Directions:

- Write your solution using \LaTeX. Submit this homework as a \texttt{.tex} file using \texttt{github}.
- Try to make your answers as clear and concise as possible.
- This is an individual homework. It’s ok to discuss approaches and strategies at a high level. However, you should not reveal specific details of a solution, nor should you show your written solution to anyone else. The only exception to this rule is work you’ve done with a lab partner \textit{while in lab}. In this case, note who you’ve worked with and what parts were solved during lab.

1. **Testing Glass Jars** (Kleinberg and Tardos 2.8)

   You’re doing some stress-testing on various models of glass jars to determine the height from which they can be dropped and still not break. The setup for this experiment, on a particular type of jar, is as follows. You have a ladder with $n$ rungs, and you want to find the highest rung from which you can drop a copy of the jar and not have it break. We call this the \textit{highest safe rung}.

   It might be natural to try binary search: drop a jar from the middle rung, see if it breaks, and then recursively try from rung $n/4$ or $3n/4$ depending on the outcome. But this has the drawback that you could break a lot of jars in finding the answer.

   If your primary goal were to conserve jars, on the other hand, you could try the following strategy. Start by dropping a jar from the first rung, then the second rung, and so forth, climbing one higher each time until the jar breaks. In this way, you only need a single jar — at the moment it breaks, you have the correct answer — but you may have to drop it $n$ times (rather than $\log n$ as in the binary search solution).

   So here is the tradeoff: it seems you can perform fewer drops if you’re willing to break more jars. To understand better how this tradeoff works at a quantitative level, let’s consider how to run this experiment given a fixed “budget” of $k \geq 1$ jars. In other words, you have to determine the correct answer — the highest safe rung — and can use at most $k$ jars in doing so.

   (a) Suppose you are given a budget of $k = 2$ jars. Describe a strategy for finding the highest safe rung that requires you to drop a jar at most $f(n)$ times, for some function $f(n)$ that grows slower than linearly.

   (b) Now suppose you have a budget of $k > 2$ jars, for some given $k$. Describe a strategy for finding the highest safe rung using at most $k$ jars. If $f_k(n)$ denotes the number of times you need to drop a jar according to your strategy, then the functions $f_1, f_2, f_3, \ldots$ should have the property that each one grows asymptotically slower than the previous one: for each $k$,

   $$\lim_{n \to \infty} \frac{f_k(n)}{f_{k-1}(n)} = 0.$$
2. **Asymptotic analysis.**

For these problems, your example functions should have domain and range the positive integers $\mathbb{N}$.

(a) Show that if $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$, then $f(n)$ is $O(h(n))$.

(b) Give a proof or counterexample: If $f$ is not $O(g)$, then $g$ is $O(f)$.

(c) Let $k$ be a fixed constant and suppose that $f_1, \ldots, f_k$ and $h$ are functions such that $f_i = O(h)$ for all $i$.

i. Let $g_1(n) := f_1(n) + \ldots + f_k(n)$. Is $g_1 = O(h)$? Prove or give a counterexample. If you give a counterexample, give an alternate statement that best captures the relationship between the two functions.

ii. Let $g_2(n) := f_1(n) \cdot \ldots \cdot f_k(n)$. Is $g_2 = O(h)$? Prove or give a counterexample. If you give a counterexample, give an alternate statement that best captures the relationship between the two functions.

3. **Close-to-sorted.** Say that a list of numbers is “$k$-close-to-sorted” if each number in the list is less than $k$ positions from its actual place in the sorted order. (So a 1-close-to-sorted list is actually sorted.) Give an $O(n \log k)$ algorithm for sorting a list of numbers that is $k$-close-to-sorted.

   In your algorithm, you may use any data structure or algorithm from CS35 by name, without describing how it works.

4. **Extra credit.** For these problems, your example functions should have domain and range the positive integers $\mathbb{N}$.

   • Find (with proof) a function $f_1$ such that $f_1(2n)$ is $O(f_1(n))$.
   • Find (with proof) a function $f_2$ such that $f_2(2n)$ is not $O(f_2(n))$.

5. **Extra credit.** Define a function $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$ such that $f = O(g)$ for all exponential functions $g$, but $f$ is *not* $O(h)$ for any polynomial function $h$. 

2