1. Suppose you are given a six-sided die, that might be biased in an unknown way. Explain how to use die rolls to generate unbiased coin flips, and determine the expected number of die rolls until a coin flip is generated. Now suppose you want to generate unbiased die rolls (from a six-sided die) given your potentially biased die. Explain how to do this, and again determine the expected number of biased die rolls until an unbiased die roll is generated. For both problems, you need not give the most efficient solution; however, your solution should be reasonable, and exceptional solutions will receive exceptional scores.

2. On a platform of your choice, implement the three different methods for computing the Fibonacci numbers (recursive, iterative, and matrix) discussed in lecture. Use integer variables. How fast does each method appear to be? Give precise timings if possible. (This is deliberately open-ended; give what you feel is a reasonable answer. You will need to figure out how to time processes on the system you are using, if you do not already know.) Can you determine the first Fibonacci number where you reach integer overflow? (If your platform does not have integer overflow – lucky you! – you might see how far each process gets after five minutes.)

   Since you should reach integer overflow with the faster methods quite quickly, modify your programs so that they return the Fibonacci numbers modulo $65536 = 2^{16}$. (In other words, make all of your arithmetic modulo 2^{16} – this will avoid overflow! You must do this regardless of whether or not your system overflows.) For each method, what is the largest value of k such that you can compute the kth Fibonacci number (or the [kth Fibonacci number] modulo 65536) in one minute of machine time?

   Submit your source code with your assignment. Please give a reasonable English explanation of your experience with your program(s).

3. Indicate for each pair of expressions (A, B) in the table below the relationship between A and B. Your answer should be in the form of a table with a "yes" or "no" written in each box. For example, if A is $O(B)$, then you should put a "yes" in the first box.

A	B	O	o	Ω	ω	Θ
$\log n$	$\log(n^2)$	*no*	*no*	*yes*	*yes*	*yes*
$\log(n!)$	$\log(n^n)$	*yes*	*yes*	*no*	*no*	*no*
$\sqrt[3]{n}$	$(\log n)^6$	*no*	*no*	*yes*	*no*	*no*
$n^2 2^n$	3^n	*yes*	*no*	*yes*	*no*	*yes*
$(n^2)!$	n^n	*no*	*no*	*yes*	*yes*	*no*
$\frac{n^2}{\log n}$	$n \log(n^2)$	*no*	*no*	*yes*	*yes*	*no*
$(\log n)^{\log n}$	$\frac{n}{\log(n)}$	*yes*	*yes*	*no*	*no*	*no*
$100n + \log n$	$(\log n)^3 + n$	*yes*	*yes*	*no*	*no*	*no*

4. For all of the problems below, when asked to give an example, you should give a function mapping positive integers to positive integers. (No cheating with 0's!)

   - Find (with proof) a function f_1 such that $f_1(2n)$ is $O(f_1(n))$.
     Let $f_1(n) = n$. $f_1(2n) = 2n = 2 * n = 2 * f_1(n)$. Therefore $f_1(2n) \leq 2f_1(n)$ for all $n \geq 1$.
   - Find (with proof) a function f_2 such that $f_2(2n)$ is not $O(f_2(n))$.
     Let $f_2(n) = n!$
   - Prove that if $f(n)$ is $O(g(n))$, and $g(n)$ is $O(h(n))$, then $f(n)$ is $O(h(n))$.
     Because $f(n)$ is $O(g(n))$ then there exists some positive c_1 and N_1 such that for all $n \geq N_1, f(n) \leq c_1 g(n)$. Similarly because $g(n)$ is $O(h(n))$ then there exists some c_2, N_2 such

that for all $n \geq N_2, g(n) \leq c_2h(n)$. Let $c = c_1c_2$ and let $N = max(N_1, N_2)$. Thus for all $n \geq N, f(n) \leq c_1g(n) \leq c_1g(n) \leq c_1c_2h(n) = ch(n)$. Therefore $f(n)$ is $O(h(n))$.

- Give a proof or a counterexample: if f is not $O(g)$, then g is $O(f)$.
  Counterexample: Consider $f(x) = sin(x)$ and $g(x) = 1$
- Give a proof or a counterexample: if f is $o(g)$, then f is $O(g)$.
  Proof: Because f is $o(g)$ then there exists some N such that for all $n \geq N$ then $f \leq cg$ for all $c > 0$. Pick $c_1 = 1$. Since $c_1 > 0$ then $f \leq c_1g$ for all $n \geq N$. Therefore f is $O(g)$.

5. **Do not turn this in. This is a suggested exercise.**
   InsertionSort is a simple sorting algorithm that works as follows on input $A[0], \ldots, A[n-1]$.

```
InsertionSort($A$)
    for $i = 1$ to $n - 1$
      $j = i$
      while $j > 0$ and $A[j-1] > A[j]$
        swap $A[j]$ and $A[j-1]$
        $j = j - 1$
```

Show that for any function $T = T(n)$ satisfying $T(n) = \Omega(n)$ and $T(n) = O(n^2)$ there is an infinite sequence of inputs $\{A_k\}_{k=1}^{\infty}$ such that A_k is an array of length k, and if $t(n)$ is the running time of InsertionSort on A_n, then the order of growth of $t(n)$ is $\Theta(T(n))$.