CS 170 Homework 1

Due **1/26/2021, 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, write "none".

2 Course Policies

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

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

Solution:

(a) **Midterm 1**: Thursday 2/18, during lecture (9:00am - 11:00am PST)
(b) **Midterm 2**: Tuesday 3/16, during lecture (9:00am - 11:00am PST)
(c) **Final**: Wednesday 5/12, 11:30am - 2:30pm PST

We do not plan on offering alternate exams.

(b) Homework is due Tuesdays 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.

(d) What is the primary source of communication for CS170 to reach students? We will email 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 Piazza.

(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) **Piazza Etiquette:** `https://cs170.org/resources/piazza-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 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 Asymptotic Complexity Comparisons

(a) Order the following functions so that for all i, j, if f_i comes before f_j in the order then $f_i = O(f_j)$. Do not justify your answers.

- $f_1(n) = 3^n$
- $f_2(n) = n^{\frac{1}{3}}$
- $f_3(n) = 12$
- $f_4(n) = 2^{\log_2 n}$

- $f_5(n) = \sqrt{n}$
- $f_6(n) = 2^n$
- $f_7(n) = \log_2 n$
- $f_8(n) = 2^{\sqrt{n}}$
- $f_9(n) = n^3$

As an answer you may just write the functions as a list, e.g. $f_8, f_9, f_1, \ldots$

Solution: $f_3, f_7, f_2, f_5, f_4, f_9, f_8, f_6, f_1$

(b) In each of the following, indicate whether $f = O(g)$, $f = \Omega(g)$, or both (in which case $f = \Theta(g)$). ***Briefly*** justify each of your answers. Recall that in terms of asymptotic growth rate, logarithmic < polynomial < exponential.

	$f(n)$	$g(n)$
(i)	$\log_3 n$	$\log_4(n)$
(ii)	$n\log(n^4)$	$n^2\log(n^3)$
(iii)	$\sqrt{n}$	$(\log n)^3$
(iv)	$n + \log n$	$n + (\log n)^2$

Solution:

(i) $f = \Theta(g)$; using the log change of base formula, $\frac{\log n}{\log 3}$ and $\frac{\log n}{\log 4}$ differ only by a constant factor.

(ii) $f = O(g)$; $f(n) = 4n\log(n)$ and $g(n) = 3n^2\log(n)$, and the polynomial in g has the higher degree.

(iii) $f = \Omega((\log n)^3)$; any polynomial dominates a product of logs.

(iv) $f = \Theta(g)$; Both f and g grow as $\Theta(n)$ because the linear term dominates the other.

5 Computing Factorials

Consider the problem of computing $N! = 1 \times 2 \times \cdots \times N$.

(a) N is $\log N$ bits long (this is how many bits are needed to store a number the size of N). Find an $f(N)$ so that $N!$ is $\Theta(f(N)))$ bits long. Simplify your answer as much as possible, and give an argument for why it is true.

Note: You may not use Stirling's formula.

Solution: When we multiply an m bit number by an n bit number, we get an $(m+n)$ bit number. When computing factorials, we multiply N numbers that are at most $\log N$ bits long, so the final number has at most $N\log N$ bits.

But if you consider the numbers from $\frac{N}{2}$ to N, we multiply at least $\frac{N}{2}$ numbers that are at least $\log N - 1$ bits long, so the resulting number has at least $\frac{N(\log N - 1)}{2}$ bits.

This means that our number has $\Theta(N\log N)$ bits.

(b) Give a simple (naive) algorithm to compute $N!$. You may assume that multiplying two bits together (e.g. $0 \times 0, 0 \times 1$) takes 1 unit of time.

Please give both the algorithm description and runtime analysis.

Solution: We can compute $N!$ naively as follows:

```
factorial ($N$)
    $f = 1$
    for $i = 2$ to $N$
        $f = f \cdot i$
```

Running time : we have N iterations, each one multiplying an $N \cdot \log N$-bit number (at most) by an $\log N$-bit number. Using the naive multiplication algorithm, each multiplication takes time $O(N \cdot \log^2 N)$. Hence, the running time is $O(N^2 \log^2 N)$.

6 Polynomial Evaluation

Given coefficients $a_0, \ldots, a_n \in \mathbb{N}$, consider the polynomial:

$$p(x) = \sum_{k=0}^{n} a_k x^k$$

For this problem, assume that addition and multiplication of two natural numbers takes only $\mathcal{O}(1)$ time (regardless of how large they are).

(a) Describe a naive algorithm that, given $[a_0, \ldots, a_n]$ and x, computes $p(x)$. Give an analysis of its runtime as a function of the degree of the polynomial n.

Solution: The following is pseudocodefor the evaluation algorithm:

```
function eval ($[a_0, \ldots, a_n], x$)
    $result = 0$
    for $i = 0$ to $n$
        $result$ += $a_i * x^i$
    return $result$
```

It takes i time to compute x^i on the ith iteration. Adding up all the time taken gives $\sum_{i=0}^{n} i = \mathcal{O}(n^2)$

(b) As an alternative, we can compute the following expression:

$$p(x) = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + x \cdot a_n) \ldots))$$

Describe and analyze an algorithm that evaluates the polynomial using the above expression. The runtime should be a function of n as well.

Give a 3-part solution as described in the homework guidelines.

Solution:

Algorithm description. We give the description as pseudocode.

```
function eval ($[a_0, \ldots, a_n], x$)
    $result = a_n$
    for $i = 1$ to $n$
        $result$ = $result \cdot x + a_{n-i}$
    return $result$
```

Proof of correctness. We first show the following invariant: 'at the end of the ith loop iteration, $result = (a_{n-i} + x(a_{n-i+1} + \ldots + x(a_{n-1} + x \cdot a_n) \ldots))$'.

As a base case, at the end of the 1st loop iteration, $result = a_{n-1} + x \cdot a_n$. This follows because $result$ starts at a_n.

Now assume at the end of the $(k-1)$th loop iteration, $result = (a_{n-k+1} + x(a_{n-k+2} + \ldots + x(a_{n-1} + x \cdot a_n) \ldots))$. Then at the end of the kth loop iteration,

$$result' = a_{n-k} + x(result) = (a_{n-k} + x(a_{n-k+1} + \ldots + x(a_{n-1} + x \cdot a_n) \ldots))$$

So when the algorithm terminates after the nth loop iteration, it will have computed the expression $a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + x \cdot a_n) \ldots))$. By repeatedly distributing, we confirm that $\sum_{k=0}^{n} a_k x^k = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + x \cdot a_n) \ldots))$

Runtime analysis. Each loop iteration does an addition and a multiplication, which is a constant amount of work. The loop executes $\mathcal{O}(n)$ iterations, so it takes time $\mathcal{O}(n)$.

Note: It is also possible to write this eval function as a recursive algorithm, but then the induction in the proof of correctness is on the size of the input, not as an invariant.

Note: This technique is called *Horner's method*.