Math 560 Homework 6 Fall 2020

Due: Tuesday, October 26

1. **The Index Problem** [4 pts]

   Let A be an array of n distinct integers where A is **already sorted** in ascending order. Our problem is to find an index i, $1 \leq i \leq n$, such that $A[i] = i$ or determine that no such i exists.

   Describe an algorithm for this problem with $O(\log n)$ worst case running time. You should give the algorithm (in clear English or in clear high-level pseudo-code) and briefly explain why the running time is $O(\log n)$ in the worst case.

2. **Recursive Calls**[1] [5 pts]

Professor Mae Trix has devised an algorithm for computing the Trixian function on two $n \times n$ matrices. (Nevermind what that function does - we only care about the algorithm!)

(a) Prof. Trix's first attempt at her algorithm has a worst-case running time described by the recurrence relation:

$$\begin{aligned} T(1) &= c \\ T(n) &= 8T(n/2) + cn^4 \end{aligned}$$

What is the big-O asymptotic runtime of this algorithm as a function of n? Show your work.

(b) By using some clever tricks, Prof. Trix has removed *half* of the recursive calls and now has an algorithm with a worst-case runtime described by the recurrence relation:

$$\begin{aligned} T(1) &= d \\ T(n) &= 4T(n/2) + dn^4 \end{aligned}$$

What is the big-O asymptotic runtime of this algorithm as a function of n? Show your work.

(c) Is the second algorithm asymptotically better than the first in this case? Briefly, what do you think is the reason for this outcome?

3. **Stooge Sort**[1] [6 pts]

Professors Curly, Mo, and Larry have proposed the following sorting algorithm:

- First sort the first two-thirds of the elements in the array.
- Next sort the last two-thirds of the elements in the array.
- Finally, sort the first two-thirds again.

The code is given below. Notice 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!

```
def Stooge-Sort(A, i, j):

    if A[i] > A[j]:
        swap A[i] and A[j]

    if i+1 >= j:
        return

    k = ⌊(j-i+1)/3⌋
    Stooge-Sort(A, i, j-k) # Sort the first two-thirds.
    Stooge-Sort(A, i+k, j) # Sort the last two-thirds.
    Stooge-Sort(A, i, j-k) # Sort the first two-thirds again!
```

(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 here.

(b) Find a recurrence relation for the worst-case runtime of Stooge-Sort. To simplify your recurrence relation, you may assume each of the recursive calls is on a portion of the array that is *exactly* two-thirds the length of the original array.

(c) Next, solve the recurrence relation using the work tree method. **Show all of your work**. In your analysis, it will be convenient to choose n to be a^k for some fixed constant a. (For example, we used $a = 2$ when analyzing the multiplication problem or Mergesort in class. Here you will want to use a different value of a. The value of a that you choose might not even be an integer! As we've seen in class, this is valid and allows us to significantly simplify the analysis.)

(d) How does the worst-case runtime of Stooge-Sort compare with the worst-case runtime of the other sorting algorithms that we've seen so far?

[1] Adapted from problem sets created by Harvey Mudd College CS Professor Ran Libeskind-Hadas.