CS 170 Homework 1

Due **Tuesday 9/5/2023, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write "none".

Solution: None.

2 Course Policies

(a) What dates and times are the exams for CS170 this semester? Are there planned alternate exams?

Note: We will make accommodations for students in faraway timezones.

Solution: Two midterms: Tuesday Oct. 3, 2023 from 7-9PM and Midterm 2 on Nov. 7, 2023 also from 7-9PM. Final is on Friday Dec. 15, 2023 from 8-11AM. There are no planned alternate or online exams.

(b) Homework is due Mondays at 10:00pm, with a late deadline at 11:59pm. At what time do we recommend you have your homework finished?

Solution: It's recommended that homeworks are done by 10:00pm; the extra two hours past 10 are reserved for technical issues.

(c) We provide 2 homework drops for cases of emergency or technical issues that may arise due to homework submission. If you miss the Gradescope late deadline (even by a few minutes) and need to submit the homework, what should you do?

Solution: There are two automatic homework drops that are specifically designed for this reason. Since there are no exceptions to this rule, if you miss the late deadline, I don't think there's anything you can do besides receive the 0.

(d) What is the primary source of communication for CS170 to reach students? We will send out all important deadlines through this medium, and you are responsible for checking your emails and reading each announcement fully.

Solution: Ed discussion.

(e) Please read all of the following:

(i) **Syllabus and Policies:** https://cs170.org/syllabus/
(ii) **Homework Guidelines:** https://cs170.org/resources/homework-guidelines/
(iii) **Regrade Etiquette:** https://cs170.org/resources/regrade-etiquette/
(iv) **Forum Etiquette:** https://cs170.org/resources/forum-etiquette/

Once you have read them, copy and sign the following sentence on your homework submission.

"I have read and understood the course syllabus and policies."

Solution: I have read and understood the course syllabus and policies.

3 Understanding Academic Dishonesty

Before you answer any of the following questions, make sure you have read over the syllabus and course policies (https://cs170.org/syllabus/) carefully. For each statement below, write *OK* if it is allowed by the course policies and *Not OK* otherwise.

(a) You ask a friend who took CS 170 previously for their homework solutions, some of which overlap with this semester's problem sets. You look at their solutions, then later write them down in your own words.

Solution: Not ok, since that friend wasn't cited. You should also never be asking for solution from past semester's work.

(b) You had 5 midterms on the same day and are behind on your homework. You decide to ask your classmate, who's already done the homework, for help. They tell you how to do the first three problems.

Solution: Not ok.

(c) You look up a homework problem online and find the exact solution. You then write it in your words and cite the source.

Solution: OK. As long as you have understood what the solution wants you to do, I see no real issue in doing this.

(d) You were looking up Dijkstra's on the internet, and inadvertently run into a website with a problem very similar to one on your homework. You read it, including the solution, and then you close the website, write up your solution, and cite the website URL in your homework writeup.

Solution: This seems to imply also that you've understood the steps to arrive at the solution, so probably ok.

4 Math Potpourri

The following subparts will cover several math identities, tricks, and techniques that will be useful throughout the rest of this course.

(a) Simplify the following expressions into a single logarithm (i.e. in the form $\log_a b$):

(i) $\frac{\ln x}{\ln y}$

Solution: Using the result that $\log_a b = \frac{\log_c b}{\log_c a}$ in reverse, we get:

$$\frac{\ln x}{\ln y} = \log_x y$$

(ii) $\ln x + \ln y$

Solution: Converting product to sum:

$$\ln x + \ln y = \ln xy$$

(iii) $\ln x - \ln y$

Solution: Same thing as part (c), except now we're doing division instead of multiplication:

$$\ln x - \ln y = \ln\left(\frac{x}{y}\right)$$

(iv) $170 \ln x$

Solution: Use the property that $a \log b = \log b^a$, we get:

$$170 \ln x = \ln x^{170}$$

(b) Give a simple proof for each of the following identities:

