CS140 - Assignment 1

Due: Sunday, Jan. 28 at 10pm

http://xkcd.com/399/

This problem set should be done with a *different* partner than you worked with last week. If you would like help finding a partner, reach out ASAP.

You must use LaTeX to format your solution; one person should upload the pdf to gradescope.

1. **[10 points]** The Geometric Series

   (a) Use induction on n to show that for all integers $n \geq 0$

   $$1 + a + a^2 + a^3 + \ldots + a^n = \frac{a^{n+1} - 1}{a - 1}$$

   where a is some arbitrary real number other than 1. Please make sure to write your proof carefully, with a base case, induction hypothesis, induction step, and conclusion.
   ANSWER:
   Base case: We show that it's true for $n = 0$.
   $1 = \frac{a^{0+1}-1}{a-1} = \frac{a-1}{a-1} = 1$ which is true since when n = 0, the number of terms in the summation is the first term, $(1 = a^0)$.
   Inductive case: Assume that $n = k-1$ is true, that is, $1+a+a^2+a^3+\ldots+a^{k-1} = \frac{a^k-1}{a-1}$ and we need to prove that the same case holds for $n = k$, that is,
   $1 + a + a^2 + a^3 + \ldots + a^{k-1} = \frac{a^{k+1}-1}{a-1}$
   We start at $\sum_{i=0}^{k} a^i$ and we want to end up here: $\frac{a^{k+1}-1}{a-1}$

   $$\begin{aligned}
   \sum_{i=0}^{k} a^i &= a^k + \sum_{i=1}^{k-1} a^i \quad \text{using the inductive hypothesis.} \\
   &= \frac{a^k(a-1) + a^k - 1}{a-1} \\
   &= \frac{a^{k+1} - a^k + a^k - 1}{a-1} \\
   &= \frac{a^{k+1} - 1}{a-1} \quad \text{which is where we wanted to end up.}
   \end{aligned}$$

   Therefore we have proved that

   $$1 + a + a^2 + a^3 + \ldots + a^n = \frac{a^{n+1} - 1}{a - 1}$$

   using induction.

(b) Explain where your induction proof relied on the fact that $a \neq 1$.
Our induction proof relies on the fact that $a \neq 1$ at the base case. The validity of our inductive argument hinges on the precondition that $a \neq 1$. This is critical because a value of $a = 1$ would result in a division by zero within the base case, rendering it undefined. Such a scenario would undermine the entire proof, as the inductive step depends on the base case being well-defined to propagate the hypothesis for subsequent iterations.

(c) What does the sum evaluate to when $a = 1$?
The sum ends up being $1 + 1 + 1^2 + 1^3 + \ldots + 1^n = 1 * n = n$

2. [**40 points**] Asymptotics

Indicate whether each of the following statements is true or false and then carefully **prove** your answer using the formal definition of Big-O notation and properties of logarithms from the last assignment (hint, hint). Remember that to show that one of these statements is false, you must obtain a formal contradiction; it does not suffice to just say "false".

One useful summation:

$\Sigma_{i=1}^{n} \frac{1}{i} = \ln n + c$

(a) 2^{n+1} is $O(2^n)$.
Answer = True:
We need to show that 2^{n+1} is $O(2^n)$.
By the definition of big-O:

$$\begin{aligned} 2^{n+1} &\leq c2^n \\ c2^n &\geq 2^{n+1} \\ c &\geq \frac{2^{n+1}}{2^n} \\ c &\geq 2 \end{aligned}$$

So as long as $c \geq 2$, 2^{n+1} is $O(2^n)$.

(b) 2^{2n} is $O(2^n)$.
Answer = False:
We need to show that 2^{2n} is $O(2^n)$.
By the definition of big-O:

$$\begin{aligned} 2^{2n} &\leq c2^n \\ c2^n &\geq 2^{2n} \\ c &\geq \frac{2^{2n}}{2^n} \\ c &\geq n \end{aligned}$$

There cannot be a c where we can choose an n_0 where this works since $c \geq n$ is a contradiction.

(c) $3n^2 \log_2 n + 16n$ is $O(n^3)$.

