DSC 190 - Discussion 02

Problem 1.

In lecture, we saw that inserting an element into an existing heap takes $\Theta(\log n)$ time in the worst case, where n is the number of elements currently in the heap. This means that if we start with an empty heap and insert n elements, the time taken in the worst case is $\Theta(n \log n)$. In this problem, we'll see that we can actually build a heap in $\Theta(n)$ time if we already have all of the elements to be inserted stored in an array.

a) Now suppose we have an array with n elements that we wish to turn into a heap. We will do this by calling `._push_down(i)` on each heap node, but in a particular order. We don't need to call it on the leaf nodes, as they are already as low as they can go. Instead, we'll start by calling `.push_down(i)` on the nodes at height 1, then nodes at height 2, and so on, going from right to left.

Implement this strategy in code.

b) Show that building a heap in this way takes $\Theta(n)$ time, where n is the length of the array.

Hint: $\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$

c) (Extra) Let's check that starting from an empty heap and inserting n elements one by one actually does take $\Theta(n \log n)$ time overall. This is a little trickier than it might seem, since n is changing as we insert elements. The first insert takes time roughly $c \log 1$ (for some constant c), the second takes time $c \log 2$, and so forth, until the last takes time $c \log n$. So the total time is:

$$c\,(\log 1 + \log 2 + \log 3 + \ldots + \log n)$$

Show that this is $\Theta(n \log n)$.

Hint: the upper bound is easier than the lower bound. For the lower bound, try splitting the sum in half and working with just the larger half.

Problem 2.

Describe a simple algorithm which takes in an array of size n and an integer parameter k and returns the k *most frequent* elements of the array. State the time complexity of your approach.

Example: given `[1, 9, 2, 4, 5, 2, 3, 4, 1, 1, 5]`, and $k = 3$, return 1, 2, and 5 (in no particular order).