Written Problem Set 2: Divide and Conquer

The first thing you should do in ps1.tex is set up your name as the author who is submitting by replacing the line, `\submitter{TODO: your name}`, with your name and UVA email id, e.g., `\submitter{Grace Hopper (gmh1a)}`.

Next, put your response for problem 1 into problem1.tex, problem 2 into problem2.tex, etc.

Before submitting , also remember to:

– List your collaborators and resources, replacing the TODO in `\collaborators{TODO: replace ...}` with your collaborators and resources. (Remember to update this before submitting if you work with more people.)

– Replace the second line in ps2.tex, `\usepackage{algo}` with `\usepackage[response]{algo}`.

Collaborators: TODO: replace this with your collaborators (if you did not have any, replace this with *None*)

Resources: TODO: replace this with your resources (if you did not have any, replace this with *None*)

Problem 1: Flights

You have been hired to plan the flights for Professor Floryan's brand new passenger air company, "Receding Airlines". You are going to provide service to n cities. This airline will only service the United States, and will only fly you North.

You recognize that in order to enable all your passengers to travel from any city to any other city (to the North) with a single flight requires $\Omega(n^2)$ different routes. Prof. Floryan says that the airline cannot be profitable when supporting so many routes. Another option would be to order the cities in a list (from South to North), and have flights that go from the city at index i, to the city at index $i+1$. This, however, would mean some passengers would require $\Omega(n)$ connections to get to their destination. Devise a compromise set of routes which requires no passenger have more than a single connection (i.e. must take at most two flights), and requires no more than $O(n \log n)$ routes. Justify that it satisfies these requirements.

Base Case. Describe your base case here. Be sure to mention what maximum input size defines your base case and also what your algorithm returns.

Divide Step. Describe the divide step of your algorithm here, making sure to describe the sub-problems (including their sizes) that are produced. Include the total number of routes created during the divide step .

Combine Step. Describe the combine step of your algorithm here. Include the total number of routes created during this combine step.

Algorithm. Summarize your whole algorithm here.

Correctness. Justify the correctness of your algorithm. Namely, show that the selected routes have the property that one can travel from any city to an Northward city with at most two flights.

Number of Routes. Express the number of routes your algorithm selects as a recurrence relation. Describe, using the master theorem, how you know that this recurrence is $O(n \log n)$.

Problem 2a: Fast Exponentiation

Given a pair of positive integers (a, n), devise a divide and conquer algorithm that computes a^n using only $O(\log n)$ calls to a multiplication routine. You need to show that the algorithm is correct (i.e. it always produces the right answer) and also that it only uses $O(\log n)$ multiplications.

Base Case. Describe your base case here. Be sure to mention what maximum input size defines your base case and also what your algorithm returns.

Divide Step. Describe the divide step of your algorithm here, making sure to mention what subproblems are produced. Include its worst case running time.

Combine Step. Describe the combine step of your algorithm here. Include its worst case running time.

Algorithm. Summarize your whole algorithm here

Correctness. Prove the correctness of your algorithm. Namely, show that it always produces the correct answer.

Running Time. Express the running time of your algorithm as a recurrence. Describe, using the tree method, how you know that this recurrence is $O(\log n)$

Problem 2b: Fast Matrix Exponentiation

Computer graphics software typically represents points in n dimensions as $(n+1)$-dimensional vectors. To make transformations on the points (e.g. rotating a modelled figure, zooming in, or making the figure appear as though seen through a fish-eye lens) we use a $(n+1) \times (n+1)$ matrix which defines the transformation, and then we multiply each vector by this matrix to transform that point. Let's say we are developing software for very high dimension graphics (n dimensions), and we have a transformation that we would like to apply to a particular point n times. Develop an algorithm which can multiply this $(n+1)$-dimensional point by this $(n+1) \times (n+1)$ transformation matrix n times in $o(n^3)$ (little-oh of n^3) time. In summary, your algorithm should compute $T \times T \times T \times ... \times T \cdot v = T^n \cdot v$.

Note: graphics using more than 3 dimensions graphics is not actually absurd. As an example, I may have points that are in 3d space, and then each point has a color, requiring a vector of length 7 to represent.

Algorithm. Summarize your whole algorithm here

Running Time. Express the running time of your algorithm. Justify why the running time given matches that of the algorithm you described,, and demonstrate that this function is $o(n^3)$.

Problem 3: Iterated Functions

When solving recurrence relations using the tree method, we will need to determine the height of the tree. For instance, consider the Merge Sort Algorithm from class. The depth of our recursion in this case was $\lceil \log_2 n \rceil$ for a list of length n (i.e., the number of times we needed to divide n by 2 until hitting a base case—a value ≤ 1). For this problem, we will formalize and generalize this idea of iterative re-application of a function until a target value is reached. This will help us determine the recursive depth of algorithms.

Definition 1 *Define the iterated function* $f_c^*(n) = \min\{i \geq 0 \mid f^{(i)}(n) \leq c\}$ *where* f *is an increasing function,* c *is a fixed constant, and for an integer* $i > 1$, $f^{(i)}(n) = f(f^{(i-1)}(n))$, *e.g.,* $f^{(3)}(n) = f(f(f(n)))$.

In other words, for a given function $f(n)$, we say that $f_c^*(n)$ gives the count of times $f(n)$ must be repeatedly applied to its own output before the result is not more than c. In the definition above, this is the smallest value of i such that $f^{(i)}(n) \leq c$.

In the Merge Sort algorithm, we could describe the depth of our recursion using this new iterated function notation with $f(n) = \frac{n}{2}$ and $c = 1$, giving $f_1^*(n) = \lceil \log_2 n \rceil$.

Fill in the table below. For each of the given functions $f(n)$ and constants c, give as tight a bound as possible on $f_c^*(n)$ (we want a precise bound, not an asymptotic one). The first row is done for you. Justify each answer.

$f(n)$	c	$f_c^*(n)$
$n/2$	1	$\lceil \log_2 n \rceil$
$n-1$	0	
$n/2$	2	
$\sqrt{n}$	2	
$\sqrt{n}$	1	
$n^{1/3}$	2	

Justification: Briefly explain your answer for each row.

Problem 4: Solving Recurrences

Solve, i.e. provide a tight Θ (big-Theta) bound on the following recurrences using the indicated method. You may use any base cases you'd like.

1. $T(n) = 37 \cdot T(n/23) + n$ (using Master Theorem)
2. $T(n) = T(n/3) + T(n/4) + T(n/12) + n$ (using Guess and Check)
3. $T(n) = T(n/2) + 2$ (using Master Theorem)

