The Master Theorem For a recurrence like $T(n) = aT(\frac{n}{b}) + f(n)$, where $a \geq 1$ and $b > 1$, the Master Theorem states three cases:

1. If $f(n) \in O(n^c)$ where $c < \log_b a$ then $T(n) \in \Theta(n^{\log_b a})$.
2. If for some constant $k \geq 0$, $f(n) \in \Theta(n^c (\log n)^k)$ where $c = \log_b a$, then $T(n) \in \Theta(n^c (\log n)^{k+1})$.
3. If $f(n) \in \Omega(n^c)$ where $c > \log_b a$ **and** $af(\frac{n}{b}) \leq kf(n)$ for some constant $k < 1$ and sufficiently large n, then $T(n) \in \Theta(f(n))$.

1 I'm a Lumberjack (And I'm Okay)

Your task is to design algorithms to solve the following problems. For full credit, your algorithm must run in logarithmic time.

1.1 Root-finding

Given a positive number n and a (user-specified) error tolerance e, you want to approximate the square root of n to within error tolerance e. Specifically, you want to return an $x \approx \sqrt{n}$ that satisfies $|x^2 - n| \leq e$. For example, to compute the square root of $n = 2$ with $e = 0.01$, an acceptable answer would be $x = 1.414$, because $1.414^2 = 1.999396$. Note that $x = 1.415$ is also acceptable, as $1.415^2 = 2.002225$, but you only need to return a single answer.

Assume that you can't perform any arithmetic functions other than addition, subtraction, multiplication, and division.

1. Write pseudocode for an efficient algorithm to solve this problem.

2. Briefly justify a good asymptotic bound on the runtime of your algorithm in terms of n (i.e., give a runtime in terms of n assuming a fixed value of e).