1. [7 points] Please give short answers to the following, giving reasons where called for. Unless a question says otherwise, time estimates refer to asymptotic bounds (\(O(\cdots), \Omega(\cdots), \Theta(\cdots)\)). Always give the simplest bounds possible (\(O(f(x))\) is “simpler” than \(O(f(x) + g(x))\) and \(O(Kf(x))\)).

a. 2-4 trees can have up to three (ordered) keys in each node and up to 4 children, with one key between each two consecutive children. Suppose we push these numbers up. If we put approximately \(\sqrt[6]{N} - 1\) keys in every node of a tree containing a total of \(N\) keys, the height (distance from the root to the empty leaves) will remain constant at about 6. We’ve said that the cost of searching a binary search tree is proportional to its height, so does this mean that the cost of searching one these “root-6 trees” is constant (that is, as \(N\) increases and the number of keys in each node grows as \(\sqrt[6]{N}\), does the cost of searching remain constant)? Why or why not? If not, how does the cost grow?

Clearly, this comes down to the cost of searching each node for the appropriate key. Assuming these keys are stored in an array, we get a cost of \(\Theta(lg \sqrt[6]{N}) = \Theta(lg N)\). (Since \(lg \sqrt[6]{N} = lg N/k\).)

b. What is the cost of inserting one new key into a “root-6 tree” from problem 1a? Assume that the appropriate node turns out to be at the bottom (all its children are empty, in other words), there is room to insert another key, and no rebalancing is required.

We have to find the insertion point and then insert. With an array implementation of each node, we get \(\Theta(lg N)\) to find and \(\Theta(\sqrt[6]{N})\) to insert. With an ordered linked list, we get \(\Theta(\sqrt[6]{N})\) to find and \(\Theta(1)\) to insert. In either case, the total cost is \(\Theta(\sqrt[6]{N})\).

c. What is the worst-case execution time of the following program fragment, assuming that the function test executes in constant time?

```c
int i, j;
i = 0;
while (i < N) {
    j = 0;
    while (j < i) {
        while (j < i & test (i, j))
            j += 1;
        j += 1;
    }
    i += 1;
}
```

Despite the nesting, \(j\) can only be incremented \(i\) times per iteration of the outer loop, for a total of \(\Theta(N^2)\) time in the worst case.
d. I have a connected, undirected graph with $N$ nodes and $N-1$ edges. Each node has at most two adjacent neighbors. Suggest a better description of this data structure.

   It’s just a (doubly-linked) list.

e. A DAG has $V$ vertices and $E$ edges. I want to list all of the vertices so that each is listed before all vertices that are reachable from it (i.e., reachable by following directed edges). How long will this take by the best algorithm you know? Justify your answer.

   This is simply a topological sort. It requires that we follow each edge once, and also that we visit all vertices, for a total of $\Theta(V + E)$. (The $V$ term is necessary only if the graph is not connected).

f. A hash table contains integer values. I want to determine all values, $x$, in the table such that $L \leq x < U$. How long will this take? Assume this is a “good” hash table in which $N/B < C$, where $N$ is the total number of integers in the table, $B$ is the number of buckets, and $C$ is a constant, and assume that the hash function is ideally chosen for the integers that are stored.

   Search time is constant in such a hash table, but in general, we have to check each of the possible integers in the range, for a total of $\Theta(U - L)$.

g. Given a pseudo-random number generator, how long should it take to select $K$ samples out of a population of $N$ “randomly” without replacement (that is, never choosing the same sample twice). Assume that a sequence of $N$ items already exists, and is presented as a parameter. Justify your answer.

   The random-permutation algorithm given in class selects the last $K$ elements in a permutation with $K$ swaps, so $\Theta(K)$. Since the population already exists, we don’t have to worry about the time to generate the $N-K$ individuals we don’t select.

2. [1 point] Where would one normally expect to find the following literary gem?

   “It wasn’t the best of times; it wasn’t the worst of times; it was the times you’d get if you arranged all possible times (including even fictional times in which the nights were usually dark and stormy) in order from worst to best on the real number line from 0.0 inclusive to 1.0 exclusive and then used a really good uniform random number generator to pick a value in that range thus choosing the corresponding times—that’s the times it was.”

