Optional Extra Make-up Homework
Show all work.

1. You have a chain of \( n \) switches numbered from 1 to \( n \). The chain has a random behavior such that when any switch number \( i \) is turned on \( (1 \leq i \leq n - 1) \), it either turns on the next switch number \( i + 1 \), or it stops the switching chain and outputs \( i \). If the chain reaches switch number \( n \), it stops and outputs \( n \). The probability that a switch will turn on the next switch is \( \alpha \), and the probability that it will stop the chain and output is \( 1 - \alpha \), where \( \alpha \) is a known parameter of the switching chain with \( 0 < \alpha < 1 \). You turn on switch 1 and wait for the chain of switches to output a number. What is the expected value of the output of the chain of switches? Your answer should be a formula in terms of \( \alpha \) and \( n \).

**Solution:** With probability \( (1 - \alpha) \) the chain stops at switch 1 and outputs 1. With probability \( \alpha \cdot (1 - \alpha) \), the chain reaches switch 2 and stops. With probability \( \alpha^2 \cdot (1 - \alpha) \), the chain reaches switch 3 and stops. In general, with probability \( \alpha^{k-1} \cdot (1 - \alpha) \), the chain reaches switch \( k \) and stops. The exception is switch \( n \), where the switch reaches and stops with probability \( \alpha^{n-1} \). Now we combine these probabilities with the outputs to compute the expected value. Let \( X \) be a random variable for the output of the chain. Then 
\[
\mathbb{E}(X) = \sum_{k=1}^{n-1} \alpha^{k-1} \cdot (1 - \alpha) + n \cdot \alpha^{n-1}
\]
is the expected value of the output.

2. Construct two functions \( f \) and \( g \) that are both positive and strictly increasing (that is \( i > k \) implies that \( f(i) > f(k) \) and similarly for \( g \)), such that neither \( f \) is \( O(g) \) nor \( g \) is \( O(f) \).

**Solution:** Many solutions are possible. The basic idea is that the behavior of the two functions oscillates such that neither permanently bounds the other. Here is one possible example:

\[
f(n) = \begin{cases} 
1 & \text{if } n = 0 \\
 f(n-1) + 1 & \text{if } n \text{ is even} \\
 f(n-1) + 2^n & \text{if } n \text{ is odd}
\end{cases}
\]

\[
g(n) = \begin{cases} 
1 & \text{if } n = 0 \\
g(n-1) + 2^n & \text{if } n \text{ is even} \\
g(n-1) + 1 & \text{if } n \text{ is odd}
\end{cases}
\]

These two functions recursively defined over nonnegative integers are both strictly increasing, because the recursive definition always adds something as \( n \) increases, and neither is bounded by a constant factor of the other, because \( f \) spurs ahead doubly exponentially in the odd case and \( g \) spurs ahead doubly exponentially in the even case.
3. Suppose you are given a list of \( n \) integers with many duplications. Suppose it is known that the number of distinct integers in the sequence is at most \( \lg(n) \). Give an algorithm to sort the sequence using \( O(n \cdot \lg(\lg(n))) \) comparisons. **Hint:** You may use the fact that balanced trees like height-balanced trees and red-black trees use height \( O(\lg(k)) \) to contain \( k \) keys, and build such a tree as part of a data structure in your algorithm.

**Solution:** The solution is an algorithm called “tree sort.” Build a balanced binary search tree over the list of integers. Each node of the binary search tree can contain an integer and its number of occurrences. Since there are \( \lg(n) \) distinct integers, the height of the tree will be \( \lg(\lg(n)) \) and so each insertion will use \( O(\lg(\lg(n))) \) comparisons. Thus, the \( n \) insertions to insert the list of integers will require \( O(n \cdot \lg(\lg(n))) \) comparisons. Then to produce the list in sorted order, perform an inorder traversal of the tree. Here is pseudocode for this idea:

```
Algorithm tree-sort(list L)
   allocate a new binary tree T with one empty root node
   for each element x in list L
      Search for x in T
      if x is not found
         add a new node to T containing x with 1 occurrence
         rebalance T
      else
         in the node containing x, increment number of occurrences
      end if
   end for
   for each node in T in an inorder traversal
      x := the integer stored in the node
      n := number of occurrences of x
      output n copies of x
   end for
end algorithm
```

4. You are given a graph in the array of adjacency lists representation. Describe an algorithm to determine whether the graph contains a “triangle,” that is a cycle of 3 vertices and 3 edges. Give a big-\( O \) bound for your algorithm in terms of \( e \), the number of edges, and \( n \), the number of vertices in the graph.

**Solution:** One possible solution is to perform a depth-first search on the graph, and with every visit to a vertex, record the immediate ancestor and the ancestor’s ancestor of the vertex in the depth-first search tree. If there is a triangle in the graph, the search will eventually expand from a vertex and find it’s ancestor’s ancestor. To see this, consider a triangle
consisting of the vertices $v_i, v_j$ and $v_k$. Since the labels are arbitrary, suppose without loss of generality the search first reaches $v_i$, and then expands from $v_i$ to $v_j$. Now $v_j$ records that $v_i$ is its ancestor. After this, at some point the search expands from $v_j$ to $v_k$ and records that $v_i$ is the ancestor’s ancestor of $v_k$. Finally, at some point the search attempts to expand from $v_k$ to $v_i$, and by comparing with its ancestor’s ancestor, it discovers the triangle. Conversely, clearly if the search discovers an edge from a vertex $v$ to its ancestor’s ancestor, the graph contains the triangle from $v$’s ancestor’s ancestor to $v$’s ancestor to $v$ and back again to $v$’s ancestor’s ancestor. The asymptotic bound on the algorithm is $O(n + e)$ as usual for depth-first search.