HW 5

Due on **Wednesday, October 7, 2020 at 10am CST**

CS/ECE 374: Algorithms & Models of Computation, Fall 2020 Version: **1.4**

Submission instructions as in previous homeworks.

13 (100 PTS.) That's so Hanoi-ing.

Consider the following variants of the Towers of Hanoi. For each of variant, describe an algorithm to solve it in as few moves as possible. **Prove** that your algorithm is correct. Initially, all the n disks are on peg 1, and you need to move the disks to peg 2. In all the following variants, you are not allowed to put a bigger disk on top of a smaller disk.

13.A. (30 PTS.) Hanoi 0: Suppose you are forbidden to move any disk directly between peg 1 and peg 2, and every move must involve (the third peg) 0. Exactly (i.e., not asymptotically) how many moves does your algorithm make as a function of n?

13.B. (30 PTS.) Hanoi 2: Suppose you are only allowed to move disks from peg 0 to peg 1, from peg 1 to peg 2, or from peg 2 to peg 0.

Provide an upper bound, as tight as possible, on the number of moves that your algorithm uses.

(One can derive the exact upper bound by solving the recurrence, but this is too tedious and not required here.)

13.C. (40 PTS.) Hanoi Bye Bye: Finally consider the disappearing Tower of Hanoi puzzle where the largest remaining disk will disappear if there is nothing on top of it. The goal here is to get all the disks to disappear and be left with three empty pegs (in as few moves as possible).

Provide an upper bound, as tight as possible, on the number of moves your algorithm uses.

14 (100 PTS.) Divide and Merger

Suppose you are given k sorted arrays $A_1, A_2, \ldots, A_k$ (potentially of different sizes). Let $n_i > 0$ be the size of the ith array A_i, for $i = 1, \ldots, k$, with $\sum_{i=1}^{k} n_i = n$. Assume that all the numbers in all the arrays are distinct. You would like to merge them into a single sorted array A of n elements.

14.A. (30 PTS.) Use a divide and conquer strategy to derive an algorithm that sorts the given sorted arrays in $O(n \log k)$ time, into one big happy sorted array.

14.B. (30 PTS.) In **MergeSort** we split the array of size N into two arrays each of size $N/2$, recursively sort them and merge the two sorted arrays. Suppose we instead split the array of size N into k arrays of size N/k each and use the merging algorithm in the preceding step to combine them into a sorted array. Describe the algorithm formally and analyze its running time via a recurrence. You do not need to prove the correctness of the recursive algorithm.

14.C. (40 PTS.) Describe an algorithm (not necessarily divide and conquer) for the settings of **(14.A.)** that works in $O(n + \sum_{i=1}^{k} n_i \log \frac{n}{n_i})$ time. Prove the correctness of your algorithm, Note that this is potentially an improved algorithm if the n_is are non-uniform. For example, if $n_i = n/2^i$, for $i = 1, \ldots, k$, then the overall running time is linear. One can verify (but you do not need to do it – it is not immediate) that $O(\sum_{i=1}^{k} n_i \log \frac{n}{n_i}) = O(n \log k)$. This implies that this algorithm is a strict improvement over **(14.A.)**.

15 (100 PTS.) Fowl business.

You were given a kettle of n birds, which look all the same to you. To decide if two birds are of the same species, you perform the following experiment – you put the two of them in a cage together. If they are friendly to each other, then they are of the same species. Otherwise, you separate them quickly before survival of the fittest kicks in.

15.A. (60 PTS.) Suppose that strictly more than half of the birds belong to the same species. Describe and analyze an efficient algorithm that identifies every bird among the n birds that belong to this dominant species.

15.B. (40 PTS.) Now suppose that there are exactly p species present in your kettle of n birds. and one species has a plurality: more birds belong to that species than to any other species. Present a procedure to pick out the birds from the plurality species as efficiently as possible (i.e., minimize the number of experiments you have to do as a function of n and p). Do *not* assume that $p = O(1)$.