CS 161 (Stanford, Winter 2026) Homework 2

Style guide and expectations: We do NOT accept handwritten solutions. Please see the "Homework" part of the "Resources" section on the webpage for guidance on what we look for in homework solutions. We will grade according to these standards. You should cite all sources you used outside of the course material. Please do not distribute this material on any public forum.
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!

1 Exercise: Solving Recurrence Relations

1.1 (4 pt.)

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

$$T(n) = 6T(n/3) + n^3$$

[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:

$$T(n) = 4T(n/5) + n$$

[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 hand-draw and upload an image of your recursion tree (for LaTeX, use includegraphics)**]**.

2 Exercise: Big O via Induction

The Fibonacci numbers are a famous sequence defined by

$$F(0) = 0,\ F(1) = 1,\ F(n+2) = F(n) + F(n+1) \text{ for } n \geq 0.$$

For example, the first few Fibonacci numbers are

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ \ldots$$

In this problem, we will use induction to get a sense for how quickly the Fibonacci numbers grow.

(a) **(4 pt.)** Show that $F(n) = O(2^n)$.

(b) **(4 pt.)** Show that $F(n) = \Omega(1.5^n)$.

(c) **(0 pt.)** Show that $F(n) = \Theta(\varphi^n)$, where

$$\varphi = \frac{1+\sqrt{5}}{2}$$

is the golden ratio.

[We are expecting: For part (a) and (b), a formal proof by induction. Make sure your base case, inductive step, and conclusion are clearly identifiable from your proof. For part (c), nothing! This part is optional, and significantly harder than the previous two parts, but it may be rewarding to characterize the precise asymptotic growth rate of the Fibonacci numbers.**]**

3 Exercise: Modified MergeSort

Let's see how changing the size of the subproblems affects the runtime of MergeSort.

(a) **(3 pt.)** Your friend gives you this modified version of MergeSort, and he claims that it runs asymptotically better than the version of MergeSort we showed in class. Is he correct in his claim? Write down a recurrence relation and runtime for this version of MergeSort. (`Merge` is the same as we saw in lecture)

```
MergeSortThirds(A):
  n = len(A)
  if(n <= 1):
    return A

  L = MergeSortThirds(A[:n/3])
  M = MergeSortThirds(A[n/3:2n/3])
  R = MergeSortThirds(A[2n/3:])

  temp = Merge(L,M)
  return Merge(temp,R)
```

(b) **(4 pt.)** Inspired by your friend's idea, Pepper the Peculiar Penguin (Plucky's little sibling) decides to write the following version of MergeSort:

```
MergeSortN(A):
  n = len(A)
  if n <= 1:
      return A

  sortedSublists = []
  for i = 0,...,n-1
      temp = MergeSortN(A[i:i+1] )
      sortedSublists.append(temp)


  sortedA = []
  for each sublist in sortedSublists:
    sortedA = Merge(sortedA, sublist)

  return sortedA
```

Pepper says that this version of MergeSort is faster than the version we saw in lecture. Pepper's argument is as follows:

> "This modified MergeSort splits the array into n-subproblems of size O(1) immediately. We therefore don't waste time with the 'log(n) levels' worth of splitting. Additionally, in this modified sort, we're calling `Merge` on a bunch of sublists of size 1. Each merge would therefore take time
>
> $$O(\text{size of sublist A}) + O(\text{size of sublist B}) = O(1) + O(1) = O(1)$$
>
> That's constant time per merge! Yay! We now have an algorithm that should sort the array A in $O(n)$ time-n merges of $O(1)$ time each."

Sadly, Pepper's analysis is wrong. It was a good try, though!

Let's help a penguin out: Tell Pepper what her mistake was and explain what the true runtime of this modified MergeSort truly is.

[We are expecting: For part a), a recurrence relation for `MergesortThirds` and an explanation of its runtime. For part b), an explanation of Pepper's incorrect reasoning and an explanation of the true runtime of `MergesortN`**]**

4 Faster Exponentiation

This problem will investigate methods for calculating exponents of the form a^b where b is a large integer.

(a) **(1 pt.)** How many multiplications are needed to compute a^b by multiplying a by itself b times?

(b) **(3 pt.)** Come up with a divide and conquer algorithm to calculate a^b using $O(\log b)$ multiplications.

(c) **(3 pt.)** Fermie the Ferret, a visitor to the CS 161 town, claims that they came up with a truly marvelous algorithm that computes a^b using only $\log \log b$ multiplications. Sadly, Fermie's time in town is too short to tell you how the algorithm works. Explain why Fermie must have made a mistake.

Hint: Think inductively about the largest power you can compute using n multiplications.

Bonus: (no extra points) Can you make your argument work to rule out $O(\log \log b)$ multiplications, not just $\log \log b$?

[We are expecting: For part (a), a number. For part (b), an algorithm in pseudocode and an explanation of why it requires $O(\log b)$ multiplications. For part (c), an explanation of why it is impossible to compute a^b using $\log \log b$ multiplications.**]**

5 Quagga Trouble

You stumble across a secret zoo where questionable experiments have been taking place. To your astonishment, you find a field full of what appear to be quaggas!

Unfortunately, you quickly see that the zebras at this zoo have escaped their field, run through some mud, and are mingling with the quaggas. Your goal in this problem is to distinguish all quaggas from zebras.

Not being an animal expert, you can't tell the difference between a quagga and a muddy zebra. However, you can gather a pair of animals and ask them to evaluate each other. Zebras, being herd animals, can always tell if another animal is a zebra or not, and will let you know. These quaggas, however, have just been rescued from over a century of extinction, and are too new to the world to give you an accurate response. For example, if Zamantha and Zathaniel the zebras evaluate each other, they will both say that the other is a zebra. But if Zamantha and Quincy the quagga evaluate each other, then Zamantha will say that Quincy is a quagga, but Quincy may say either that Zamantha is a zebra or a quagga. We will refer to one of these interactions as a "quagga evaluation". The outcomes of quagga evaluations are as follows:

Animal A	Animal B	A says (about B)	B says (about A)
Zebra	Zebra	Zebra	Zebra
Zebra	Quagga	Quagga	Either
Quagga	Zebra	Either	Quagga
Quagga	Quagga	Either	Either

Suppose that there are n animals in the zoo, and that strictly more than $n/2$ of them are zebras.

(a) **(4 pt.)** Give an algorithm that uses $O(n^2)$ quagga evaluations and identifies all of the quaggas.

[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) **(8 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 $n/2$ quagga evaluations, 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 animals (zebras and quaggas), where n is even, so that there are strictly more than $n/2$ zebras in the population.
- **Output:** A population of m animals, for $0 < m \leq n/2$, so that there are strictly more than $m/2$ zebras in the population.
- **Constraint:** The number of quagga evaluations used by your procedure 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) **(0 pt.) [This problem 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 animals, where n is odd, so that there are strictly more than $n/2$ zebras in the population.
- **Output:** A population of m animals, for $0 < m \leq \lceil n/2 \rceil$, so that there are strictly more than $m/2$ zebras in the population.
- **Constraint:** The number of quagga evaluations used by your procedure 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) **(4 pt.)** Using the procedures from parts (b) and (c), design a recursive algorithm that uses $O(n)$ quagga evaluations and finds a *single* zebra.

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

(e) **(6 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.**]**

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

(f) **(6 pt.)** Prove that your algorithm in part (d) uses $O(n)$ quagga evaluations. If you find that you are working with floors and ceilings, you may ignore them (i.e. assume that the quantity is an integer).

[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* quaggas using $O(n)$ quagga evaluations.

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