Homework will be due at the **BEGINNING** *of class on the date due. Your answers should be written legibly if handwritten and your name should be clearly written on your homework assignment. Try to make your answers as clear and concise as possible; style will count in your overall mark. Be sure to read and know the collaboration policy in the course syllabus.*

Assignments are expected to be turned in electronically. A single zip file should be submitted containing a) a writeup ***in pdf format*** *and b) any Python programs that you are asked to provide. If you do assignments by hand, you will need to scan in your results to turn them in. Instructions for how to electronically submit assignments will be given in class and on the class website.*

For all homework problems where you are asked to give an algorithm, you must prove the correctness of your algorithm and establish the best upper bound that you can give for the running time. You should always write a clear informal description of your algorithm in English. You may also write pseudocode if you feel your informal explanation requires more precision and detail, but keep in mind pseudocode does NOT substitute for an explanation. Again, try to make your answers as clear and concise as possible.

1. In Python, implement the three different methods (recursive, iterative, and matrix) for computing the Fibonacci numbers discussed in lecture (see Mitzenmacher's lecture notes linked to on the course website for reference). Please call these programs recursive.py, iterative.py, and matrix.py respectively.

   Make sure you are computing the numbers exactly (no floating point arithmetic). How far does each process get after one minute of machine time? You will need to figure out how to time processes on the system you are using, if you do not already know.

   To help ascertain how much of the runtime is due to the complexity of multiplying large integers, 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}). For each method, what is the largest Fibonacci number you can compute in one minute of machine time?

   Submit your source code with your assignment. Please give a reasonable English explanation of your experience with your programs.

   If you use NumPy on this problem, you will lose points. This is because NumPy is largely written in C, and merely wrapped in Python for ease of use. Whereas Python contains native support for large-integer arithmetic, C does not. This means that NumPy will silently experience integer overflow errors when multiplying or adding large enough numbers. It's a good exercise to try this out and see the errors for yourself.

2. 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
$\log n$	$\log(n^2)$					
$\sqrt[3]{n}$	$(\log n)^6$					
$n^2 2^n$	3^n					
$\frac{n^2}{\log n}$	$n \log(n^2)$					
$(\log n)^{\log n}$	$\frac{n}{\log(n)}$					
$100n + \log n$	$(\log n)^3 + n$					

3. 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 $O(f_1(n))$.
   - Find (with proof) a function f_2 such that $f_2(2n)$ is not $O(f_2(n))$.
   - Prove that if $f(n)$ is $O(g(n))$, and $g(n)$ is $O(h(n))$, then $f(n)$ is $O(h(n))$.
   - Give a proof or a counterexample: if f is not $O(g)$, then g is $O(f)$.
   - Give a proof or a counterexample: if f is $o(g)$, then f is $O(g)$.

4. Buffy and Willow are facing an evil demon named Stooge, living inside Willow's computer. In an effort to slow the Scooby Gang's computing power to a crawl, the demon has replaced Willow's hand-designed super-fast sorting routine with the following recursive sorting algorithm, known as StoogeSort. For simplicity, we think of Stoogesort as running on a list of distinct numbers. StoogeSort runs in three phases. In the first phase, the first 2/3 of the list is (recursively) sorted. In the second phase, the final 2/3 of the list is (recursively) sorted. Finally, in the third phase, the first 2/3 of the list is (recursively) sorted again.

   Willow notices some sluggishness in her system, but doesn't notice any errors from the sorting routine. This is because StoogeSort correctly sorts. For the first part of your problem, prove rigorously that StoogeSort correctly sorts. To keep things simple, you may assume in your proof that the length of the input is divisible by three. Your proof should use induction.

   Unfortunately, StoogeSort can be slow. Derive a recurrence describing its running time, and use the recurrence to bound the asymptotic running time of Stoogesort.

5. Give asymptotic bounds for $T(n)$ in each of the following recurrences. You may invoke the Master Theorem. Hint: You may have to change variables somehow in the last one.

   - $T(n) = 4T(n/2) + n^3$.
   - $T(n) = 17T(n/4) + n^2$.
   - $T(n) = 9T(n/3) + n^2$.
   - $T(n) = T(\sqrt{n}) + 1$.

6. The traditional Devonian/Cornish drinking song "The Barley Mow" has the pseudolyrics described in the box below[1], where container$[i]$ is the name of a container that holds 2^i ounces of beer.[2]

[1] Pseudolyrics are to lyrics as pseudocode is to code.

[2] One version of the song uses the following containers: nipperkin, gill pot, half-pint, pint, quart, pottle, gallon, halfanker, anker, firkin, half-barrel, barrel, hogshead, pipe, well, river, and ocean. Every container in this list is twice as big as its predecessor, except that a firkin is actually 2.25 ankers, and the last three units are just silly.

```
BarleyMow(n):
  "Here's a health to the barley-mow, my brave boys,"
  "Here's a health to the barley-mow!"

  "We'll drink it out of the jolly brown bowl,"
  "Here's a health to the barley-mow!"
  "Here's a health to the barley-mow, my brave boys,"
  "Here's a health to the barley-mow!"

  for i ← 1 to n
      "We'll drink it out of the container[i], boys,"
      "Here's a health to the barley-mow!"
      for j ← i downto 1
          "The container[j],"
      "And the jolly brown bowl!"
      "Here's a health to the barley-mow!"
      "Here's a health to the barley-mow, my brave boys,"
      "Here's a health to the barley-mow!"
```

Suppose each container name container[i] is a single word, and you can sing four words a second. How long would it take you to sing BARLEYMOW(n)? (Give a tight asymptotic bound, and explain why the bound is correct.)

7. Implement Merge Sort in Python and turn in your source code (please call your program mergesort.py). The implementation should read a comma-separated list of numbers from stdin, sort the list, and print out the result. You may assume the length of the list is a power of two.

   DO NOT HAND IN ANY CODE THAT YOU DID NOT WRITE YOURSELF.