A. We've shown in a previous assignment that $n^2 = \mathcal{O}(2^n)$. That's a pretty mild example of a wider phenomenon: Any polynomial function is asymptotically bounded by any exponential function. For example, $n^{1000000} = \mathcal{O}(1.000001^n)$! Prove it. That is, prove that $n^k = \mathcal{O}(b^n)$ for any integer $k \geq 0$ and any real number $b > 1$.

B. Here's the usual recursive algorithm for computing the Fibonacci numbers $F_n$, implemented in Python:

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

I've claimed in class (or I will soon) that `fib(n - k)` appears a total of $F_{k+1}$ times in the call tree for `fib(n)`. State this idea precisely, with the correct quantifiers and bounds on $n$ and $k$, and prove it.

C. Using Problem B and a previous assignment, prove that the running time of `fib(n)` is $\Omega((3/2)^n)$.

Matrices are covered in Section 2.4.2 of our DLN textbook, but here's the short version. A *matrix* is a rectangular grid of numbers. A matrix with $p$ rows and $q$ columns is said to be a $p \times q$ matrix. If $M$ is a $p \times q$ matrix, then for any $i$ and $j$ (satisyfing $1 \leq i \leq p$ and $1 \leq j \leq q$), $M_{ij}$ denotes the number in the $i$th row (counting from the top) and $j$th column (counting from the left) of $M$. There are three basic operations on matrices:

- If $M$ is a $p \times q$ matrix and $c$ is a number, then there is a *scalar product* matrix $cM$, which is also $p \times q$, defined by multiplying each entry of $M$ by $c$. In other words,

$$(cM)_{ij} = cM_{ij}.$$

- If $M$ and $N$ are both $p \times q$ matrices, then there is a *sum* matrix $M + N$, which is also $p \times q$, defined by adding the corresponding entries of $M$ and $N$. That is,

$$(M + N)_{ij} = M_{ij} + N_{ij}.$$

- If $M$ is a $p \times q$ matrix and $N$ is a $q \times r$ matrix, then there is a *product* matrix $MN$, which is $p \times r$, defined by

$$(MN)_{ij} = \sum_{k=1}^{q} M_{ik} N_{kj}.$$

  Here's another way to think of it. The $(i, j)$th entry of $MN$ is what you get by multiplying the $i$th row of $M$ by the $j$th column of $N$, entry by entry, and then summing up those products.

Matrices are flabbergastingly important in mathematics, statistics, physics, and a wide variety of computational problems. So fast algorithms for computing with matrices are highly desirable.

D. What is the asymptotic running time of multiplying two $n \times n$ matrices using the formula above? (Your elementary operations are additions and multiplications of ordinary numbers.)

Here's another way to think of multiplying two $n \times n$ matrices $M$ and $N$. Assume that $n$ is even. Cut $M$ and $N$ each into four quarters, labeled $M^{11}, M^{12}, \ldots$, like this:

$$M = \begin{bmatrix} M^{11} & M^{12} \\ M^{21} & M^{22} \end{bmatrix}, \; N = \begin{bmatrix} N^{11} & N^{12} \\ N^{21} & N^{22} \end{bmatrix}.$$

Each of these quarters $M^{11}, M^{12}, \ldots$ is an $n/2 \times n/2$ matrix. (In case you're confused, the superscripts 11, 12, etc. are not exponents. They're just superscripts, that denote which quarter of $M$ or $N$ we're looking at. I'd use subscripts for this purpose, but we're already using subscripts to denote the individual entries of $M$ and $N$. For example, $M_{11}$ is the left-upper-most entry in $M$, but $M^{11}$ is the left upper quarter of $M$.) It turns out that

$$MN = \begin{bmatrix} (MN)^{11} & (MN)^{12} \\ (MN)^{21} & (MN)^{22} \end{bmatrix} = \begin{bmatrix} M^{11}N^{11} + M^{12}N^{21} & M^{11}N^{12} + M^{12}N^{22} \\ M^{21}N^{11} + M^{22}N^{21} & M^{21}N^{12} + M^{22}N^{22} \end{bmatrix}.$$

That is, you can compute $MN$ by computing various sums of products of matrices half as big as $M$ and $N$. This suggests a recursive, divide-and-conquer algorithm for multiplying matrices.

E. For convenience, assume that $n$ is a power of 2. Describe this divide-and-conquer algorithm explicitly. Write a recurrence relation $T(n)$ for its running time when given two $n \times n$ matrices. Use the master theorem for divide-and-conquer algorithms to compute $T(n)$.

Here's where things get clever. Define seven new $n/2 \times n/2$ matrices $A, B, C, \ldots, G$ as follows.

$$\begin{aligned} A &= (M^{11} + M^{22})(N^{11} + N^{22}), \\ B &= (M^{21} + M^{22})N^{11}, \\ C &= M^{11}(N^{12} - N^{22}), \\ D &= M^{22}(N^{21} - N^{11}), \\ E &= (M^{11} + M^{12})N^{22}, \\ F &= (M^{21} - M^{11})(N^{11} + N^{12}), \\ G &= (M^{12} - M^{22})(N^{21} + N^{22}). \end{aligned}$$

Then it turns out that

$$MN = \begin{bmatrix} (MN)^{11} & (MN)^{12} \\ (MN)^{21} & (MN)^{22} \end{bmatrix} = \begin{bmatrix} A + D - E + G & C + E \\ B + D & A - B + C + F \end{bmatrix}.$$

This suggests a slightly different recursive, divide-and-conquer algorithm for multiplying matrices.

F. For convenience, assume that $n$ is a power of 2. Describe this divide-and-conquer algorithm explicitly. Write a recurrence relation $T(n)$ for its running time. Use the master theorem to compute $T(n)$.