CS 205: Homework Set 3

16:198:205 (Sections 4 - 6)

Complete each of the following problems to the best of your ability. Remember, you can't be graded on what you don't write down so (unless you are just making stuff up) something is better than nothing. Discussing problems between each other is fine, but your final writeup and work must be your own.

- **Algorithms and Sorting**
  Suppose you are given a sequence or array of n real numbers $(a_1, a_2, \ldots, a_n)$, all distinct. We are interested in sorting this list, and ideally sorting it as efficiently as possible / with minimal effort.

  We can summarize the total ordering information in terms of a permutation of the numbers 1 to n. For instance, given (a_1, a_2, a_3), a total ordering permutation of $(3, 1, 2)$ would tell us that $a_3 \leq a_1 \leq a_2$ and therefore that the sorted form of the list would be (a_3, a_1, a_2).

  1) How many possible total ordering permutations are there? How many ways can a list of n numbers be ordered?
  2) Argue that if you know or are given the total ordering permutation, then you can sort the list in linear time, without any additional comparisons or tests.
  3) Argue that if you don't know the total ordering permutation (or only know partial / incomplete information about it) you cannot sort the list without additional comparisons or tests.
  4) Taking this view, argue that any sorting algorithm must do *at least enough work* to determine the total ordering permutation of the list.
  5) If every element comparison (testing whether $a_i \leq a_j$) provides at most one bit of information, argue that you need at least on the order of $\ln(n!)$ many tests/comparisons to sort the list.
  6) Argue that for $n \geq 1$, $n! \leq n^n$.
  7) Argue that for $n \geq 2$, $n! \geq (n/2)^{n/2}$.
  8) What can you conclude about the asymptotic growth / order of $\ln(n!)$? What can you conclude about the minimal number of tests/comparisons needed in a universal sorting algorithm?
  9) Based on the prior result, argue that merge sort is, asymptotically, as or more efficient than any other sorting algorithm.

  Bonus) What are the tightest (simple) bounds you can achieve and verify on $\ln(n!)$?

- **Modular Arithmetic and Divisibility Rules**
  These problems concern divisibility in base-10. Let N be an n-digit base-10 number,

$$N = D_n D_{n-1} \ldots D_2 D_1. \tag{1}$$

  1) Argue that N can be checked for divisibility by 2 in constant time. Do you even have to read all the digits in?
  2) Argue that for any integer $k \geq 0$, the following holds:

$$10^k \equiv 1 \ (\textbf{mod } 3). \tag{2}$$

3) Argue from this that if N is divisible by 3, the sum of the digits $D_n + D_{n-1} + \ldots + D_2 + D_1$ must also be divisible by 3, and vice versa.

4) Use this fact to construct an algorithm for checking divisibility by 3 (in base-10). What are the base cases you have to deal with?

5) Estimate the big O complexity of summing the digits of an n-digit number. For an n-digit number N, how big can the digit sum be?

6) Estimate, as tightly as you can, the big O complexity of checking divisibility by 3 according to this algorithm. *Can you relate the complexity of running the algorithm to a smaller version of the same problem?*

7) Look up an algorithm for computing division and remainder, and compare the complexity of your algorithm to the complexity of simply dividing N by 3 and checking the resulting remainder. Which one is better? Why?

8) Show that for any k,

$$2^k \textbf{ (mod 3)} \equiv 1 \textbf{ if } k \textbf{ is even}, 2 \textbf{ if } k \textbf{ is odd} \tag{3}$$

and use this to suggest an algorithm for checking divisibility by base 2. Estimate its complexity.

9) What is the complexity of checking for divisibility by 3 in base 3?