Homework will be due on Friday at 5pm. 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. Be sure to check the back of the page; problems occasionally show up there too!

Assignments are expected to be turned in electronically in pdf format. 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. Generally better running times will get better credit; generally exponential time algorithms (unless specifically asked for) will receive no or little credit. 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. Answers that consist solely of pseudocode will receive little or no credit. Again, try to make your answers as clear and concise as possible.

There is a (short) programming problem on this assignment; you should NOT WORK with others on this problem (e.g., write code together) like you will for the “major” programming assignments. (You may talk about the problem, as you can for other problems.)

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.

<table>
<thead>
<tr>
<th></th>
<th>(A)</th>
<th>(B)</th>
<th>(O)</th>
<th>(o)</th>
<th>(\Omega)</th>
<th>(\omega)</th>
<th>(\Theta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\log n)</td>
<td>(\log(n^2))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\log(n!))</td>
<td>(\log(n^n))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\sqrt{n})</td>
<td>((\log n)^{b})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(n^22^n)</td>
<td>(3^n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>((n^2)!)</td>
<td>(n^n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\frac{n^n}{\log n})</td>
<td>(n \log(n^2))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>((\log n)^{\log n})</td>
<td>(\frac{n}{\log(n)})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100n + log n</td>
<td>((\log n)^3 + n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 \(O(f_1(n))\).
- 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 \(O(g)\).

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 \(\Theta(T(n))\).