Discrete Structures (CSCI 246)

Homework 5

Purpose & Goals
The following problems provide practice relating to:

- asymptotic analysis (Big O),
- properties of Big O,
- algorithm analysis,
- recurrence relations,
- proof by induction, and
- the problem solving process.

Submission Requirements

- **Type or clearly hand-write your solutions** into a pdf format so that they are legible and professional. Submit your pdf to Gradescope. **Illegible, non-pdf, or emailed solutions will not be graded.**
- Each problem should start on a new page of the document. When you submit to Gradescope, associate each page of your submission with the correct problem number. Please post in Discord if you are having any trouble using Gradescope.
- Try to model your formatting off of the proofs from lecture and/or the textbook.
- Submit to Gradescope early and often so that last-minute technical problems don't cause you any issues. Only the latest submission is kept. Per the syllabus, assignments submitted within 24 hours of the due date will receive a 25% penalty and assignments submitted within 48 hours will receive a 50% penalty. After that, no points are possible.

Academic Integrity

- You may work with your peers, but **you must construct your solutions in your own words on your own.**
- Do not search the web for solutions or hints, post the problem set, or otherwise violate the course collaboration policy or the MSU student code of conduct.
- Violations (regardless of intent) will be reported to the Dean of Students, per the MSU student code of conduct, and you will receive a 0 on the assignment.

Tips

- Answer each problem to the best of your ability. Partial credit is your friend!
- Read the hints for where to find relevant examples for each problem.
- Refer to the problem solving and homework tips guide.

- It is not a badge of honor to say that you spent 5 hours on a single problem or 15 hours on a single assignment. Use your time wisely and get help (see “How to Get Help” below).

How to Get Help

When you are stuck and need a little or big push, **please ask for help!**

- Timebox your effort for each problem so that you don’t spend your life on the problem sets. (See the problem solving tips guide for how to do this effectively.)
- Post in Discord. If you’re following the timebox guide, you can post the exact statement that you produced after spending 20 minutes being stuck.
- Come to office hours or visit the CS Student Success Center.

Problem 1 (12 points)

Hint: See the lecture introducing asymptotic analysis for examples of proving and disproving that $f(n) = O(g(n))$.

(a) (3 points) Prove that $4n^3+2n^2-n = O(n^3)$ by constructing $c > 0, n_0 \geq 0 : \forall n \geq n_0 : 4n^3+2n^2-n \leq c \cdot n^3$.

Grading Notes. This rubric is straightforward: give a correct c and n_0.

(b) (9 points) Prove that $4n^3+2n^2-n \neq O(n^2)$ by disproving $\exists c > 0, n_0 \geq 0 : \forall n \geq n_0 : 4n^3+2n^2-n \leq c \cdot n^2$.

Grading Notes. While a detailed rubric cannot be provided in advance as it gives away the solution details, the following is a general idea of how points are distributed for this problem. We give partial credit where we can.

(7) **Correctness.** If your proof is not correct, this is where you'll get docked. Make sure you model your proof off of examples of proofs that function f is not big O of function g from class or the textbook.

(2) **Communication.** We need to see a mix of notation and intuition. If you skip too many steps at once, or we cannot follow your proof, or if your proof is overly wordy or confusing, this is where you'll get docked.

Problem 2 (15 points)

Hint: See the lecture introducing asymptotic analysis for examples of proving and disproving that $f(n) = O(g(n))$.

(a) (2 points) Give two functions, f and g, such that $f = O(n^2)$ and $g = O(2^n)$ but $f \neq O(g)$.

Grading Notes. This rubric is straightforward: give a correct f and g.

(b) (6 points) Prove that $f = O(n^2)$ and $g = O(2^n)$.

(c) (6 points) Prove that $f \neq O(g(n))$.

Grading Notes. For each of (b) and (c) above, here is the grading rubric:

(5) **Correctness.** If your proof is not correct, this is where you'll get docked. Recall that a proof that f is big O of g involves specifying a $c > 0, n_0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)$, and a proof that f is not big O of g involves demonstrating that no such c, n_0 can exist.