Answer = True:

We need to show that $3n^2 \log_2 n + 16n$ is $O(n^3)$.
By the definition of big-O:

$$\begin{aligned} 3n^2 \log_2 n + 16n &\leq cn^3 \\ cn^3 &\geq 3n^2 \log_2 n + 16n \\ c &\geq \frac{3 \log_2 n}{n} + \frac{16}{n^2} \end{aligned}$$

Let's choose $n_0 = 2$ therefore:

$$\begin{aligned} c &\geq \frac{3 \log_2 2}{2} + \frac{16}{4} \\ c &\geq 1.5 + 4 \\ c &\geq 5.5 \end{aligned}$$

So we've found a c and n_0 that satisfy the big-O requirement thus $3n^2 \log_2 n + 16n$ is $O(n^3)$.

(d) $25 \log_2 8n^{10}$ is $O(\log_{10} n)$.

Answer = True:

We need to show that $25 \log_2 8n^{10}$ is $O(\log_{10} n)$.
By the definition of big-O:

$$\begin{aligned} 25 \log_2(8n^{10}) &\leq c \log_{10} n \\ c \log_{10} n &\geq 25 \log_2(8n^{10}) \\ c &\geq \frac{25 \log_2(8n^{10})}{\log_{10} n} \\ c &\geq \frac{\frac{25 \log(8n^{10})}{\log 2}}{\log n} \\ c &\geq \frac{83.05(\log 8 + 10 \cdot \log(n^{10}))}{\log n} \\ c &\geq \frac{83.05 \cdot \log 8 + 83.05 \log(n^{10})}{\log n} \\ c &\geq 830.5 + \frac{75}{\log n} \end{aligned}$$

Let's choose $n_0 = 10$ therefore $c \geq 830.5 + 75 = 905.5$. So we've found a (range of) c(s) and n_0 that satisfy the big-O requirement thus $25 \log_2 8n^{10}$ is $O(\log_{10} n)$.

(e) $8^{\log_2 n}$ is $O(n^3)$.

Answer = True:

We can simplify $8^{\log_2 n}$ as $n^{log_2 8} = n^3$.

We need to show that n^3 is $O(n^3)$

$$\begin{aligned} n^3 &\leq cn^3 \quad \text{by the definition of big-O} \\ cn^3 &\geq n^3 \\ c &\geq 1 \end{aligned}$$

So as long as $c \geq 1$, $8^{\log_2 n}$ is $O(n^3)$ for any n_0.

3. **[12 points]** Solving Recurrences

Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible (in other words, give Θ bounds where possible) and make sure to justify your answers. (Note: you may want to read appendix A.1 in the textbook, which has some summation properties.)

(a) $T(n) = 4T(n/2) + cn$

The recurrence $T(n) = 4T(n/2) + cn$ follows the pattern of the Master Theorem as follows:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

We have:

- $a = 4$,
- $b = 2$,
- $f(n) = cn$.

Then, by the Master Theorem states the the solution to the recurrence must be in one of the 3 following cases:

- $f(n)$ is $n^{\log_b a}$ is $O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$,
- $f(n)$ is $\Theta(n^{\log_b a})$,
- $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$ for some $\varepsilon > 0$, and if $af(n/b) \leq kf(n)$ for some constant $k < 1$ and sufficiently large n, then $T(n) = \Theta(f(n))$.

Now $n^{\log_b a} = n^{\log_2 4} = n^2$. For the three cases, we have to ask ourselves:

- Is $cn = O(n^{2-\varepsilon})$?
- Is $cn = \Theta(n^2)$?
- Is $cn = \Omega(n^{2+\varepsilon})$?

We see that for small ε, cn is not bounded below by $\Omega(n^{2+\varepsilon})$. Say we choose $\varepsilon = 0.01$, so $\Omega(n^{2.01})$ which is above or bigger than cn. However we see that for small ε, $cn = O(n^{2-\varepsilon}) = O(n^{1}.99)$, thus cn is bounded above by $O(n^{2-\varepsilon})$. This means that the recurrence relation, $T(n) = \Theta(n^2)$ as per case 1's conclusion.

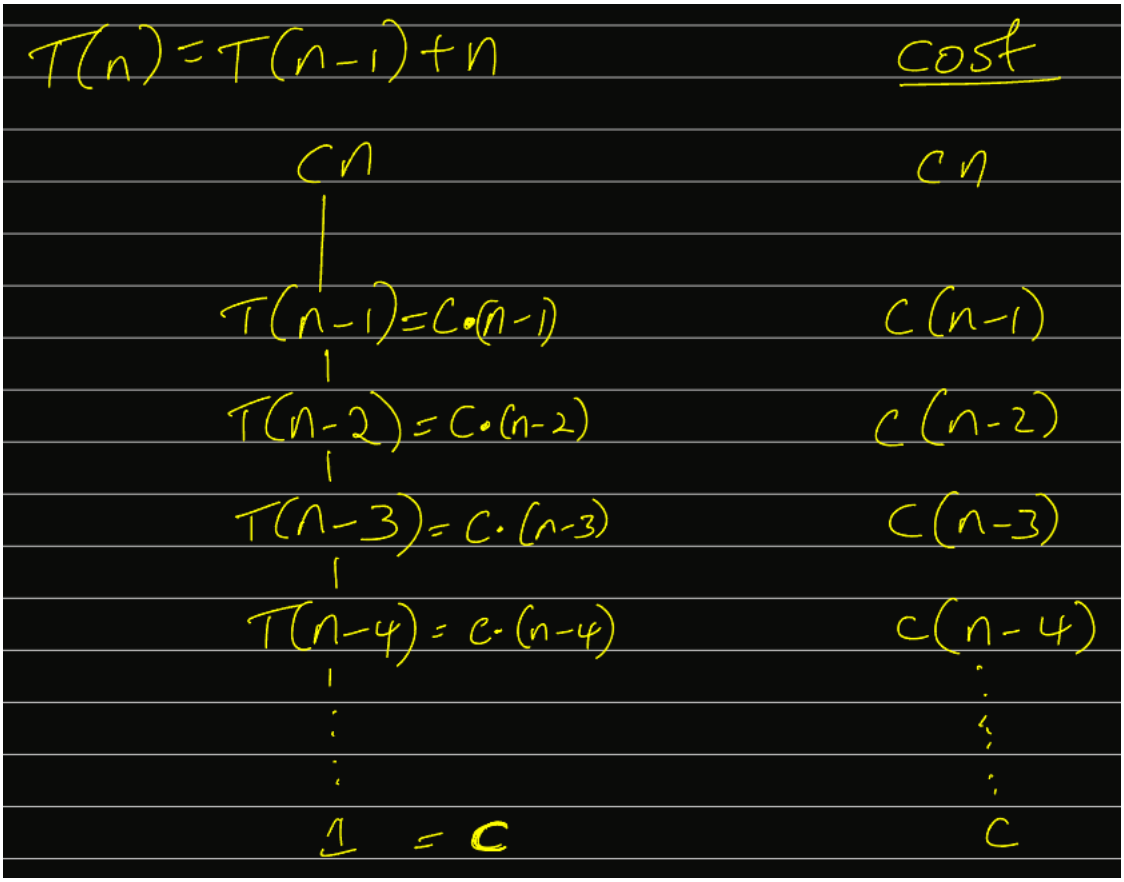

Figure 1: Recurrence tree of 3 (b)

(b) $T(n) = T(n-1) + n$

The figure above shows the tree we constructed from the recurrence tree method:
Now, as per the recurrence tree method, we need to find the total cost:

$T(n) = cn + c(n-1) + c(n-2) + c(n-3) + \cdots + c$

where the term in brackets in the following line is a well-known series (the partial sums)

$T(n) = c[n + (n-1) + (n-2) + (n-3) + \cdots + 1]$

$T(n) = \dfrac{cn(n+1)}{2}$

$T(n)$'s total cost thus evaluates to $\Theta(n^2)$.

