CS 161 (Stanford, Winter 2024) Homework 2

Style guide and expectations: Please see the "Homework" part of the "Resources" section on the webpage for guidance on what we are looking for in homework solutions. We will grade according to these standards. You should cite all sources you used outside of the course material.
What we expect: Make sure to look at the "**We are expecting**" blocks below each problem to see what we will be grading for in each problem!

Exercises. The following questions are exercises. We suggest you do these on your own. As with any homework question, though, you may ask the course staff for help.

1 Exercise: Solving Recurrence Relations

1.1 (4 pt.)

State and prove a tight bound for the following recurrence relation:

$$T(n) = 5T(n/3) + n^2$$

[We are expecting: A brief but formal justification, likely citing a well-known theorem discussed in class.**]**

1.2 (4 pt.)

Use a recursion tree to give an upper bound on the following recurrence relation:

$$T(n) = 3T(n/5) + n$$

[We are expecting: A drawing of a recursion tree including a justification for the weight (i.e. amount of work involved) of the entire tree based on the weight at each level and the number of levels. You are welcome to upload an image of your recursion tree (for LaTeX, use includegraphics)**]**.

2 Exercise: Analyze Runtime (6 pt.)

Consider the following procedure, which takes as input an array A:

Algorithm 1: The recursive function printStuff

```
Function printStuff(A):
    n ← length of A
    if n ≤ 4 then
        return
    for i = 0, ..., n − 1 do
        print A[i]
    printStuff (A[:n/4])
    printStuff (A[3n/4:])
    return
```

What is the asymptotic running time of `printStuff`?

[We are expecting: The best answer you can give of the form "The running time of `printStuff` is $O(\dots)$" and a short explanation.**]**

3 Exercise: Induction Error (8 pt.)

Recall the naive divide-and-conquer algorithm for integer multiplication from Lecture 1, which takes two n-digit integers and multiplies them:

Algorithm 2: The recursive function multiply

```
Function multiply(x, y):
    n ← maximum number of digits in x or y
    if n = 1 then
        return x · y      // We know how to do single-digit multiplication.
    Let a, b be n/2-digit numbers such that x = a · 10^(n/2) + b
    Let c, d be n/2-digit numbers such that y = c · 10^(n/2) + d
    ac ← multiply(a, c)
    ad ← multiply(a, d)
    bc ← multiply(b, c)
    bd ← multiply(b, d)
    return ac · 10^n + (ad + bc) · 10^(n/2) + bd
```

Find out what is the error in the proof of the following "claim":

"Claim": The algorithm runs in time $T(n) = O(n \log(n))$.

"Proof": At the top level, we add two numbers of $n/2$ digits each, which takes $O(n)$ time. At each subsequent level, we increase the number of sub-problems by a constant factor (4), and the sub-problems only become smaller. Therefore, the running time per level can only increase by a constant factor. By induction, since for the first level, it is $O(n)$, it will be $O(n)$ for all levels. There are $O(\log(n))$ levels, so in total the running time is $T(n) = O(n \log(n))$.

[We are expecting: A clear and concise description, in plain English, of the error with this proof.**]**

4 Exercise: Count Occurrences

Given a sorted array of integers containing duplicates, design an algorithm to count the number of occurrences of a given element (or return -1 otherwise). For example, if you are given the following inputs:

$$\begin{aligned} \text{arr} &= [2, 2, 5, 5, 5, 7, 13, 14] \\ \text{target} &= 5 \end{aligned}$$

Your algorithm would return the following:

Element 5 occurs 3 times

4.1 (1 pt.)

Give a straightforward algorithm that runs in $O(n)$ time.

[We are expecting: A brief but clear English description of the algorithm.**]**

4.2 (8 pt.)

Give an improved algorithm that runs in $O(\log n)$ time, using a divide-and-conquer strategy.

[We are expecting: Pseudocode and a brief English description, as well as an informal justification of the correctness and of the running time.**]**

Problems. The following questions are problems. You may talk with your fellow CS 161-ers about the problems. However:

- Try the problems on your own *before* collaborating.
- Write up your answers yourself, in your own words. You should never share your typed-up solutions with your collaborators.
- If you collaborated, list the names of the students you collaborated with at the beginning of each problem.

5 Dog Safety

You decide to open a brand-new dog-walking business that specializes in walking large dogs. Because large dogs can be difficult to walk, you want to screen applicants based on whether they are already experienced in handling large dogs.

You have n applicants, and you are tasked with finding experienced dog walkers among them and figuring out which ones are relatively new to walking large dogs.

You decide to adopt the following method. Two walkers will walk with each other and mutually evaluate each other's dog walking skills. We will refer to such walks as "evaluations". For the purposes of this problem, an experienced walker will immediately recognize another experienced walker, and will always tell you honestly that the other is experienced. However, new walkers are not good judges of dog handling quality and their conclusion can be wrong. They might say that the other person is or isn't an experienced walker, regardless of the skills observed.

