CS 161	**Problem Set 2**
Winter 2021	**Due:** Wed, Jan 27 at 11:59 pm PST

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

We suggest you do these on your own. As with any homework problem, though, you may ask the course staff for help.

---

1. **(4 pt.)** There exist different ways to solve the recurrence relation $T(n) = 2 \cdot T(n/2) + n$ with $T(1) = 1$. From lectures, we have seen that $T(n)$ is exactly $n(1 + \log(n))$ when n is a power of 2. In this exercise, you'll analyze a few more recurrences for a few variants.

   (a) What is the exact solution to $T(n) = 3 \cdot T\left(\frac{n}{3}\right) + n$ with $T(1) = 3$, when n is a power of 3?

   (b) What is the exact solution to $T(n) = 3 \cdot T\left(\frac{n}{3}\right) + 3n$ with $T(1) = 1$, when n is a power of 3?

   [**We are expecting:** Your answer — **no justification required.** Notice that we want the exact answer, so don't give an $O()$ statement.]

2. (4 pt.) Use any of the methods we've seen in class so far to give big-Oh solutions to the following recurrence relations. You may treat fractions like $n/2$ as either $\lfloor n/2 \rfloor$ or $\lceil n/2 \rceil$, whichever you prefer.

(a) $T(n) = 3T\left(\frac{n}{9}\right) + \sqrt{n}$ for $n \geq 9$, and $T(n) = 1$ for $n < 9$.

(b) $T(n) = T(n-4) + n$ for $n \geq 4$, and $T(n) = 1$ for $n < 4$. (You may assume $n \bmod 4 = 0$.)

(c) $T(n) = 6T\left(\frac{n}{4}\right) + n^2$ for $n \geq 4$, and $T(n) = 1$ for $n < 4$.

(d) $T(n) = 5T\left(\frac{n}{2}\right) + n^2$ for $n \geq 2$, and $T(n) = 1$ for $n < 2$.

[**We are expecting:** For each item, the best answer you can give of the form $T(n) = O(\_\_\_\_)$ and a justification. (That is, all of these satisfy $T(n) = O(2^n)$, but you can do better). You do not need to give a formal proof, but your justification should be convincing to the grader. You may use the Master Theorem if it applies.]

