SE 5930
Machine Problem 7 (MP7)
Comparing Lists & Trees

30 points

We are learning about **Linked Lists** & **Trees.** In this program you will compare the search performance of an **Unsorted List**, a **Sorted List**, and a **Binary Search Tree.**

- The Unsorted List will be implemented using Python's `list class` as the parent class for `class UnsortedList(list):`
- The Sorted List will be implemented using the Doubly Linked Sorted List from the Lab 11 assignment… `class SortedList():`
- The Binary Search Tree will be implemented using the Linked Binary Tree example from section 8.3 and 8.5, along with the additional instance methods from the Linked Binary Search Tree example in section 11.1

Your program will…

- Instantiate an Unsorted List, a Sorted List, and a Binary Search Tree
- Generate 10 random `int` values between `1` and `150`, inserting each of the 10 values into each of the 3 data structures
- Print the contents of each data structure…use `print()` for the Unsorted List, the `printList()` instance method from Lab 11 for the Sorted List, and the Inorder Binary Tree Traversal instance method, `inorder()`, for the Binary Search Tree

That part of the assignment is to verify that your program is correctly inserting the elements into your 3 data structures… after that, your program will…

- Instantiate 3 new data structures
- Populate each data structure again using random `int` values between `1` and `150`, but this time insert 100 `int` values
- For each of the 3 data structures, 1 at a time, generate 100,000 random `int` values between `1` and `150` and execute a search for each value counting the number of successful searches, unsuccessful searches, the number of visits for each successful search, and the number of visits for each unsuccessful search

- Finally, your program will show the results of your 3 trials… your output must look like this…

```
Unsorted List: [131, 149, 1, 73, 132, 21, 59, 113, 41, 93]
Sorted List: [ 1 21 41 59 73 93 113 131 132 149 ]
BST Inorder Traversal: 1 21 41 59 73 93 113 131 132 149

                         Found  Avg Visits Not Found  Avg Visits
Unsorted List            45734       45.64     54266      100.00
Sorted List              45959       53.44     54041       52.95
Binary Search Tree       45869        6.42     54131        8.49
```

Here are two more program runs from my solution so you can see the range of values that would be normal for your results…

```
Unsorted List: [91, 138, 6, 131, 46, 33, 85, 71, 76, 132]
Sorted List: [ 6 33 46 71 76 85 91 131 132 138 ]
BST Inorder Traversal: 6 33 46 71 76 85 91 131 132 138

                         Found  Avg Visits Not Found  Avg Visits
Unsorted List            49725       44.89     50275      100.00
Sorted List              50069       49.85     49931       48.83
Binary Search Tree       49708        6.67     50292        8.77


Unsorted List: [83, 89, 5, 86, 95, 17, 70, 93, 144, 90]
Sorted List: [ 5 17 70 83 86 89 90 93 95 144 ]
BST Inorder Traversal: 5 17 70 83 86 89 90 93 95 144

                         Found  Avg Visits Not Found  Avg Visits
Unsorted List            47925       42.80     52075      100.00
Sorted List              47981       51.54     52019       50.09
Binary Search Tree       48116        7.16     51884        8.91
```

Notice some things about the results before we look at more details of the assignment…

- The number of successful and unsuccessful searches are not exactly the same for each data structure (even though the stored values are exactly the same) because the 100,000 values searched for are randomly different for each
- Roughly half of the 100,00 searches are successful and half are unsuccessful
- The Unsorted List has 100 visits for every unsuccessful search because all 100 elements have to be looked at before it can be determined that the element is not there
- In the sorted list, roughly half of the elements are visited whether the search is successful or unsuccessful
- In the BST, log$_2$100 = 6.64… the BST performance is vastly superior to the 2 lists

The Unsorted List: Augment Python's `list` class in order to be able to count the number of locations accessed during a sequential search of the `list` (normally executed by the `in` statement). Either put this definition at the top of your program, or put it in another file and `import` it at the top of your program.

```
class UnsortedList(list):

#
# class UnsortedList is a subclass of Python's built-in list class
#
# No additional Instance Variables
#
# Constructor:
#         __init__(self)        Simply calls the parent function constructor
#
# 1 Public Instance Method:
#         find(self, element)   Executes a sequential search of the list
#                               starting at location 0 and continuing until
#                               element is found or the end of the list is
#                               reached. Returns a list with 2 values - the
#                               first value is True if element was found,
#                               False otherwise - the second value is the
#                               number of locations visited during the search.
#
```

The Sorted List: In Lab 11 we used a Doubly Linked List to execute a Sorted List. Import that `class` into your program with a few notes and 1 addition.

- Since we completed this in Lab 11, you do not need to document this `class`
- `class SortedList` uses `class Node` and `class LinkedBinaryTree` uses `class _Node`… notice the names are different… they must have different names because the Nodes are different for the Doubly Linked List and the Linked Binary Tree… You can rename them, but the names must be different
- `class SortedList` must define an additional instance method `find()`

```
def find(self, element):

    #
    # Executes a sequential search of the Doubly Linked List starting at
    # self._head._next and continuing until element is found, element has a
    # greater value than the current Node's element, or self._tail is reached.
    #
    # self            The implicit parameter for the list
    #
    # element         A reference to the search value
    #
    # Returns a list with 2 values – the first value is True if element was
    # found, False otherwise - the second value is the number of locations
    # visited during the search.
    #
```

The Linked Binary Search Tree: In section 8.3 and 8.5 we defined a Linked Binary Tree implementation. In section 11.1 we added to that definition to make it a Linked Binary Search Tree. `import` that `class` into your program with 1 addition…

- Again, you do not need to document this `class`
- Remember to use a different name for the Nodes in Doubly Linked List and the Linked Binary Tree… You can rename them, but the names must be different
- `class LinkedBinarySearchTree` must define an additional instance method `find()`

```
def find(self, element):

    #
    # Executes a search of the Binary Search Tree beginning at self.root()
    # and continuing until element is found, or the search runs off the end
    # of a leaf.
    #
    # self            The implicit parameter for the Tree
    #
    # Returns a list with 2 values – the first value is True if element was
    # found, False otherwise - the second value is the number of locations
    # visited during the search.
    #
```

The Main Program: Your solution will define 1 function in your Main Program…

```
def trial(storage, highest, numSearches):

    #
    # Conducts multiple searches in the data structure defined by the parameter
    # storage. Generates a random int value between 1 and highest. Uses the
    # instance method storage.find() to search the data structure for that int
    # value. Repeats this process numSearches times. Counts the number of
    # successful searches, the number of unsuccessful searches, the number of
    # total visits for the successful searches, and the number of total visits
    # for the unsuccessful searches. Returns those values in a list.
    #
    # storage           The data structure to be searched. The data structure
    #                   must store int values and have an instance method find()
    #                   that conducts the search and returns a list where the
    #                   first value is True if the search was successful, False
    #                   otherwise, and the second value is the number of visits
    #                   made during the search.
    # highest           An int value between 1 and highest will be randomly
    #                   generated and searched for.
    # numSearches       The function will conduct this many searches.
    #
    # Returns a list with 4 values... the number of successful searches, the
    # number of total visits for those successful searches, the number of
    # unsuccessful searches, and the number of total visits for those
    # unsuccessful searches.
    #
```

For full credit…

- Hand in a hard copy of your UnsortedList class definition (with documentation), your SortedList class definition (no documentation required), your LinkedBinarySearchTree class definition (no documentation required), your main program script (with documentation), and the output that your script generates.
- Hand in your solution on-time.
- Write clear, well-organized code.
- Use variable names that make sense for the data that they contain.
- Please, hand in 3 sample runs.