# 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".

## 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:**

(a) **Midterm 1**: Tuesday 10/3, 7:00pm - 9:00pm PST

(b) **Midterm 2**: Tuesday 11/7, 7:00pm - 9:00pm PST

(c) **Final**: Friday 12/15, 8:00am - 11:00am PST

We do not plan on offering alternate 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:** 10:00pm

(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:** The 2 homework drops are provided in case you have last minute issues and miss the Gradescope deadline. Homework extensions are not granted because solutions need to be released soon after the deadline, and so you do nothing.

However, we do give exceptions for DSP-related reasons and/or severe circumstances that affect multiple weeks at a time. If you believe you qualify for those, please contact cs170@.

(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:** The primary source of communication is Edstem.

(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. -Alan Turing

## 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.

(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:** Not OK. As a general rule of thumb, you should never be in possession of any exact homework solutions other than your own.

(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:** OK. Given that you'd inadvertently found a resource online, clearly cite it and make sure you write your answer from scratch.

## 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:** $\log_y x$

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

**Solution:** $\ln xy$

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

**Solution:** $\ln(x/y)$

(iv) $170 \ln x$

**Solution:** $\ln(x^{170})$

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

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

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

(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:**

(a) $x^{\log_{1/x} y} = x^{\frac{\ln y}{\ln(1/x)}} = x^{\frac{\ln y}{-\ln x}} = \left(x^{\log_x y}\right)^{-1} = y^{-1} = \frac{1}{y}$

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

(c) Let $S_n = \sum_{k=0}^{n} ar^k$. When $r \neq 1$, we have

$$\begin{aligned} S_n &= a + ar + \cdots + ar^n \\ rS_n &= ar + ar^2 + \cdots + ar^{n+1} \\ \implies S_n - rS_n &= (a + ar + \cdots + ar^n) - (ar + ar^2 + \cdots + ar^{n+1}) \\ (1-r)S_n &= a - ar^{n+1} \\ S_n &= \frac{a - ar^{n+1}}{1-r} = a\left(\frac{1-r^{n+1}}{1-r}\right) \end{aligned}$$

Otherwise when $r = 1$, we have

$$S_n = a + ar + \cdots + ar^n = \underbrace{a + a + \cdots + a}_{n+1 \text{ times}} = a(n+1)$$

## 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:** Expanding out $T(n)$,

$$\begin{aligned}
T(n) &= 10n + 3 \cdot (10n/4 + 3 \cdot T(n/16)) && (1)\\
&= 10n + 10 \cdot 3n/4 + 10 \cdot 9n/16 + 10 \cdot 27n/64 \cdots && (2)\\
&= \sum_{k=0}^{\lfloor \log_4 n \rfloor} (3/4)^k \cdot 10 \cdot n && (3)\\
&\leq \sum_{k=0}^{\infty} (3/4)^k \cdot 10 \cdot n && (4)\\
&= \frac{1}{1 - 3/4} \cdot 10 \cdot n && (5)\\
&= \Theta(1) \cdot 10 \cdot n && (6)\\
&= \Theta(n) && (7)
\end{aligned}$$

Notice that we drop a constant fraction (1/4 in this case) of the input at each level. I.e. there are 3 recursive calls on sub-problems of size $n/4$ each, so the total input size on each subsequent level is $3n/4$ and $n/4$ of the input is dropped.

Hence, $T(n)$ will turn into a geometric series. We only need the fact that the summation in (3) converges, and not its actual value, since it is a constant. So, the amount of work will be the same as the work in the first level $\Theta(n)$.

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

**Solution:** Answer is $\Theta(n)$.

No need to expand out the recurrence, as we use the observations from (a). The number of recursive calls is less that the number of subproblems so the problem size drops by a constant fraction (3/100) each time. In such cases the runtime is dominated by the work at the first level, $\Theta(n)$.

(The master theorem is not needed, but it is useful to be able to recognize this special case of dropping constant fraction of input size at each level).

(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:**

$$T(n) = 3T(n/5) + \Theta(n^2)$$

Same idea as (a) and (b), we drop a constant fraction of the input size/amount of work done in the first level, so the answer is $\Theta(n^2)$.

