CS230 Recitation/Assignment 03 - Math Tools

Recitation 02/05, Assignment Due 02/11

1 Recitation

1. (Ungraded activity) Continued icebreaking and knowing each other in recitations.
2. (Continued from class meeting) Prove that $f(n) = O(g(n)) \implies \ln f(n) = O(\ln g(n))$ if both $f(n)$ and $g(n)$ are **positive**, **non-decreasing**, and **always larger than** 1. Note that we cannot assume $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ exists.
3. Let $f(n) = (\log_3 n)^n$ and $g(n) = (\log_2 n)^n$.
   - *Disprove* that $f(n) = \Omega(g(n))$ by writing out the definition and showing that the claim leads to a contradiction.
   - Alternatively, disprove $f(n) = \Omega(g(n))$ by directly proving $f(n) = o(g(n))$.
4. (Stooge Sort). This is a weird sorting algorithm that works (i.e., correctly sorts any array of numbers).

Algorithm 1 `StoogeSort`(A, L, R)

Input: Array $A[1 \dots n]$; array positions L and R
Output: A with elements between positions L and R sorted in place

```
1: if R ≤ L then return                                   ▷ If the array has at most one element
2: else if A[L] > A[R] then                               ▷ If the leftmost element is larger than the rightmost element
3:     Swap(A, L, R)
4: end if
5: if R − L > 1 then
6:     x ← (R−L+1)/3                                      ▷ Determine the length of "a third of the array"; note that this might not be an integer
7:     StoogeSort(A, L, ⌈R − x⌉)                          ▷ Sort the first two thirds of the array
8:     StoogeSort(A, ⌊L + x⌋, R)                          ▷ Sort the last two thirds of the array
9:     StoogeSort(A, L, ⌈R − x⌉)                          ▷ Sort the first two thirds of the array again
10: end if
```

(a) Collectively simulate the algorithm using the following array of 6 numbers: $A = [4, 2, 6, 1, 5, 3]$.
How many times is `StoogeSort()` called in total?

(b) Write a recurrence relation to represent the running time of `StoogeSort()` on an array of n elements. You should think of the running time of `Swap()` as $O(1)$, not just 1 or 2 or 3. Your answer to this part should involve ceiling and/or floor functions.

(c) Now discard all ceiling/floor functions in part (4b). Solve the resulting recurrence relation using the tree method.

(d) Solve the same recurrence relation using the Master's Theorem (refer to the class meeting notes for the theorem statement.) Note that to use the theorem, we should first verify our recurrence relation satisfies some criteria.

2 Gradescope Assignment (Complete outside recitations)

1. (Arithmetico-geometric progressions). Recall that arithmetic and geometric progressions have the general forms

$$a, a+d, a+2d, a+3d, \ldots,$$

and

$$a, ar, ar^2, ar^3, \ldots,$$

respectively. Arithmetico-geometric progressions are just a hybrid of them, with a general form of

$$a, (a+d)r, (a+2d)r^2, (a+3d)r^3, \ldots$$

Assuming $r \neq 1$, evaluate $\sum_{i=0}^{k}(a+id)r^i$, i.e., the finite sum of the first $(k+1)$ terms of the above form. Two hints:

- You may want to review Thm 6.79 for the formula for $\sum_{k=0}^{n} ar^k$. In fact, how the formula was derived is probably more relevant than the formula itself.
- Your final formula may be several terms that look unable to be further simplified. That is normal, don't worry.

2. In recitation, we dropped the ceiling and floor functions from our recurrence relation without any valid reasoning. We now take a look at dealing with recurrence relations with ceiling/floor notations via an easier example: `MergeSort`.

   Recall that we usually represent the running time of `MergeSort` as

$$T(n) = 2T(\frac{n}{2}) + n, \quad T(1) = 1,$$

   where we conveniently assume n is even. This recurrence relation, in fact, is not even well-defined for values like $n = 3$ or $n = 7$, as it does not specify what happens when $\frac{n}{2}$ is not an integer. Assume for now that when n is odd, we just "pad" the input array by a dummy value,[1] so that the array can still be split into two identically lengthed subarrays. This makes the recurrence relation become

$$C(n) = 2C(\lceil\frac{n}{2}\rceil) + n, \quad S(1) = 1,$$

   where C stands for *ceiling*. Note that unlike $T(n)$, $C(n)$ is well-defined for all integer values of n.

   On the other hand, define another recurrence relation

$$F(n) = 2F(\lfloor\frac{n}{2}\rfloor) + n, \quad F(1) = 1,$$

   where F stands for *floor*. $F(n)$ is also well-defined for all integer values of n, but it does not represent any algorithm.

   (a) Using 1-2 sentences, briefly explain why that $F(n) = T(n) = C(n)$ for all n for which $T(n)$ is well-defined (i.e, for all n where n is a power of 2).

   (b) Using part (a), prove $F(n) = O(n \log n)$. You may directly use the fact that **all three recurrences are nondecreasing** without a proof, and the fact that $T(n) = O(n \log n)$ as we have done that in class.
   Hint: look at Assignment 2, Problem 1. That tells us that for any positive integer n, we can always find a $m \in (n, 2n]$ such that m is a power of 2. ***How can we use m to conclude $F(n) \leq T(2n)$?*** *And* ***why does that imply $F(n) = O(n \log n)$?***

   (c) Prove $C(n) \leq F(2n)$ for all n.
   Hint: reuse the m in part (b). This part should be very similar to part (b).

   (d) Using parts (b)-(c), prove $C(n) = O(n \log n)$.

[1] There are better approaches than this in reality. This is for the sake of our assignment.