3. (2 pt.) Consider the following algorithm, which takes as input an array A:

```
def printStuff(A):
    n = len(A)
    if n <= 4:
        return
    for i in range(n):
        print(A[i])
    printStuff(A[:n/3])  # recurse on first n/3 elements of A
    printStuff(A[2*n/3:])  # recurse on last n/3 elements of A
    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(____)" and a short explanation.]

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.

---

4. Matrix multiplication. Suppose that we have $n \times n$ matrices X and Y and we'd like to multiply them.

(a) **(2 pt.)** What is the running time of the standard algorithm (that computes the inner product of rows of X and columns of Y)? You can assume that simple arithmetic operations, like multiplication, take constant time ($O(1)$).

[**We are expecting:** A detailed analysis of the runtime of your algorithm including the Big-O time in terms of n.]

(b) **(3 pt.)** Now let's divide up the problem into smaller chunks like this, where the eight $\frac{n}{2} \times \frac{n}{2}$ sub-matrices (A, B, C, D, E, F, G, H) are each quarters of the original matrices, X and Y:

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

We now have a divide and conquer strategy! Find the recurrence relation of this strategy and the runtime of this algorithm.

[**We are expecting:** A detailed analysis of the runtime of your algorithm including the Big-O time in terms of n.]

(c) **(3 pt.)** Can we do better? It turns out we can by calculating only 7 of the subproblems:

$$P_1 = A(F - H) \qquad P_5 = (A + D)(E + H)$$
$$P_2 = (A + B)H \qquad P_6 = (B - D)(G + H)$$
$$P_3 = (C + D)E \qquad P_7 = (A - C)(E + F)$$
$$P_4 = D(G - E)$$

And we can solve XY by

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

We now have a more efficient divide and conquer strategy! What is the recurrence relation of this strategy and what is the runtime of this algorithm?

[**We are expecting:** A detailed analysis of the runtime of your algorithm including the Big-O time in terms of n.]

Historical note: Sophisticated variants of this strategy have resulted in (asymptotically) better-and-better algorithms for matrix multiplication over the years. For a humorous take on the most recent improvement, see here (just for fun): https://www.smbc-comics.com/comic/mathematicians.

(d) **(2 pt.)** Your friend tried to solve part (c) of the problem, and came to the following conclusion:

Claim: The algorithm runs in time $T(n) = O(n^2 \log(n))$.

Proof: At the top level, we have 7 operations adding/subtracting $n/2 \times n/2$ matrices, which takes $O(n^2)$ time. At each subsequent level of the recursion, we increase the number of subproblems by a constant factor (7), and the subproblems only become smaller. Therefore, the running time per level can only increase by a constant factor. By induction, since for the top level it is $O(n^2)$, it will be $O(n^2)$ for all levels. There are $O(\log(n))$ levels, so in total the running time is $T(n) = O(n^2 \log(n))$.

What is the error in this reasoning?

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

5. On an island, there are trustworthy tarsiers and tricky tarsiers. (Tarsier, by the way, is an insectivorous primate of the family Tarsiidae, having very large eyes and long feet, native mainly to several islands of Southeast Asia.) The trustworthy tarsiers always tell the truth; the tricky tarsiers may lie or may tell the truth. The tarsiers themselves can tell who is tricky and who is trustworthy, but an outsider can't tell the difference: they all just look like tarsiers.

You arrive on this island, and are tasked with finding the trustworthy tarsiers. You are allowed to pair up the tarsiers and have them evaluate each other. For example, if Tiffany the tarsier and Tomás the tarsier are both Trustworthy tarsiers, then they will both say that the other is trustworthy. But if Tiffany the tarsier is a Trustworthy tarsier and Tyrannus the tarsier is a Tricky tarsier, then Tiffany will call Tyrannus out as tricky, but Tyrannus may say either that Tiffany is tricky or that she is trustworthy. We will refer to one of these interactions as a "tarsier-to-tarsier comparison." The outcomes of comparing tarsiers A and B are as follows:

tarsier A	tarsier B	A says (about B)	B says (about A)
Trustworthy	Trustworthy	Trustworthy	Trustworthy
Trustworthy	Tricky	Tricky	Either
Tricky	Trustworthy	Either	Tricky
Tricky	Tricky	Either	Either

Suppose that there are n tarsiers on the island, and that there are strictly more than $n/2$ trustworthy tarsiers.

In this problem, you will develop an algorithm to find all of the trustworthy tarsiers, that only uses $O(n)$ tarsier-to-tarsier comparisons. Before you start this problem, think about how you might do this—hopefully it's not at all obvious! Along the way, you will also practice some of the skills that we've seen in Week 1. You will design two algorithms, formally prove that one is correct using a proof by induction, and formally analyze the running time of a recursive algorithm.

(a) **(4 pt.)** Give a straightforward algorithm that uses $O(n^2)$ tarsier-to-tarsier comparisons and identifies all of the trustworthy tarsiers.

[**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.]

(b) **(4 pt.)** * Now let's start designing an improved algorithm. The following proce-

*This is the trickiest part of the problem set! You may have to think a while.

dure will be a building block in our algorithm—make sure you read the requirements carefully!

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

- **Input:** A population of n tarsiers, where n is even, so that there are strictly more than $n/2$ trustworthy tarsiers in the population.
- **Output:** A population of m tarsiers, for $0 < m \leq n/2$, so that there are strictly more than $m/2$ trustworthy tarsiers in the population.
- **Constraint:** The number of tarsier-to-tarsier comparisons 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.]

(c) [This part is **NOT REQUIRED**, but you may assume it for future parts.] Extend your argument for odd n. That is, given a procedure that does the following:

- **Input:** A population of n tarsiers, where n is odd, so that there are strictly more than $n/2$ trustworthy tarsiers in the population.
- **Output:** A population of m tarsiers, for $0 < m \leq \lceil n/2 \rceil$, so that there are strictly more than $m/2$ trustworthy tarsiers in the population.
- **Constraint:** The number of tarsier-to-tarsier comparisons is no more than $\lfloor n/2 \rfloor$.

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

(d) **(2 pt.)** Using the procedures from parts (b) and (c), design a recursive algorithm that uses $O(n)$ tarsier-to-tarsier comparisons and finds a *single* trustworthy tarsier.

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

(e) **(4 pt.)** Prove formally, using induction, that your answer to part (d) is correct.

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

(f) **(2 pt.)** Prove that the running time of your procedure in part (d) uses $O(n)$ tarsier-to-tarsier comparisons.

[**We are expecting:** A formal argument. Note: do this argument "from scratch," do not use the Master Theorem.]

(g) **(2 pt.)** Give a procedure to find *all* trustworthy tarsiers using $O(n)$ tarsier-to-tarsier comparisons.

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