Homework 8

Due Friday, April 12 by 5:00pm

"On Wednesdays we wear pink!" — Karen Smith

Handing In

To hand in a homework, go to the directory where your work is saved and run `cs0160_handin hwX` where X is the number of the homework. Make sure that your written work is saved as a .pdf file, and any Python problems are completed in the same directory or a subdirectory. You can re-handin any work by running the handin script again. We'll only grade your most recent submission. To install stencil Python files for a homework, run `cs0160_install hwX`. **You will lose points if you do not hand in your written work as a .pdf file.**

1 Written Problems

Problem 8.1

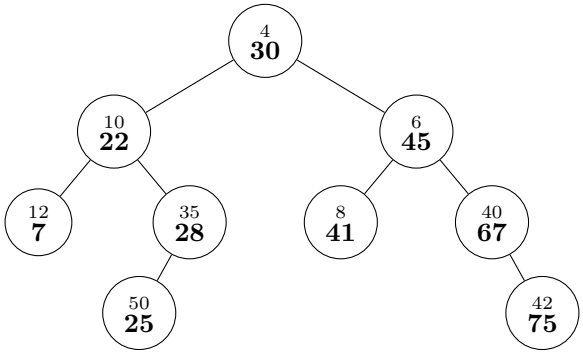
Treaps

A treap is a data structure whose nodes hold two values: a key and a priority. Using these two values, the insertion algorithm places the node in the correct spot in the treap, using both binary search tree order and heap order (hence the name).

Binary search order: This is done with respect to keys. For any node n, all nodes in n's left subtree will have keys smaller than n's key, and all nodes in n's right subtree will have keys greater than n's key.

Heap order: This is done with respect to priorities. For any node n, n's priority will be less than its children's priorities and greater than its parent's priority.

Here is an example, where the priority is on top and the key is on the bottom:



You may assume that all the keys are distinct and all the priorities are distinct. Fun fact: Treaps can be used with random priorities as probabilistically self-balancing binary search trees. (See Wikipedia if you're interested in reading more: **link**)

1. Give a recursive definition of a treap, as in:
   A binary tree T with a key and a priority at each node is a *treap* if

   (a) T is empty, or

   (b) T is non-empty and contains a root with left and right subtrees where ... *(recursive part here)*

2. Write pseudocode for the recursive function buildTreap which, given any list $(k_1, p_1), \ldots, (k_n, p_n)$ of key-priority pairs (where all keys are distinct and all priorities are distinct) builds a Treap.
   The pairs are EntryPair objects, which have getKey() and getPriority() methods. You may also assume that there exists a TreapNode object, which stores a key, a priority, and references to its left and right children. Additionally, assume you have a helper function findRootPair(List⟨ EntryPair⟩ pairs) which takes in any list of key-priority pairs and returns the EntryPair with the minimum priority (hint: the root), where the input list is assumed to be non-empty. Your buildTreap method should return the root of the treap, in the form of a TreapNode.

   Your function must be implemented recursively. Hint: your approach will likely involve recurring on smaller and smaller subsets of the key-priority pairs. How will you divide the pairs?

   This is what the signature of buildTreap would look like if it were a Java method:

   ```
   public TreapNode buildTreap(List<EntryPair> pairs)
   ```

   Note: You don't need to write an insert method to add new pairs to an already-existing treap. Your pseudocode only needs to be able to build a new treap given all the pairs that will be stored inside it.

Problem 8.2

Fast Graduation

Silly premise: Ms.Norbury is a pusher. She pushes people. She pushed her husband into law school. That was a bust. She pushed herself into working three jobs. And now she's pushing Cady. Ever since Cady joined the Mathletes, Ms. Norbury and Kevin G. have been trying to make fetch happen and get her to graduate early. Help Cady fend them off and finish out her curriculum as fast as possible.

Suppose Cady's curriculum consists of n courses, all mandatory, all of them lasting one semester, and all of them offered every semester (i.e. the limit does not exist). The prerequisite structure of the curriculum can be organized in a graph, where each node is a course, and there's a directed edge from node A to node B if and only if A is a prerequisite for B. (Note that if A is a prereq for B and B for C, then that implicitly makes A a prereq for C. The arrow from A to C may or may not be included in the graph.) A course may have any number of prerequisites.

Write pseudocode for an algorithm that computes the minimum number of semesters needed for Cady to complete the curriculum. (She may take any number of courses in each semester). The run time of your algorithm should be $O(|V| + |E|)$.

2 Python Problems

Problem 8.3

Merge, Quick, and Radix Sort

Implement in Python the following sorting algorithms: merge sort, quick sort, and radix sort.

Requirements

Your job is to implement the sorts listed above by filling in the methods defined in the stencil code (in `sort.py`) so that, given an array of integers, each will return an array of the same integers sorted in **descending order**. If you sort in ascending order or sort in ascending order and then reverse the list, you will receive a 0. Also please note that you **may not** change the signature of any stencil method (doing so will result in no credit). You may also, of course, not call Python's built-in `sort` procedure.

Merge sort must run in worst case $O(n \log n)$ time.

Quick sort must run in expected case $O(n \log n)$ time.

Radix sort must run in worst case $O(dn)$ time where d is the number of digits in the largest number. Note that while we only discussed radix sort for positive integers in class, your solution must work for all integers! There are several elegant ways to accomplish this.

Radix Challenge! Try to make your radix sort as fast as possible!

Note: Make sure you throw `InvalidInputException`s if the input list is `None`. Empty lists are fine, though. You may consider them an already-sorted list.

How To Test Your Code

To test your code, add more assert statements to `sort_test.py`. Be sure to test all significant cases, as well as testing that your algorithms handle invalid inputs properly.

Sort Profiling

Now that you have your sorting algorithms, why not time them to observe their relative performances? We have provided among the install files two lists of 10,000 numbers (one per line). numlist1.txt has a truly random assortment of integers and numlist2.txt is partially sorted. We have also provided a profiler `sort_profiler.py`. This profiler runs all your sorts on each file and outputs the time it takes to run each.

Run the sort profiler and write a brief readme (in `profiler_readme.txt`) to discuss the differences between your relative sorting times, including the differences in timing for the two text files.