For example, if Alice the walker and Bob the walker are both experienced, then they will both say that the other is experienced. But if Alice the walker is experienced and Bob the walker is new, then Alice will rate Bob as new, but Bob may say either that Alice is experienced or that she is new. The rating outcomes of an evaluation with walkers A and B are as follows:

Walker A	Walker B	A says (about B)	B says (about A)
Experienced	Experienced	Experienced	Experienced
Experienced	New	New	Either
New	Experienced	Either	New
New	New	Either	Either

Suppose that among the n applicants, strictly more than $n/2$ are experienced.

In this problem, you will develop an algorithm to find all of the experienced walkers, while using only $O(n)$ evaluations. We will build up to this algorithm over multiple sub-parts, so it's okay if it isn't immediately obvious how the initial subparts tie into the overall solution! Along the way, you will also practice some of the skills that we've seen in Week 1.

5.1 (4 pt.)

Give an algorithm that uses $O(n^2)$ evaluations and identifies all of the experienced walkers.

[We are expecting: A description of the procedure (either in pseudocode or very clear English), with a brief explanation of what it is doing and why it works.**]**

5.2 (12 pt.)

Now let's start designing an improved algorithm. The following procedure will be a building block in our algorithm—make sure you read the requirements carefully!

Suppose that n is even. Show that, using only $n/2$ evaluations, you can reduce the input to our problem to a valid input of the same problem with less than half the size. That is, give a procedure that does the following:

- **Input:** A group of n applicants, where n is even, so that there are strictly more than $n/2$ experienced walkers in the group.
- **Output:** A group of m applicants, for some $0 < m \leq n/2$, so that there are strictly more than $m/2$ experienced walkers in the group.
- **Constraint:** The number of evaluations is no more than $n/2$.

[We are expecting: A description of this procedure (either in pseudocode or very clear English), and rigorous argument that it satisfies the **Input**, **Output**, and **Constraint** requirements above.**]**

5.3 (0 pt.)

Bonus: Extend your argument for odd n. That is, give a procedure that does the following:

- **Input:** A group of n applicants, where n is odd, so that there are strictly more than $n/2$ experienced walkers in the group.
- **Output:** A group of m applicants, for $0 < m \leq \lceil n/2 \rceil$, so that there are strictly more than $m/2$ experienced walkers in the group.
- **Constraint:** The number of evaluations is no more than $\lfloor n/2 \rfloor$.

($\star$) *For all of the following parts, you may assume that the procedures in parts 5.2 and 5.3 exist even if you have not done those parts.*

5.4 (4 pt.)

Using the procedures from parts 5.2 and 5.3, design a recursive algorithm that uses $O(n)$ evaluations and finds a *single* experienced walker.

[We are expecting: A description of the procedure (either in pseudocode or very clear English).**]**

5.5 (8 pt.)

Prove formally, using induction, that your answer to part 5.4 is correct.

[We are expecting: A formal argument by induction. Make sure you explicitly state the inductive hypothesis, base case, inductive step, and conclusion.**]**

5.6 (8 pt.)

Prove that the procedure you provided in part 5.4 uses $O(n)$ evaluations. Do this argument "from scratch", do not use the Master Theorem. If you find that you are working with floors and ceilings, you may ignore them (i.e., assume that the quantity is a whole number).

[We are expecting: A formal argument.**]**

5.7 (4 pt.)

Give a procedure to find *all* experienced walkers using $O(n)$ evaluations.

[We are expecting: An informal description of the procedure. **]**

5.8 (6 pt.)

When we work with algorithms, we often need to ignore or change aspects of a real-world situation in order to turn it into an algorithmically solvable problem. For example, we can write an algorithm that sorts a numbered list without knowing what "the numbers" are numbers of (rides? burritos?).

- **Abstraction** is when we omit details of the real-world situation. For example, we could omit the kind of thing being sorted by our algorithm, what condition it is in, what color it is, or how long it has been in the list.
- **Idealization** is when we deliberately change aspects of the real-world situation. For example, we could round the numbers being sorted to make them whole numbers.

Compare the problem you just solved to the real-world problem of evaluating the experience of dog walkers or worker ratings in the app-based gig economy. Identify one abstraction and one idealization.

[We are expecting: 4-6 sentences which identify one aspect of the problem setup above as an abstraction, one identified as an idealization of the real-world problem. For each, explain why you believe it is an idealization or abstraction**]**

5.9 (6 pt.)

In many gig economy platforms, a two-way rating system (similar to but more complicated than what we had here), is used to evaluate gig economy workers and users. However, unlike in our problem, the ratings customers give workers and the ratings workers give customers are not equally impactful. Read one of the following articles (ordered by approximate length).

- To fix racial bias in the gig economy, start with the rating systems
- How Gig Workers Strive To Exert Some Control Over Their Work

Identify one aspect of real-world gig platform ratings that is unfair, biased, or otherwise in need of improvement and explain the problem you see with it. Can you trace the problem back to an abstraction or idealization in the rating system?

[We are expecting: 2-3 sentences describing the problem you identified, and 2-3 sentences explaining whether and how it may be explained by abstraction or idealization in the rating system.**]**