Discrete Structures (CSCI 246)
Homework 12

Purpose & Goals

The following problems provide practice relating to:

- asymptotic analysis (Big O),

- worst-case algorithm analysis,

- algorithm analysis for recursive algorithms, 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 (6 points)

Consider the three sorting algorithms below.

Hint: See the lecture on worst-case runtime analysis.

Algorithm 1 bubbleSort($A[1..n]$)

1: **for** $i = 1$ to n **do**
2:     **for** $j = 1$ to n **do**
3:         **if** $A[j] > A[j + 1]$ **then**
4:             swap $A[j]$ and $A[j + 1]$

Algorithm 2 selectionSort($A[1..n]$)

1: **for** $i = 1$ to n **do**
2:     $minIndex = i$
3:     **for** $j = 1$ to n **do**
4:         **if** $A[j] < A[minIndex]$ **then**
5:             $minIndex = j$
6:     swap $A[i]$ and $A[minIndex]$

Algorithm 3 insertionSort($A[1..n]$)

1: **for** $i = 2$ to n **do**
2:     $j = i$
3:     **while** $j > 1$ and $A[j] < A[j - 1]$ **do**
4:         swap $A[j]$ and $A[j - 1]$
5:         $j = j - 1$

For each, give:

(a) (1 point each) The exact worst-case number of swaps for an input of size n, and the tightest big O upper bound on the worst-case number of swaps for an input of size n

(b) (1 point each) The exact best-case number of swaps for an input of size n, and the tightest big O upper bound on the best-case number of swaps for an input of size n

Problem 2 (18 points)

Hint: See the lecture on analysis of recursive algorithms for examples of analyzing recursive algorithms.

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 4 parity1(n)

1: **if** $n = 0$ or $n = 1$ **then**
2:    return n
3: **else**
4:    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 5 parity2(n)

1: **if** $n = 0$ or $n = 1$ **then**
2:    return n
3: **else**
4:    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.