Analysis of Algorithms
CS 375, Fall 2018
Homework 13
Due **AT THE BEGINNING OF CLASS** Wednesday, November 7

- **Reading Assignment:** Same as last time: From your textbook (Levitin), read Chapter 5 up to (and including) Section 5.3.
- *A general note:* When writing up your homework, please write neatly and explain your answers clearly, giving all details needed to make your answers easy to understand. Graders may not award credit to incomplete or illegible solutions. Clear communication *is* the point, on every assignment.

Exercises

1. **Curly, Mo, and Larry's Totally Excellent (?) Sorting Algorithm!** Professors Curly, Mo, and Larry of the Portland Institute of Technology (PIT) have proposed the following in-place sorting algorithm: First, sort the first two-thirds of the elements in the array. Next, sort the last two-thirds of the array. Finally, sort the first two-thirds again. (Notice that this algorithm is similar to Mergesort except that it uses three recursive calls rather than two and there is no merging step! As a consequence, this algorithm is very easy to implement!)

   The pseudocode is given below. Recall that the floor function, $\lfloor x \rfloor$, simply rounds down to the nearest integer. This is just used to compute the appropriate two-thirds and round to an integer so that we don't use non-integer indices into our array.

   ```
   Stooge-sort (A,i,j)
   begin
       if A[i] > A[j] then
           swap A[i] and A[j]
       if i + 1 ≥ j then
           return
       k = ⌊(j − i + 1)/3⌋.
       Stooge-sort(A, i, j − k)      Comment: Sort first two-thirds.
       Stooge-sort(A, i + k, j)      Comment: Sort last two-thirds.
       Stooge-sort(A, i, j − k)      Comment: Sort first two-thirds again!
   end
   ```

   (a) Give an informal but convincing explanation (not a rigorous proof by induction) of why the approach of sorting the first two-thirds of the array, then sorting the last two-thirds of the array, and then sorting again the first two-thirds of the array yields a sorted array. A few well-chosen sentences should suffice.

   (b) Find a recurrence for the worst-case running time of Stooge-sort. (Don't forget a base case!) To simplify your recurrence, you may assume each recursive call is on a portion of the array that is *exactly* two-thirds the length of the original array.

   (c) Next, solve the recurrence using the recursion-tree method. **Be sure to show all of your work.** Do not use the "Master Theorem." In your analysis, it may be convenient to choose n to be c^k for some fixed constant c. (The value of c that

you choose might not even be an integer! This may seem a bit strange, but it may significantly simplify the analysis!)

(d) How does the worst-case running time of Stooge-Sort compare with the worst-case running times of other sorting algorithms (Insertion sort, Mergesort, Quicksort, Bubblesort, Heapsort)?