Solutions to Homework 2

This homework has 8 questions, for a total of 100 points and 5 extra-credit points.
Unless otherwise stated, each question requires *clear*, *logically correct*, and *sufficient* justification to convince the reader.
For bonus/extra-credit questions, we will provide very limited guidance in office hours and on Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must *write your own solution* for all problems and *may not use any other student's write-up*.

(0 pts) 0. **Before you start; before you submit.**

(a) Before starting this assignment, carefully read Handout 3 about "giving an algorithm," and follow it in your solutions.

(b) Correctness arguments and runtime analysis for divide-and-conquer/recursive algorithms should follow the recommended structure described in this week's discussion worksheet.

(c) If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution:

(10 pts) 1. **Self assessment.**

Carefully read and understand the posted solutions to the previous homework. Identify one part for which your own solution has the most room for improvement (e.g., has unsound reasoning, doesn't show what was required, could be significantly clearer or better organized, etc.). Copy or screenshot this solution, then in a few sentences, explain what was deficient and how it could be fixed.

(Alternatively, if you think one of your solutions is significantly *better* than the posted one, copy it here and explain why you think it is better.)

If you didn't do last week's homework, choose a problem from it that looks challenging to you, and in a few sentences, explain the key ideas behind its solution in your own words.

Solution:

2. **(Potential) infinite games.**

Recall that the potential method requires several properties to hold in order to prove that a given process must eventually halt, within a certain number of "time units." These requirements include (but are not limited to):

1. the potential cannot go below some fixed lower bound; or if it does, the process must immediately halt.[1]
2. the potential must *decrease by (at least) some fixed positive amount* with each time unit.

For each of the following processes and candidate potential functions, state with justification whether they satisfy both of the above properties. If not, either describe an alternative potential

[1]In class we required a lower bound of zero, which is without loss of generality, but a fixed lower bound suffices.

function that does satisfy both properties (and allows us to conclude that the process must halt), or briefly explain why the process might not ever halt (and hence there is *no* valid potential function for the process).

(8 pts) (a) Consider a one-player game involving a pile of maize and blue chips. Initially, there is a finite number of each color of chip. The player repeatedly takes a "turn" until no chips remain in the pile, if this ever happens. In each turn, the player must *remove* one chip of its choice from the pile, and if the removed chip is maize, the player may *add* up to *three blue chips* to the pile.

Define a time unit to be a round of the game, and the potential n_i after i rounds to be the total number of chips in the pile. The game ends when there is no remaining chips in the pile, i.e., $n_i = 0$.

Solution:

(8 pts) (b) Eric is playing a chasing game with his naughty duck Waquackquack. Initially, Waquackquack is $d > 0$ miles ahead of Eric. On each "turn", Waquackquack waddles forward by a fixed distance of 1 mile, then Eric runs *half* the remaining distance to Waquackquack. The game ends when Eric catches Waquackquack, if this ever happens.

Define a time unit to be one turn of this chase, and let the potential d_i after i turns to be the remaining distance between Eric and Waquackquack. The chase ends when Eric catches Waquackquack, i.e., when $d_i = 0$.

Solution:

3. **Potential functions of functions.**

For each of the following functions, determine whether the function *must* eventually halt, regardless of the input.

If your answer is 'yes', prove it by defining and analyzing a potential function in terms of the variable(s) in the algorithm. Otherwise, give a specific input that causes the function to loop along with a brief explanation for why it loops on that input.

Reminder: When defining a potential function, you need to also define the "time unit" in a way that your potential function meets the two requirements outlined in Question 2.