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

**Solution:** Draw a recurrence tree and you'll see that each level sums to $\Theta(n)$ and we can have at most $\log n$ levels, so the overall runtime is $\Theta(n \log n)$.

(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:** The Master theorem does not directly apply to this problem. We will try a different approach where we will first attempt to guess a solution. Firstly, note that we have $T(n) \geq 2T(n/2)$. Therefore, we have $T(n) \geq O(n)$. By a similar reasoning, we have $T(n) \leq 2T(4n/5)$. This gives $T(n) \leq O\left(n^{\log_{5/4} 2}\right)$. From these two inferences, we guess that $T(n)$ is probably of the form $an^b$ for some values of $a$ and $b$. To determine the value of $a$, we have $T(1) = 1 = a$. Therefore, we have $T(n) = n^b$. Now, to determine the value of $b$, we use the recurrence relation:

$$T(n) = n^b = T(3n/5) + T(4n/5) = \left(\frac{3}{5}\right)^b n^b + \left(\frac{4}{5}\right)^b n^b$$

The above equation is uniquely satisfied for $b > 0$ at $b = 2$ as the function $f(b) = (3/5)^b + (4/5)^b$ is decreasing in $b$. Therefore, the solution to the above recurrence is $T(n) = n^2$. Therefore, the answer to the $\Theta(n^2)$.

## 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:** $f$ fails (1) for all $k > d$.

(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:** $f$ fails (2) for all $c < \lg a$

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:** Take $D(n)$ to be $\lg n$. Then for any constant $k$,

$$f(n) = n^{\lg n} = \Omega(n^k)$$

Also for any constant c,

$$n^{\lg n} = 2^{(\lg n)^2} = O(2^{cn})$$

Any other $D(n) = o(n/\lg n)$ also works because $n^{D(n)} = 2^{\lg n \cdot D(n)} = 2^{o(n)}$.

# 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!*

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

(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.

**Solution:**

(a) Similar to Karatsuba's algorithm, we can write $x$ as $10^{n/2} \cdot a + b$ for two $n/2$-digit numbers $a, b$. The algorithm is to recursively compute the binary representations of $10^{n/2}, a$, and $b$. We can then compute the binary representation of $x$ using one multiplication and one addition.

(b) At each recursive step, we *divide* (and recurse) on 3 half-size subproblems. Then when *conquering*, the multiplication takes $O(n^{\log_2(3)})$ time and the addition takes $O(n)$ time. So the recurrence we get is

$$T(n) = 3T(n/2) + [O(n^{\log_2(3)}) + O(n)] = 3T(n/2) + O(n^{\log_2(3)}).$$

Either by using Master Theorem or drawing out a tree, we can show that this has solution $O(n^{\log_2(3)} \log n)$.

(There is an $O(n^{\log_2(3)})$-time algorithm: We can compute the binary representation of $10^n$ in time $O(n^{\log_2(3)})$ by doing the multiplications $10 * 10, 10^2 * 10^2, 10^4 * 10^4 \ldots$ - note that the multiplication of $10^{n/2} * 10^{n/2}$ dominates the total runtime of these multiplications. This gives the recurrence $T(n) = 2T(n/2) + O(n^{\log_2(3)})$ instead, with solution $T(n) = O(n^{\log_2(3)})$.)

(c) We provide the coding solution below:

```
def decimal_to_binary(decimal):
    """
    args:
        decimal:string = decimal representation of a number, passed
            as a string
    returns:
        A string representing the binary representation of the number
    """
    # BEGIN SOLUTION
    # base case
    if not decimal:
        return '0'
    if len(decimal) == 1 or decimal == '10':
        return digit_to_binary(decimal)

    # split the decimal into halves and recursively
    # generate the binary representation
    lhs = decimal[:(len(decimal)+1)//2]
    rhs = decimal[(len(decimal)+1)//2:]
    power_of_10 = '1' + '0'*len(rhs)

    lhs_bin = decimal_to_binary(lhs)
    rhs_bin = decimal_to_binary(rhs)
    power_of_10_bin = decimal_to_binary(power_of_10)

    # reconstruct the original number by multiplying and adding
    return add_binary(mul_binary(lhs_bin, power_of_10_bin), rhs_bin)
    # END SOLUTION
```