Recursion and Sorting: Binary Search Trees and Treesort

ISTA 350 Hw5, Due 4/2/2020 at 11:59 pm

Introduction. This homework is the culmination of the first half of the course's content, incorporating linked data structures, magic methods, and algorithms whose runtimes and big O's we will examine in class. You will be implementing `Node` and `BST` classes. The `BST` class is a linked implementation of a binary search tree built from `Node` objects. The rules of binary search trees are (a recursive definition):

- The left subtree of a node contains only nodes with data lesser than the node's datum.
- The right subtree of a node contains only nodes with data greater than the node's datum.
- The left and right subtrees each must also be binary search trees.

Special note: almost all of the tests depend on the previously tested methods working correctly.

Instructions. Create a module named `hw5.py`. Below is the spec for two classes containing eleven methods between them and a function. Implement them and upload your module to the D2L Assignments folder.

Testing. Download `hw5_test.py` and auxiliary files and put them in the same folder as your `hw5.py` module. Run it from the command line to see your current correctness score. Each of the 11 methods and the function is worth 8.3% of your correctness score. You can examine the test module in a text editor to understand better what your code should do. The test module is part of the spec. The test file we will use to grade your program will be different and may uncover failings in your work not evident upon testing with the provided file. Add any necessary tests to make sure your code works in all cases.

Documentation. Your module must contain a header docstring containing your name, your section leader's name, the date, `ISTA 350 Hw5`, and a brief summary of the module. Each method/function must contain a docstring. Each docstring should include a description of the function's purpose, the name, type, and purpose of each parameter, and the type and meaning of the function's return value.

Grading. Your module will be graded on correctness, documentation, and coding style. Code should be clear and concise. You will only lose style points if your code is a real mess. Include inline comments to explain tricky lines and summarize sections of code.

Collaboration. Collaboration is allowed. You are responsible for your learning. Depending too much on others will hurt you on the tests. "Helping" others too much harms them in reality. Cite any sources/collaborators in your header docstring. Leaving this out is dishonest.

`class Node:`

`init`: `init` takes one argument. It initializes an instance variable called `datum` to the argument, and instance variables `left` and `right` to `None`.

`add`: Insert a new item (this method's sole argument) into the subtree that has `self` as its root, if the item is not already in the tree. Do this such that the rules of binary search trees are maintained (see **Introduction**) after insertion.

`contains`: This magic method takes an item and returns `True` if the item is in the subtree that has `self` as its root, `False` otherwise. What built-in Python functionality do you think this enables?

`sort`: Takes a list and appends the data in the subtree that has `self` as its root in sorted order.

`eq`: Takes a `Node` object as an argument. Return `True` if the subtrees that has `self` and `other` as their roots have the same shape and the corresponding `Node` objects have the same data.

`class BST`:

`init`: `init` takes one argument with a default value of `None`. If the argument is `None`, set `self.root` to `None`. Otherwise, initialize `self.root` to a `Node` containing the argument.

`add`: Insert a new item (this method's sole argument) into `self`.

`from_file`: This classmethod takes a filename and a type (e.g. `int`, `str`, `float`, etc.) with a default value of `None`. Make a tree, open the file, add the value on each line to the tree, casting it first to the type argument if it is not `None`. Return the tree.

`contains`: This magic method takes an item and returns `True` if the item is in `self`, `False` otherwise.

`sort`: Return a list of the data in `self` in sorted order.

`eq`: Takes a `BST` object as an argument. Return `True` if `self` and `other` have equal root `Nodes`.

Function Specifications:

`selection_sort`: Takes a list and sorts it in place. Do not use any built-in sorting functionality. Go through the list, find the location of the smallest element, and swap it with the element in the first position. Starting from the second position, go through the list…