DSC 190 - Homework 02

Due: Wednesday, January 19

Write your solutions to the following problems by either typing them up or handwriting them on another piece of paper. Unless otherwise noted by the problem's instructions, show your work or provide some justification for your answer. Homeworks are due via Gradescope at 11:59 PM.

Programming Problem 1.

In a file named `bst_with_fcd.py`, create a class named `BinarySearchTree` which implements a binary search tree with the following methods:

- `.insert(key)`: insert a new node with the given key. Should take $O(h)$ time. Should allow duplicate keys. This method should return a `Node` object representing the node.
- `.delete(node)`: remove the given `Node` from the BST. Should take $O(h)$ time.
- `.query(key)`: return the `Node` object with the given key, if it exists; otherwise raise `ValueError`. Should take $O(h)$ time.
- `.floor(key)`: return the `Node` with the largest key which is $\leq$ the given key. If there is no such key, raise `ValueError`. Should take $O(h)$ time.
- `.ceil(key)`: return the `Node` with the smallest key which is $\geq$ the given key. If there is no such key, raise `ValueError`. Should take $O(h)$ time.

You can find starter code for this problem on the course webpage.

Programming Problem 2.

In a file named `min_heap.py`, implement a `MinHeap` class. Your class should have the following methods:

- `.min()`: return (but do not remove) minimum key
- `.decrease_key(i, key)`: reduce the value of node i's key to `key`
- `.insert(key)`: insert a new key, maintaining the heap invariant
- `.pop_min_key()`: remove and return the minimum key

You may assume that the keys are numbers.

Hint: a max heap was implemented in lecture. One solution is to modify its code to transform it into a min heap. There's another, cleverer approach. Is there an easy way to use a max heap to implement a min heap without changing the code of the max heap?

Programming Problem 3.

In a file named `online_median.py`, create a class named `OnlineMedian` which stores a collection of numbers and has which has two methods: `.insert(x)`, which inserts a number into the collection in $O(\log n)$ time, and `.median()` which computes the median of all numbers inserted so far in $\Theta(1)$ time.

If no numbers have been inserted so far, `.median()` should return `NaN`.

Problem 1. (Challenge)

Note: this problem is *not graded.* It's just a problem that you can think about it you're interested. Feel free to come to office hours to discuss it.

How would you design a class that maintains a collection of numbers and supports the following operations:

- `.insert(x)`: insert a new number into the collection;
- `.remove(x)`: remove a number from the collection;
- `.min_gap()`: return the minimum gap between any two numbers in the collection.

There is a solution that has $O(\log n)$ time, where n is the size of the collection. The solution makes use of balanced binary search tree(s).