(6 pts) (a)

```
Input: positive integers $a$ and $b$
function FUNC($a$, $b$)
    if $a = 1$ or $b = 1$ or $a = b$ then
        return 0
    if $a > b$ then
        return FUNC($a/2, 2b$)
    else
        return FUNC($2a, b/2$)
```

Solution:

(8 pts) (b)

```
Input: positive integers $a$ and $b$
function ALG($a$,$b$)
    if $a \le b$ then
        return $b$
    if $b$ is even then
        return ALG($a-1, b+1$)
    else
        return ALG($a, b+1$)
```

Solution:

(8 pts) (c)

```
Input: a positive integer $x$
function FOO($x$)
    while $x > 10$ do
        if $x$ is odd then
            $x \leftarrow x+3$
        else
            $x \leftarrow x/2$
```

Solution:

4. **Can you *master* the sorting algorithms?**

In this problem, we will analyze the runtime recurrence of two sorting algorithms (which are both correct!), namely, SLOWSORT and STOOGESORT. For each of the following algorithms,

1. give (with brief justification[2]) a recurrence for the algorithm's running time $T(n)$ as a function of the input array size n, and
2. state whether the Master Theorem is applicable to the recurrence, and if so, use it to give the closed-form solution; if not, explain why not.

(6 pts) (a)

```
1: function SLOWSORT($A[1,\dots,n]$)
2:     SLOWSORT($A[1,\dots,\lfloor\frac{n}{2}\rfloor]$)
3:     SLOWSORT($A[\lfloor\frac{n}{2}\rfloor+1,\dots,n]$)
4:     if $A[\lfloor\frac{n}{2}\rfloor] > A[n]$ then
5:         swap $A[\lfloor\frac{n}{2}\rfloor]$ and $A[n]$
6:     SLOWSORT($A[1,\dots,n-1]$)
7:     return $A$
```

Solution:

[2]You may refer to specific lines in the algorithm.

(6 pts) (b)

```
function StoogeSort($A[1, \ldots, n]$)
    if $n = 1$ then
        return $A$
    if $n = 2$ and $A[1] > A[n]$ then
        swap $A[1]$ and $A[n]$
    if $n > 2$ then
        $t \leftarrow \lceil 2n/3 \rceil$
        StoogeSort($A[1, \ldots, t]$)
        StoogeSort($A[n - t + 1, \ldots, n]$)
        StoogeSort($A[1, \ldots, t]$)
    return $A$
```

Solution:

5. **Karatsuba for polynomials.**

If you haven't already, carefully read Handout 3 about "giving an algorithm," and follow it for the next two problems.

In lecture, we have seen how the Karatsuba algorithm reduces the number of multiplications needed for mutiplying large integers. In this problem, we will see how a similar idea can be used to multiply polynomials efficiently.

Let $A(x) = a_0 + a_1 x + \ldots a_{n-1}x^{n-1}$ and $B(x) = b_0 + b_1 x + \ldots + b_{n-1}x^{n-1}$ be two polynomials of degree[3] at most $n - 1$. For simplicity, assume that n is a power of 2, and that arithmetic operations (e.g., addition, multiplication) between two coefficients can be done in constant time.

As an example, the product of two polynomials of degree 1, $A(x) = 1 + 2x$ and $B(x) = 3 + 4x$ is

$$A(x) \cdot B(x) = (1 + 2x)(3 + 4x) = 3 + 10x + 8x^2.$$

We can use a divide-and-conquer approach to compute the coefficients of the product polynomial $A(x) \cdot B(x)$ in $O(n^{\log_2 3})$ time. To do this, we first split the polynomials into their higher-degree and lower-degree terms

$$A(x) = p(x) \cdot x^{n/2} + q(x)$$
$$B(x) = r(x) \cdot x^{n/2} + s(x)$$

where $p(x)$ represents the higher-degree $n/2$ terms of $A(x)$ (with the $x^{n/2}$ factored out) and $q(x)$ represents the lower-degree $n/2$ terms; and similarly with $r(x)$ and $s(x)$ for $B(x)$.

(4 pts) (a) Show that

$$A(x) \cdot B(x) = (p(x) \cdot r(x)) \cdot x^n + (p(x) \cdot s(x) + q(x) \cdot r(x)) \cdot x^{n/2} + (q(x) \cdot s(x)).$$

[3]The degree of a polynomial is the highest power of the variable (like x) in the polynomial when it is written in its standard form.

Solution:

(4 pts) (b) Show that the middle term in Part a, $p(x) \cdot s(x) + q(x) \cdot r(x)$, can be computed using only one multiplication of polynomials.