(c) $T(n) = T(n-1) + 1/n$

We can solve this using the Recursion tree method.
$T(4) = T(3) + \frac{1}{4}$
$T(3) = T(2) + \frac{1}{3}$
$T(2) = T(1) + \frac{1}{2}$
$T(1) = T(0) + 1$
$T(0) = 0$
and so we can see that:
$T(n) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + ...\frac{1}{n}$
The sum is a harmonic series, which has been provided to us in this homework:

$$T(n) = \sum_{i=2}^{n} \frac{1}{i}$$

The harmonic series grows logarithmically, hence:

$$T(n) = \ln(n) + C$$

for some constant C, which leads us to conclude that the run time is:

$$T(n) = \Theta(\ln(n))$$

(d) $T(n) = T(9n/10) + n$

The recurrence $T(n) = T\left(\frac{9n}{10}\right) + n$ follows the pattern of the Master Theorem as follows:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Next, we arrange our recurrence function to fit this pattern. Then, $T(n) = T\left(\frac{n}{10/9}\right) + n$, we have:

- $a = 1$,
- $b = \frac{10}{9}$,
- $f(n) = n$.

Then, by the Master Theorem states the the solution to the recurrence must be in one of the 3 following cases:

- $f(n)$ is $n^{\log_b a}$ is $O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$,
- $f(n)$ is $\Theta(n^{\log_b a})$,
- $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$ for some $\varepsilon > 0$, and if $af(n/b) \leq kf(n)$ for some constant $k < 1$ and sufficiently large n, then $T(n) = \Theta(f(n))$.

Now, $\log_b a = \log_{\frac{10}{9}} 1 = 0$. This is because $\log 1 = 0$. Since $f(n) = n$, we can see that $f(n)$ is larger than $n^{\log_b a} = n^0 = 1$ so it cannot act as an upper bound. Thus, $f(n) = n$ is $\Omega(n^{\log_b a + \varepsilon})$ for $\varepsilon > 0$. Thus, we see that this would fall under Case 3 because n is lower bounded. Next, we prove that $af(n/b) \leq kf(n)$ for some constant $k < 1$ and sufficiently large n:

- $\frac{9n}{10} \leq kn)$
- Let $k = \frac{9.1}{10}$
- Then, $\frac{9n}{10} = \frac{9.1n}{10})$
- Thus, we can see that we found a value of k such that it is less than 1, and $af(n/b) \leq kf(n)$.

Thus, by the third case of the Master Theorem, we conclude:

$$T(n) = \Theta(f(n)) = \Theta(n)$$

Hence solution to the recurrence $T(n) = T\left(\frac{9n}{10}\right) + n$ is $T(n) = \Theta(n)$ according to the Master Theorem.

4. **[3 points]** An improvement?

   You decide to come up with a new sorting algorithm called <*Arsum+Collins*>sort. You notice that *InsertionSort* seems inefficient in how it finds the correct location to insert the next value. You decide that rather than linearly searching one at a time to find the correct place, you use binary search to find the correct place. Is this an improvement? If yes, state the running time of this new algorithm. If no, explain why this is not an improvement. Be specific and clear.

   ANSWER: No, this is not an improvement. Let's say the name is 'n' characters long such that there are n characters to sort. The running time for search with the binary search is $O(log(n))$ while the running time with search for the insertion sort was $O(n^2)$. So the searching algorithm run time has improved. However, the insertion step still takes $O(n)$ and so the over all time complexity remains the same as the original Insertion sort of $O(n^2)$.

5. **[5 points]** Intersection

   Describe an algorithm and *state the worse case running time* (in terms of Θ or O where appropriate) to solve the following problem: given two lists of numbers A and B of lengths m and n respectively, return the intersection of the lists, i.e. all those numbers in A that also occur in B. You can use procedures that we've discussed in class, but no others (e.g. no hashtables). You can assume that in any given list, the numbers are unique. You will be graded on the efficiency of your solutions.

ANSWER: Initially, use merge sort to sort list A. Merge sort always takes $O(nlogn)$ run time and so in this case the run time will be $O(mlogm)$ with m being the length of the list A. Now, for each element of B, we do a Binary Search Tree on A (already ordered and all the numbers are unique) and this gives us a run time of $O(nlogm)$ where m is the length of the list A and n is the number of elements of of B that we need to search for in A.

Then, the worst run time complexity is: $O(mlogm + nlogm)$