(a) $x^{\log_{1/x} y} = \frac{1}{y}$

Solution: First use the fact that $\log_{a^b} c = \frac{1}{b} \log_a c$, we can simplify the exponent:

$$x^{\log_{1/x} y} = x^{-\log_x y}$$

Then we can just use rules of exponentiation, combined with the fact that $a^{\log_a b} = b$:

$$x^{-\log_x y} = (x^{\log_x y})^{-1} = y^{-1} = \frac{1}{y}$$

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

Solution: We can use induction, with base case $n = 1$:

$$1 = \frac{1(2)}{2} = 1 \checkmark$$

Now for our inductive hypothesis, let's assume that $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$. Then, we prove this for the summation to $n + 1$:

$$\begin{aligned}\sum_{i=1}^{n+1} i &= \sum_{i=1}^{n} i + (n+1) \\ &= \frac{n(n+1)}{2} + (n+1) \\ &= \frac{n(n+1) + 2(n+1)}{2} \\ &= \frac{n^2 + 3n + 2}{2} \\ &= \frac{(n+1)(n+2)}{2}\end{aligned}$$

as desired.

(c) $\sum_{k=0}^{n} ar^k = \begin{cases} a\left(\frac{1-r^{n+1}}{1-r}\right), & r \neq 1 \\ a(n+1), & r = 1 \end{cases}$

Solution: The case where $r = 1$ is trivial: the summation decomposes immediately:

$$\sum_{k=0}^{n} a(1)^k = a\sum_{k=0}^{n} 1^k = a(1^0 + 1^1 + \cdots + 1^n) = a(n+1)$$

The case where $r \neq 1$ effectively requires us to prove the geometric series formula. We can do this using induction, starting with the base case of $n = 0$:

$$\sum_{k=0}^{0} ar^k = ar^0 = a$$

the right hand side gives:

$$a\left(\frac{1-r}{1-r}\right) = a$$

so the base case is verified. Now let this summation be true for some arbitrary n. We now show it is true for $n + 1$:

$$\begin{aligned}\sum_{k=0}^{n+1} ar^k &= \sum_{k=0}^{n} ar^k + ar^{n+1} \\ &= a\left(\frac{1-r^{n+1}}{1-r}\right) + ar^{n+1} \\ &= a\left[\frac{1-r^{n+1} + (1-r)r^{n+1}}{1-r}\right] \\ &= a\left(\frac{1-r^{n+2}}{1-r}\right)\end{aligned}$$

as desired. Hence, we've proven:

$$\sum_{k=0}^{n} ar^k = a\left(\frac{1-r^{n+1}}{1-r}\right)$$

for $r \neq 1$.

5 Recurrence Relations

For each part, find the asymptotic order of growth of T; that is, find a function g such that $T(n) = \Theta(g(n))$. Show your reasoning and do not directly apply any master theorems.
In all subparts, you may ignore any issues arising from whether a number is an integer.

(a) $T(n) = 3T(n/4) + 10n$

Solution: I drew a tree out on my iPad but I can't be bothered to copy it over (I don't want to draw in LaTeX), but we can imagine that every layer, we have 3 subproblems of size $\frac{n}{4}$, and the cost for each computation is $10n$. Therefore, if we follow the process from lecture, we have:

$$T(n) = \sum_{i=0}^{\log_4 n} 3^i \cdot 10\left(\frac{n}{4^i}\right) = 10n \sum_{i=0}^{\log_4 n} \left(\frac{3}{4}\right)^i$$

The summation is a geometric series where $r < 1$, and according to the lecture, this portion has runtime $\Theta(1)$, so $T(n) = 10n \cdot \Theta(1) \in \Theta(n)$.

(b) $T(n) = 97T(n/100) + \Theta(n)$

Solution: Just like the previous problem, except here we have 97 problems subproblems of size $\frac{n}{100}$, and the cost for each layer of the tree is on the order of $\Theta(n)$. I can convert this into O notation by writing the following:

$$T(n) = 97T\left(\frac{n}{100}\right) + cn$$

Therefore, our recursion relation is:

$$T(n) = \sum_{i=0}^{\log_{100} n} 97^i \cdot c \cdot \left(\frac{n}{100^i}\right) = cn \sum_{i=0}^{\log_{100} n} \left(\frac{97}{100}\right)^i$$

And since $\frac{97}{100} < 1$, then the summation is again $\Theta(1)$, so we have:

$$T(n) = cn \cdot \Theta(1) \in \Theta(n)$$

(c) An algorithm $\mathcal{A}$ takes $\Theta(n^2)$ time to partition the input into 5 sub-problems of size $n/5$ each and then recursively runs itself on 3 of those subproblems. Describe the recurrence relation for the run-time $T(n)$ of $\mathcal{A}$ and find its asymptotic order of growth.

Solution: Even though this program $\mathcal{A}$ divides into 5 subproblems, the fact that we only need to run on 3 of the 5 means that in reality, this program really just divides into 3 subproblems of size $\frac{n}{5}$. This means our recursive relation is:

$$T(n) = 3T\left(\frac{n}{5}\right) + \Theta(n^2)$$

This means our recursion relation simplifies to:

$$T(n) = \sum_{i=0}^{\log_5 n} 3^i \cdot c\left(\frac{n^2}{5^i}\right) = cn^2 \sum_{i=0}^{\log_5 n} \left(\frac{3}{5}\right)^i$$

And again since the geometric series evaluates to $\Theta(1)$, then we have:

$$T(n) = cn^2 \cdot \Theta(1) \in \Theta(n^2)$$

(d) $T(n) = 3T(n/3) + \Theta(n)$

Solution: Here, just like the previous two subproblems:

$$T(n) = \sum_{i=0}^{\log_3 n} 3^i \left(\frac{n}{3^i}\right) = n \sum_{i=0}^{\log_3 n} 1^i$$

Again using the geometric series, this geometric series, since $r = 1$, is $\Theta(\log_3 n)$. This means that in total, we have:

$$T(n) = n \cdot \Theta(\log_3 n) \in \Theta(n \log n)$$

(e) $T(n) = T(3n/5) + T(4n/5)$ (We have $T(1) = 1$)
Hint: Try to guess a $T(n)$ of the form an^b and then use induction to argue that it is correct.

Solution: Here we can bound the runtime of $T(n)$ from above and also from below. Based on the partitions, we know that

$$T(n) \leq 2T\left(\frac{4n}{5}\right)$$

This splits our input into two smaller subproblems, each of size $\frac{4n}{5}$. Analyzing the runtime of this upper bound, we find that it terminates in $\log_{5/4} n$. Since we divide into two subproblems at every step, the total amount of work is:

$$T(n) \leq \sum_{i=0}^{\log_{5/4} n} 2^i \in \Theta\left(2^{\log_{5/4} n}\right)$$

Bounding from below, we have the relation

$$T(n) \geq 2T\left(\frac{3n}{5}\right)$$

Using the same logic as the upper bound, we find that this process terminates in $\log_{5/3} n$ steps. Applying the same geometric series formula, we have:

$$T(n) \geq \sum_{i=0}^{\log_{5/3} n} 2^i \in \Theta\left(2^{\log_{5/3} n}\right)$$

Therefore, we have:

$$\Theta\left(2^{\log_{5/3} n}\right) \leq T(n) \leq \Theta\left(2^{\log_{5/4} n}\right)$$

We can then use the relation that $a^{\log_b c} = c^{\log_b a}$ to swap the 2 and n on the left and right, giving:

$$\Theta\left(n^{\log_{5/3} 2}\right) \leq T(n) \leq \Theta\left(n^{\log_{5/4} 2}\right)$$

Evaluating the loagirhtms by computer, we get:

$$\Theta\left(n^{1.3}\right) \leq T(n) \leq \Theta\left(n^{3.1}\right)$$

This motivates us to choose $T(n) \in \Theta(n^2)$. We now proceed by induction to prove this result. First, we can use the base case: $T(1) = 1 \in \Theta(1)$, so the base case is verified. For our inductive hypothesis, let's assume that for all numbers $\{1, 2, \ldots, k\}$, $T(k) \in \Theta(k^2)$. We now prove that $T(k+1) \in \Theta((k+1)^2)$. To do so, let's first write out $T(k+1)$:

$$T(k+1) = T\left(\frac{3(k+1)}{5}\right) + T\left(\frac{4(k+1)}{5}\right)$$

Now we use our inductive hypothesis to get:

$$T(k+1) = \Theta\left(\frac{9(k+1)^2}{25}\right) + \Theta\left(\frac{16(k+1)^2}{25}\right)$$

And since we don't care about constants, both of these can be combined into one term, where we can write:

$$T(k+1) = \Theta((k+1)^2)$$

Therefore completing the proof. Thus $T(n) \in \Theta(n^2)$.

