CS 624: Analysis of Algorithmns

Assignment 3

Due: Wednesday, February 17, 2021

1. Exercise 1.1 from the Stirling's Formula handout.
2. Exercise 2.1 from the Stirling's Formula handout.
3. **Either** Exercise 2.2 **or** Exercise 3.1 from the Stirling's Formula handout.
4. Exercise 2.2 from the Lecture 5 handout (on page 2).

   Really big hint: this is essentially the same as an exercise from the previous homework assignment. You can use that result here without proving it again. But you *have to* explain your reasoning clearly.
5. Exercise 8.1-3 (page 194). Be careful here. The only hard part of this problem is understanding exactly what it is asking. Once you understand that, the solution should be pretty short.

   Here's some help in understanding what the problem is asking:

   "The $n!$ inputs" is just the book's way of referring to the $n!$ possible permutations of the n input numbers. Each of those permutations is thought of as one input to the sorting algorithm we are considering.

   We proved in class that any decision tree modeling a sorting algorithm for n numbers would have depth at least $Cn \log n$ (for some C). This was based on the fact that it had to have $n!$ leaves (each leaf corresponding to one of the "inputs").

   But that's only the *worst* case. Maybe it's the case that there is some comparison sort (modeled by a binary decision tree) which has the property that even though the tree (necessarily) has maximum depth $Cn \log n$, still there is some constant A (independent of n) such that half the leaves are at depth An or less.

   The first question in the problem is asking if this is possible. (And there are two other questions in the problem as well.)
6. Problem 8-5 (a-d only) (page 207).
7. Assume that $c \geq 0$. Assume you had some kind of super-hardware that, when given two lists of length n that are sorted, merges them into one sorted list, and takes only n^c steps.

   (a) Write down a recursive algorithm that uses this hardware to sort lists of length n.

   (b) Write down a recurrence to describe the run time.

   (c) For what values of c does this algorithm perform substantially better than $O(n \log n)$? Why is it highly implausible that this kind of super-hardware could exist for these values of c?
8. Problem 9-1 (page 224). For part (c), you can use "the best asymptotic average-case running time".