Math 260, Fall 2020
Last updated: 9/19/20

HOMEWORK 5

DUE FRI. SEP. 25 (11:59 PM)

Note: this homework is the last before the midterm - which will be assigned a few days after the due date. Plan accordingly!

Exercises. (Nothing to submit here).

E1 (no submission - some exercises for practice). a) In the lecture code, I have included an example of a tree being built (inserting one node at a time), then destroyed by deleting one node at a time. Sketch the tree, then run the algorithm for deleting 'by hand' to check that you get the same result as the code.

Note that there are some special cases when deleting the root node (not addressed in class!) - what is different?

b) Write member functions `min` and `max` that return the nodes with minimum/maximum values in the binary tree.

Programming problems.

Q1 (plotting - a preview). I've added a jupyter notebook on numpy to the course github - you may refer to it as a quick introduction.

After reading the Jupyter notebook on numpy, write a `.py` file that creates a plot of the functions $\sin 2x$ and $\cos 2x$ over the interval $[0, 2\pi]$. Decorate the plot (title, axis labels, different linestyles/colors for the lines, etc.) and add a legend.

I suggest using numpy arrays and vectorized arithmetic (as in the jupyter example) to simplify the calculations.

Submit code that generates this plot and saves it as an image file (so I can easily run your code and make the plot). Other than axis labels, there's no required amount of decoration (but it's good to get acquainted with the plotting features).

Note: problems Q2 and Q3 take priority here, but you'll need the plotting post-midterm.

Q2. *Note: while part (a) is relevant, you don't need to write separate code for it. It's easier to write the merge step directly into the msort function. Part (a) is just a suggestion to write this algorithm separately first to get it to work.*

a) Suppose I have two lists of length n in **non-decreasing order**,

$$a_0 \leq a_1 \cdots \leq a_{N-1}, \qquad b_0 \leq b_1 \cdots \leq b_{N-1}.$$

Write an algorithm that takes in these two lists and returns a (new) list of length $2n$ that has all the elements in non-decreasing order.

b) Complete the (recursive) implementation of mergesort. Write code that sorts the given example list and compares to python's built in (syntax: `mylist.sort()`).

Design note: I've included what's called a 'wrapper' function `mergesort`, which is what you actually call when sorting. Write your doc-string there. It calls an 'internal' function `msort`, which is where your actual implementation goes. In writing your code, consider why it is necessary. The term 'wrapper' is used because the wrapper covers up internal details.

c) Use `time.perf_counter` to time your sorting function and compare to python's built in. You should do a reasonable number of trials (say, 20), then take the average.

For a fair comparison, each time you sort using your method and python's method, you should be using the same list!

Choose a list size n large enough that time differences are 'visible' (the time taken is on the order of a second). Then compare to a size $2n$ and size $4n$ list: does the timing scale in the way you expect? Put your answers in a comment above the relevant code.

Q3 (binary trees). You may use the given implementation of a binary tree here (if so, use the version in the `hw5` folder, not the lecture code folder). *Note:* For these functions, you may either have them take the root node as an input, or the tree itself - your choice.

a) Write a function `leaves(root)` that returns a list containing all leaves of the tree with the given root. (a regular function, not a member function).

Do so using recursion. *Hint: like the 'depth-first search code', the final recursive code should be fairly short.*

b) Define the **depth** of a binary tree to be the maximum distance from the root node to any other node (where distance is measured in 'number of edges').

Write a function `depth(root)` that returns that depth of the tree with the given root node. Again, do so using recursion.

Write test code for (a) and (b) (using e.g. the test tree from the lecture code) that demonstrates that your code works.