6 In Between Functions

In this problem, we will find a function $f(n)$ that is asymptotically worse than polynomial time but still better than exponential time. In other words, f has to satisfy two things,

- For all constants $k > 0$, $f(n) = \Omega(n^k)$ (1)
- For all constants $c > 0$, $f(n) = O(2^{cn})$ (2)

(a) Try setting $f(n)$ to a polynomial of degree d, where d is a very large constant. So $f(n) = a_0 + a_1 n + a_2 n^2 \cdots + a_d n^d$. For which values of k (if any) does f fail to satisfy (1)?

Solution: Let $f(n)$ be defined as in the prbolem, and let $g(n) = n^k$. In order for $f(n)$ to satisfy (1), we require that $g(n) \in O(f(n))$. Let's take the ratio of these two:

$$\frac{g(n)}{f(n)} = \frac{n^k}{a_0 + a_1 n + \cdots + a_d n^d}$$

Let's factor out n^d from the denominator, giving us:

$$\frac{g(n)}{f(n)} = \frac{n^k}{n^d\left(\frac{a_0}{n^d} + \frac{a_1}{n^{d-1}} + \cdots + a_d\right)} = \frac{n^{k-d}}{\frac{a_0}{n^d} + \cdots + a_d}$$

Now let's look at the case where $k < d$. This means that $k - d < 0$, hence the numerator has a negative exponent. Taking the limit as $n \to \infty$:

$$\lim_{n\to\infty} \frac{n^{k-d}}{\frac{a_0}{n^d} + \cdots + a_d} = 0$$

Since the limit tends toward zero, then this can be bounded. However, with $k > d$ we have $k - d > 0$, hence the exponent on the numerator is *positive*, so it is an *increasing* function. This implies:

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{n^{k-d}}{\frac{a_0}{n^d} + \cdots + n^d} = \infty$$

Since the limit is ∞, then this cannot be bounded, hence $g(n) \notin O(f(n))$. Thus, the values of k that fail condition (1) are those where $k > d$.

(b) Now try setting $f(n)$ to a^n, for some constant a that's as small as possible while still satisfying (1) (e.g. 1.000001). For which values of c (if any) does f fail to satisfy (2)?
Hint: Try rewriting a^n as 2^{bn} first, where b is a constant dependent on a.

Solution: We do the same thing as the previous part. Let $f(n) = a^n$, and $g(n) = 2^{cn}$, for the purposes of this subpart. To satisfy (2), we want $f(n) \in O(g(n))$, so let's take the ratio $\frac{f}{g}$:

$$\frac{f(n)}{g(n)} = \frac{a^n}{2^{cn}}$$

Now we take the limit as $n \to \infty$:

$$\lim_{n\to\infty} \frac{a^n}{2^{cn}} = \lim_{n\to\infty} \left(\frac{a}{2^c}\right)^n$$

To satisfy (2), we want this limit to equal zero. The only way this can happen is if the ratio $\frac{a}{2^c} < 1$, so in other words:

$$a < 2^c \implies \log_2 a < c$$

Thus, if $c > \log_2 a$, then f fails to satisfy (2).

So far we have found that the functions which look like $O(n^d)$ for constant d are too small and the functions that look like $O(a^n)$ are too large even if a is a tiny constant.

(c) Find a function $D(n)$ such that setting $f(n) = O(n^{D(n)})$ satisfies both (1) and (2). Give a proof that your answer satisfies both.
Hint: Make sure $D(n)$ is asymptotically smaller than n.