(1) **Communication.** We need to see a mix of notation and intuition. If you skip too many steps at once, or we cannot follow your proof, or if your proof is overly wordy or confusing, this is where you'll get docked.

Problem 3 (12 points)

For the following two algorithms, derive the "best" (i.e., tightest) big O running times.

Hint. See the lecture on introduction to algorithm analysis for examples of analyzing the runtime of algorithms.

Grading Notes. For each of (a) and (b), we need to see:

- (3 points) A proposed function representing the number of primitive operations for the algorithm in terms of the input size, addressing each line and/or loop of the algorithm. You do not need to be precise counting constant numbers of primitive operations (e.g., figuring out exactly how many primitive operations a line does). However, you should be precise about how many times a loop runs.
- (2 points) For your proposed function representing the number of primitive operations $f(n)$, the "best" (i.e., tightest) $g(n)$ such that $f(n) = O(g(n))$. If $f(n) \neq g(n)$, explain why $f(n) = O(g(n))$ using the lemmas from Chapter 6 of the textbook, which we covered in the lecture about properties of Big O.
- (1 point) Clear communication: a mix of mathematical notation and explanations in words.

(a)

Algorithm 1

```
for $i = 1$ to $n \cdot n$ do
    if $i$ is even then
        for $j = 1$ to $n$ do
            $x = x + 1$
```

(b)

Algorithm 2

```
for $i = 1$ to $n \cdot n$ do
    if $n|i$ then
        for $j = 1$ to $n$ do
            $x = x + 1$
```

Problem 4 (18 points)

Hint: See the lecture on analysis of recursive algorithms for examples of analyzing recursive algorithms. Since we didn't do the proof of the closed-form solution for factorial in class, look at 6.23 for an example.

Consider the following computer science problem, called the *parity problem*, defined by its inputs and outputs:

- Input: nonnegative integer n
- Output: the parity (i.e., evenness or oddness) of n, represented by a 0 if n is even and 1 if n is odd.

(a) (12 points) Here is a proposed solution for the parity problem.

Algorithm 3 `parity1`(n)

```
if $n = 0$ or $n = 1$ then
    return $n$
else
    return parity1(n-1)
```

(i) (1 points) Is the algorithm correct? If no, give an example input on which it does not return the correct output. If yes, give a short (one or two sentence) intuitive explanation as to why.

(ii) (2 points) Draw the recursion tree for `parity1`, indicating the size of the input at each recursive call (node) and the runtime at each recursive call. (Use a variable to represent constants.)

(iii) (3 points) Give the recurrence relation expressing the runtime of `parity1`(n), denoted by $T(n)$. You need (1 point) the base case(s) and (2 points) the recursive case.

(iv) (2 points) For $n = 2$, $n = 3$, and $n = 4$, compute the value of $T(n)$ from the recurrence relation.

(v) (1 point) Make a guess about the closed-form solution of $T(n)$ for $n \geq 1$. (The runtime of $n = 0$ doesn't quite fit, so we ignore it here.)

(vi) (3 points) Use mathematical induction to prove that your guess is correct for all $n \geq 1$.

(b) (6 points) Here is a second solution for the parity problem.

Algorithm 4 `parity2`(n)

```
if $n = 0$ or $n = 1$ then
    return $n$
else
    return parity2(n-2)
```

(i) (1 points) Is the algorithm correct? If no, give an example input on which it does not return the correct output. If yes, give a short (one or two sentence) intuitive explanation as to why.

(ii) (2 points) Draw the recursion tree for `parity2`, indicating the size of the input at each recursive call (node) and the runtime at each recursive call. (Use a variable to represent constants.)

(iii) (3 points) Give the recurrence relation expressing the runtime of `parity2`(n), denoted by $T(n)$. You need (1 point) the base case(s) and (2 points) the recursive case. You do **not** need to solve for the closed-form solution.