the average height of a binary search tree is logarithmic in n, the size of the tree.

The height of a BinarySearchTree

In Chapter 10, a claim is made that the average height of a binary search tree is logarithmic in n. In this lab, you will provide some support for that claim through a run-time experiment.

To start, recall the definition of a height function from Chapter 9:

for a binary tree t, we define the height of t, written height(t) as:

if t is empty, height(t) = -1;

otherwise, height(t) = 1 + max (height(leftTree(t)), height(rightTree(t))

This definition uses functional notation, not object-oriented notation. Here is the specification of a height method in the BinarySearchTree class:

/\*\*

\* Calculates the height of this BinarySearchTree object.

\*

\* @return an int containing the height of this BinarySearchTree object.

\*\*/

public int height()

Logarithms, Base 2

Suppose you want to perform a run-time experiment to support the claim that the average height of a binary search tree is logarithmic in n, the size of the tree. Here is one way to proceed. Start by reading in the value for n. Then test the claim for binary search trees of size n, 2n, 4n, 8n and 16n. For each size, conduct 20 trials. In each trial, a BinarySearchTree of that size with random Double objects is generated and the height of that tree is calculated. After the last trial, the ratio of the average height to log2n is printed, where n refers to the current size of the tree. Suppose this ratio does not change much for successively larger values of n. Then that should increase your confidence that the average height of a binary search tree is, indeed, logarithmic in n.

For a given value for n, the code is given to conduct 20 trials. In each trial, n random Double objects are added to a BinarySearchTree, the height of that tree is calculated and the BinarySearchTree is cleared. After the last trial, the ratio of the average height to log2n is printed. Note that a/b/c = (a/b)/c, which is not necessarily equal to a/(b/c). Duplicates are not allowed in a BinarySearchTree object, so this code makes sure that each tree actually gets n elements.

Look at the given code.

Pick the correct answers for the two multiple choice questions.

There is nothing to code; nothing to upload. Simply type your multiple choice letter for each of the multiple choice questions in the Canvas text box.

Quick Quiz 1: Assume that a definition of the height method has been developed. Which one of the following includes a correct call to that method from a method outside of the BinarySearchTree class?

a. System.out.println (HEIGHT\_MESSAGE + height());

b. System.out.println (HEIGHT\_MESSAGE + height (myTree));

c. System.out.println (HEIGHT\_MESSAGE + myTree.height());

d. System.out.println (HEIGHT\_MESSAGE + myTree.height (t));

Quick Quiz 2: Which of the following pairs contains the code for generating a random double (in the range from 0.0 inclusive to 1.0 exclusive) and the code for log**2**(n)?

a. Math.random (n); Math.log (n)

b. Math.random (n); Math.log (n) / Math.log (2)

c. Math.random(); Math.log (n) / Math.log (2)

d. Math.random(1.0); Math.log (n) / Math.log (2)

Lab 2 rubric

1 point:

Both multiple choice questions answered correctly.

Description at the top of your main method file.

Descriptive variable names.

name, date, in javadoc between import and main method header, just like NetBeans does.

1 point. Only one line in main: new Lab2().run(); --- assuming your project is called Lab2.

1 point. Thoroughly commented.

7 points. Your code reproduces similar output to the given output.

There is a loop, n, that varies the total of records added.

There is a loop, that averages the heights of 20 random trees.

There is a loop that uses the BST add(Math.random()) adding random numbers using the Math class.

You many use these print statements:

System.out.println(

"Average the heights of "+numOfTrials+" random trees");

System.out.println(" n | ratio of average height to log2n");

System.out.println(n + " | " + ratio);

Late penalty is the following algorithm:

1% per minute for the first 10 minutes.

After 10 minutes late, 10% off for the next four hours.

After five hours late, 50%.

1 minute late 1% late fee, so highest possible score 99%

2 minutes late 2% late fee

3 minutes late 3% late fee

4 minutes late 4% late fee

5 minutes late 5% late fee

6 minutes late 6% late fee

7 minutes late 7% late fee

8 minutes late 8% late fee

9 minutes late 9% late fee

10 minutes late 10% late fee

1 hour late 10% late fee,

2 hours late 20% late fee

3 hours late 30% late fee

4 hours late 40% late fee

5 hours or more late 50% late fee