Solution: Of course, naturally, let's pick $D(n) = \log n$, so we have $f(n) = O(n^{\log n})$. Let's start with condition 1, let $g(n) = n^k$. Now we compute $\frac{g}{f}$:

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \frac{n^k}{n^{\log n}} = \lim_{n\to\infty} n^{k-\log n}$$

Note that since k is a constant and $\log n$ is increasing, then there will be a point where $\log n > k$, and the exponent is negative. Hence, this limit does approach zero, and we can conclude that $f(n)$ grows asymptotically faster than $g(n)$, hence $g(n) \in O(f(n))$.

Now moving on to condition (2). Now let $g(n) = 2^{cn}$. We want $g(n)$ to grow faster than f, so let's take the ratio $\frac{f}{g}$:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^{\log n}}{2^{cn}} = \lim_{n\to\infty} \frac{(2^{\log_2 n})^{\log n}}{2^{cn}} = \lim_{n\to\infty} \frac{2^{\log_2 n \cdot \log n}}{2^{cn}} = \lim_{n\to\infty} (1)^{\log_2 n \log n - cn} = 1$$

Since the limit equals 1, then we say that $f(n) \in \Theta(g(n))$, meaning that $f(n) \in O(g(n))$.

7 [Coding] Decimal to Binary

Given the n-digit decimal representation of a number, converting it into binary in the natural way takes $O(n^2)$ steps.

(a) Give a divide-and-conquer algorithm to do the conversion in $O(n^{\log_2 3} \log n)$ time.

Hint: refer to lecture slides and take inspiration from Karatsuba's algorithm!

Solution: We'll take inspiration from Karatsuba's and partition our number into two portions. Let the number be represented by $d = d_n d_{n-1} \dots d_0$. Then, we can partition this into the following:

$$d = (d_n d_{n-1} \dots d_{\frac{n}{2}}) \cdot 10^{\frac{n}{2}} + (d_{\frac{n}{2}-1} \dots d_0)$$

Then our goal is to convert these three numbers of length (roughly) $\frac{n}{2}$ into binary, which is our recursive call. This process is carried out all the way down to single digits, from which we can convert very easily. Therefore, our process essentially boils down to the following:

- Split n into two halves, $d_n d_{n-1}, \dots, d_{\frac{n}{2}}$ and $d_{\frac{n}{2}-1} \dots, d_0$, and multiply the first half by $10^{\frac{n}{2}}$, the exponent is specified by the length of the second half.
- Convert both halves of n into binary (this is our recursive call) and also convert $10^{\frac{n}{2}}$ into binary.
- At each layer: multiply together the binary representation of the first half with the binary of $10^{\frac{n}{2}}$, then add this result to the binary representation of the second half.

(b) Write out the recurrence for your algorithm from part (a) and use it to derive the desired runtime.

Solution: We see that our algorithm splits up a problem of size n into three sub-problems of size $\frac{n}{2}$. The cost per layer is a multiplication and an addition. The addition step is $O(n)$, and we learned from class that the multiplication (with Karatsuba's optimization) is $O(n^{\log_2 3})$. Since this dominates over $O(n)$, it means that we don't care about the addition step. Therefore, each layer has a cost of $O(n^{\log_2 3})$. Putting this all together, this means that our recurrence relation is:

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n^{\log_2 3})$$

Using the master theorem, we identify that $a = 3, b = 2$ and $d = \log_2 3$. This means that in our case, $b^d = 2^{\log_2 3} = 3 = a$, so therefore invoking the master theorem tells us that our runtime is $T(n) \in O(n^{\log_2 3} \log n)$, exactly as desired.

(c) Code your solution! Go to https://github.com/Berkeley-CS170/cs170-fa23-coding and follow the directions in the README.md to complete this week's coding homework in hw01.

Notes:

- *Submission Instructions:* Please download your completed `decimal_to_binary.ipynb` file and submit it to the gradescope assignment titled "Homework 1 Coding Portion".
- *OH/HWP Instructions:* While we will be providing conceptual help on the coding portion of homeworks, OH staff will not look at your code and/or help you debug.
- *Academic Honesty Guideline:* We realize that code for some of the algorithms we ask you to implement may be readily available online, but we strongly encourage you to not directly copy code from these sources. Instead, try to refer to the resources mentioned in the notebook and come up with code yourself. That being said, we **do acknowledge** that there may not be many different ways to code up particular algorithms and that your solution may be similar to other solutions available online.