Solution:

(10 pts) (c) Give an $O(n^{\log_2 3})$-time divide-and-conquer algorithm to compute the coefficients of the product polynomial $A(x) \cdot B(x)$. Remember to include correctness proof and runtime analysis in your solution.

Solution:

6. **Sorting out errors.**

A coding project requires n different software modules, each with a unique loading priority. Modules must be loaded in an order that respects their priorities—modules with lower priority values (indicating higher precedence) should be loaded before those with higher priority values. If a module with a higher priority value (lower precedence) is loaded before one with a lower priority value, an error message is output for each such conflict.

For example, given the following load order (where the modules are loaded from left to right and are represented and referred to by their priority values):

$$[4, 2, 3, 1]$$

There are 5 error messages output:

- Module 1 causes 3 error messages, as Modules 4, 2, and 3 are all loaded before it.
- Module 2 causes 1 error message, as Module 4 is loaded before it.
- Module 3 causes 1 error message, as Module 4 is loaded before it.

In this problem we are concerned with algorithm that determine the number of error messages that will be output. The input is an array $A[1, \ldots, n]$ of the modules' load priorities, from the beginning of the load order to the end. (So, $A[1]$ is the priority of the first module being loaded, and $A[n]$ is the priority of the last module.) The desired output is the total number of error messages. (Throughout this question, assume that any two priorities can be looked up and compared in constant time, independent of n.)

(4 pts) (a) Describe briefly, clearly, and precisely (in English) a simple brute-force algorithm for this problem; do not give pseudocode. State, with brief justification, a $\Theta(\cdot)$ bound on its (worst-case) running time as a function of n. You do not need to give a formal proof.

Solution:

(10 pts) (b) Suppose that both the front half and back half of the load order (i.e., the two halves of the array A) happen to be sorted in proper order of priorities, though the entire line may not be.

Give an $O(n)$-time algorithm that outputs the number of error messages in this scenario. Your description of the algorithm can be in English or in pseudocode (as you prefer), but it should be clear and precise.

Hint: This is the same setup as for the MERGE subroutine (from MERGESORT algorithm), but a different task.

Solution:

(8 pts) (c) Give an $O(n \log n)$-time divide-and-conquer algorithm for the error message counting problem.

Hint: Modify the MERGESORT algorithm slightly, to both sort *and* count. You may call your algorithm in Part b as subroutine.

Solution:

7. **Graphs have potential too, maybe (optional extra credit).**

Reminder: For bonus/extra-credit questions, we will provide very limited guidance in office hours and on Piazza, and we do not guarantee anything about the difficulty of these questions.

Recall that a *connected graph* is a graph in which there is a path between every pair of vertices. The following algorithm determines (or attempts to determine) a certain property of a connected graph.

Input: a connected graph G
Output: [**FOR YOU TO DETERMINE**: truth value indicating some property of G]

```
function ALGORITHM($G$)
    Initialize empty sets $S$, $A$, and $B$
    $v \leftarrow$ vertex in $G$ with the lowest lexicographic order
    $S \leftarrow S \cup \{v\}$ // Add $v$ to set $S$
    $A \leftarrow A \cup \{v\}$ // Add $v$ to set $A$
    while $S$ is not empty do
        $u \leftarrow$ vertex in $S$ with the lowest lexicographic order
        Remove $u$ from $S$
        for each neighbor $t$ of $u$ do
            if $t$ and $u$ are both in $A$ or both in $B$ then
                return False
            if if $t$ is in neither $A$ nor $B$ then
                $S \leftarrow S \cup \{t\}$ // Add $t$ to set $S$
                Add $t$ to a different set from $u$ ($A$ or $B$)
    return True
```

(3 EC pts) (a) Determine whether ALGORITHM *must* eventually terminate, regardless of the input. If your answer is 'yes', prove it by defining and analyzing a potential function. Otherwise, give a specific graph that causes the function to loop along with a brief explanation for why it loops on that input.

Solution:

(2 EC pts) (b) Determine the property that the above algorithm determines (or attempts to determine, based on your conclusion in Part a) about its input graph G. You do not have to justify your solution.

Solution: