CS 170 HW 1

Due 2019-09-04, at 10:00 pm (grace period until 10:30pm)

1 Instructions:

You are welcome to form small groups (e.g. of 4 people) to work through the homework, but you must write up all solutions by yourself. List your study partners for homework on the first page, or “none” if you had no partners.

Begin each problem on a new page. Clearly label where each problem and subproblem begin. The problems must be submitted in order (all of P1 must be before P2, etc).

No late homeworks will be accepted. No exceptions. This is not out of a desire to be harsh, but rather out of fairness to all students in this large course. The lowest two homework scores will be dropped.

Extra credit questions We might have some extra credit questions in the homework for people who really enjoy the materials. However, please note that you should do the extra credit problems only if you really enjoy working on these problems and want an extra challenge. It is likely not the most efficient manner in which to maximize your score.
2 Study Group
List the names and SIDs of the members in your study group.

3 Course Policies

(a) What dates and times are the exams for CS170 this semester? (Yes, one may still be undetermined when you do this homework. You may note the currently given date, and that’s fine. It will be determined by the start of the second week of classes.)

If you have any conflicts, please fill out the exam conflicts form:

[link]

Filling out the exam conflict form does not guarantee you will be granted accommodation.

Solution: Any answer that was consistent with the course policies at some time was accepted (Sorry for the confusion!!)

But here are the finalized times:

(a) midterm 1: 9/30, 8pm-10pm
(b) midterm 2: 11/7, 8pm-10pm
(c) final: 12/19, 8am-11am

I do not have any conflicts with the course.

(b) 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, who should you contact?

Solution: The 2 homework drops are provided in case we have last minute issues and miss the Gradescope deadline. Homework extensions are not granted because solutions need to be released immediately after the deadline, and so we do not contact anyone.

(c) 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.

Solution: The primary source of communication is Piazza.

(d) Please read all of the following:

(i) Policies: [link]
(ii) Homework FAQ: [link]
(iii) Piazza Etiquette: [link]

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

"I have read and understood the course policies, homework FAQs and Piazza etiquette."
**Solution:** I have read and understood the course policies, homework FAQs and Piazza etiquette. -Alan Turing.
4 Understanding Academic Dishonesty

Before you answer any of the following questions, please read over the course policies (https://cs170.org/policies/) 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.

5 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.

- $f_1(n) = 3^n$
- $f_2(n) = n^{1/3}$
- $f_3(n) = 12$
- $f_4(n) = 2^{\log_2 n}$
- $f_5(n) = \sqrt{n}$
- $f_6(n) = 2^n$
- $f_7(n) = \log_2 n$
- $f_8(n) = 2^{\sqrt{n}}$
- $f_9(n) = n^3$
As an answer you may just write the functions as a list, e.g. \( f_8, f_9, f_1, \ldots \)

**Solution:** \( f_3, f_7, f_2, f_5, f_4, f_9, f_6, f_1 \)

(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 < linear < polynomial < exponential.

<table>
<thead>
<tr>
<th>( f(n) )</th>
<th>( g(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \log_3 n )</td>
<td>( \log_4 (n) )</td>
</tr>
<tr>
<td>( n \log(n^4) )</td>
<td>( n^2 \log(n^2) )</td>
</tr>
<tr>
<td>( \sqrt{n} )</td>
<td>( (\log n)^3 )</td>
</tr>
<tr>
<td>( n + \log n )</td>
<td>( n + (\log n)^2 )</td>
</tr>
</tbody>
</table>

**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.

6 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.

*Hint:* You may use Stirling’s formula:

\[
\sum_{k=1}^{n} \log k \sim \log \left( \sqrt{2\pi n} \left( \frac{n}{e} \right)^n \right)
\]

Where \( f(n) \sim g(n) \) means that \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 1 \).

**Solution:** When we multiply an \( m \) bit number by an \( n \) bit number, we get an \( (m+n) \) bit number. There are two ways to show that \( N! \) is \( \Theta(N \log N) \) bits long:

1. 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 \( \frac{N}{2} \log(\frac{N}{2} - 1) \) bits.
2. By the discussion, \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 1 \) means that \( f(n) = \Theta(g(n)) \). So

\[
\log(N!) = \sum_{k=1}^{N} \log k = \Theta \left( \log \left( \sqrt{2\pi N \left( \frac{N}{e} \right)^N} \right) \right) = \Theta(N \log(N/e) + \log(\sqrt{2\pi \sqrt{N}})) = \Theta(N \log N)
\]

(b) Give a simple (naive) algorithm to compute \( N! \) (you may describe the algorithm or write pseudocode). Give its runtime, and justify the runtime by analyzing it. You may assume that multiplying two bits together (e.g. \( 0 \times 0, 0 \times 1 \)) takes 1 unit of time.

**Solution:** We can compute \( N! \) naively as follows:

```python
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 \( O(N^2 \log^2 N) \).

7 Recurrence Relations

For each part, find the asymptotic order of growth of \( T \); that is, find a function \( g \) such that \( T(n) = \Theta(g(n)) \).

(a) \( T(n) = 4T(n/2) + 42n \)

**Solution:** Use the master theorem. Or:

```
42n
  \_{21n}  \_{21n}  \_{21n}  \_{21n}
```

The first level sums to \( 42n \), the second sums to \( 84n \), etc. The last row dominates, and we have \( \log n \) rows, so we have \( 42 \cdot 2^{\log n} \cdot n = \Theta(n^2) \).

(b) \( T(n) = 4T(n/3) + n^2 \)

**Solution:** Use the master theorem (the case \( d > \log_b a = \log_3 4 \)), or expand like the previous question. The answer is \( \Theta(n^2) \).

(c) \( T(n) = T(\sqrt{n}) + 1 \) (You may assume that \( T(2) = T(1) = 1 \))

**Solution:** The answer to this question is \( \Theta(\log \log n) \). Notice that after the \( k \)th step of the recursion, we have the size of the input to be \( 2^{\log n / 2^k} \). Since, the number of recursive levels is \( \log \log n \) and each level contributes 1, the solution to the problem is \( \Theta(\log \log n) \).
8 Sorted Array

Given a sorted array $A$ of $n$ (possibly negative) distinct integers, you want to find out whether there is an index $i$ for which $A[i] = i$. Devise a divide-and-conquer algorithm that runs in $O(\log n)$ time.

A satisfactory answer will follow the "Responding to Algorithm Guidelines" on the website (https://cs170.org/policies/#responding-to-algorithm-problems). You do not need to give a proof of correctness for this question. Give a clear description of the algorithm (pseudocode is OK, but it shouldn’t be executable code), and justify why the runtime is $O(\log n)$.

Solution: Along the same lines as binary search, start by examining $A[\frac{n}{2}]$. Clearly, if $A[\frac{n}{2}] = \frac{n}{2}$ then we have a satisfactory index; if $A[\frac{n}{2}] > \frac{n}{2}$ then no element in the second half of the array can possibly satisfy the condition because each integer is at least one greater than the previous integer, and hence the difference of $A[\frac{n}{2}] - \frac{n}{2}$ cannot decrease by continuing through the array; and if $A[\frac{n}{2}] < \frac{n}{2}$ then by the same logic no element in the first half of the array can satisfy the condition. While the algorithm has not terminated or left an empty array, we discard the half of the array that cannot hold an answer and repeat the same check. At each step we do a single comparison and discard at least half of the remaining array (or terminate), so this algorithm takes $O(\log n)$ time.