Homework Assignment 5 Sorting and Search Trees

Due Wednesday May 22 at 11:59pm

See the Canvas assignment page for information on how to complete and submit this assignment.

Problem 1 Insertion sort (15 Points)

To gain familiarity with insertion sort, take the textbooks code for insertion sort and reverse the sort order so that it sorts the list from largest to smallest value.

Problem 2 Bubble sort (15 points)

To gain familiarity with bubble sort, which we will not talk about in class, take the textbooks code for bubble sort and reverse the sort order so that it sorts the list from largest to smallest value. Also, to make it more Pythonish, change the three lines of code used to swap values so that it is done with one simultaneous assignment.

Problem 3 Selection sort (15 points)

To gain familiarity with selection sort, add an extra parameter k to the selection sort procedure presented in class in Lecture 19 slide 45. Modify the loop so that selection sort runs in linear time for fixed k by returning early with only the first k items in the list sorted.

Problem 4 Merge Sort (25 Points) Challenge problem

To gain greater familiarity with merge sort, we will do a modified version of programming exercise 14 in the book. You will remove the slice operator from in the book's implementation of the mergeSort function.

This will involve two modifications to the code. Similar to what was done to remove slicing from binary search, you will need to pass pointers for the beginning and end of the sublist instead of passing a slice of the list.

Then, as we showed in lecture 20, you will use a temporary list to hold the your result from merging the sorted sublists. That result can be immediately copied back into the original list overwriting the sorted sublists once merging is complete.

We have provided the appropriate function definition based on the books implementation and enough lines of code to get you started. The new code will look a lot like the books code except that you will be merging from the list being sorted into a temporary list. So take some time to

understand exactly what the code in the book is doing before using it as a template for what you will need to add to the new function.

Problem 5 Binary Search Trees (30 points)

Modify the books implementation of the binary search tree described in chapters 6.10-6.13 so that it handles duplicate keys properly. The BinarySearchTree class implements a binary search tree where every node of the tree contains a key and a corresponding data item (the payload). Much like the HashTable class, this is so BinarySearchTree can be used in a similar manner as a Python dictionary.

Currently if a key is already in the tree, then a node is added containing duplicate key and its payload. We want this fixed so it behaves like a Python dictionary. If a key is already in the binary search tree then the old payload is replaced with the new payload. No node should be added.

The put and _put methods of the BinarySearchTree class insert nodes into the binary search tree. The put method will check to see if the tree already has a root. If there is not a root then put will create a new node and install it as the root of the tree. If a root node is already in place then put calls the recursive, helper function _put to search the tree for where to insert the new node.

The only method you will need to modify is _put. Understand what it is doing before you start proposing changes.