   “Dark and stormy” is a dead giveaway. This is from the 1999 Bulwer-Lytton contest.
3. [7 points] Consider using a trie to store a set of non-negative integers, each represented by its base-2 representation. The children of the root of the tree represent the 1s bit (the left child for 0, the right for 1). The children of each of these nodes represent 2s bits, and so forth. Thus, the path to each node is a binary number presented from least- to most-significant bit. Each node that represents a number that is in the trie (as opposed to representing a substring of the number) will be marked. We mark nodes as close to the root as possible to represent a number: we store 1 as ‘1’, not ‘01’ or ‘001’, and we store 0 by marking the root node. For example, the trie containing 0, 3, 2, and 7 would look like this (here, asterisks denote ‘marks’):

```
   *  
  0   1
   1   1
   *  
  1
   *
```

Fill in the indicated parts of the following class definition.

```java
/** A set of non-negative integers. */
public class IntTrieSet {

    private IntTrieNode root;

    private static class IntTrieNode {
        /** A Trie node whose children are LEFT and RIGHT, and
         * which indicates a number in the trie (as opposed to
         * a mere inner node) if MARK. */
        IntTrieNode (boolean mark, IntTrieNode left, IntTrieNode right) {
            this.mark = mark; this.left = left; this.right = right;
        }

        boolean mark;
        IntTrieNode left, right;
    }

    /** An initially empty set of integers. */
    public IntTrieSet () {
        root = new IntTrieNode (false, null, null);
    }

    Problem continues on the next page.
```
/** True iff I contain X. */
public boolean contains (long x) {
    if (x < 0)
        return false;
    IntTrieNode N;
    N = root;
    while (x != 0 && N != null) {
        if (x & 1 == 1) { /* (or x % 2 == 1) */
            N = N.right;
        } else {
            N = N.left;
            x >>= 1;
        }
    }
    return N != null && N.mark;
}

/** Insert X into me, if it is not already present. Has no effect
* if I already contain X. */
public void add (long x) {
    if (x < 0)
        throw new IllegalArgumentException ("cannot add negatives");
    IntTrieNode N;
    N = root;
    while (x != 0) {
        if (x & 1 == 1) {
            if (N.right == null)
                N.right = new IntTrieNode (false, null, null);
            N = N.right;
        } else {
            if (N.left == null)
                N.left = new IntTrieNode (false, null, null);
            N = N.left;
        }
    }
    N.mark = true;
}
4. [6 points] On the first test, we had a problem involving networks of components. Here we go again. This time, we have networks of components that may include cycles, like this:

![Diagram of network with components Q, R, S, T, and D with directional wires and delays marked with 'D'.]

The wires here are directional, from outputs (on the right of components) to inputs (on the left). The (single) output of a component may fan out to any number of inputs. Each input may be connected to at most one output. Inputs are numbered from 0.

We wish to allow cycles in the diagram only if at least one component along any path from an input back to any input of the same device is a delay (marked ‘D’ in the diagrams). Thus, while there are legal cycles from points Q, R, S, and T back to their respective ‘*’ devices (since these cycles go through the center component), the dashed line would introduce an illegal cycle back to the device connected to T, and so is disallowed. Your task here is to detect illegal cycles. Fill in the following class definitions here and on the next page to fulfill the comments.

```java
package devices;

public class Devices {
    public static Device makeDelay () {
        return new DelayDevice ();
    }
    // Other device generators would go here, but we’re not asking for them.
}
```

Problem continues on next page
public class Device {

    /** The number of inputs I have. */
    public int numInputs () {
        // ASSUME THIS IS IMPLEMENTED SOMEHOW.
    }

    private ArrayList<Device> connections = new ArrayList<Device> ();

    /** Add a connection from my output to input #INPUTNUMBER of DEST, *
     * assuming no other Device connects to it. My output may *
     * connect to any number of inputs. */
    public void connect (Device dest, int inputNumber) {
        connections.add (dest);
        /* For this problem, I can ignore the inputNumber. */
    }

    /** True iff there is a cycle from any of my inputs back to me
     * that does not include a delay. */
    public boolean inBadCycle () {
        HashSet<Device> marked = new HashSet<Device>();
        traverse (marked);
        return marked.contains (this);
    }

    /** Mark all Devices that can be reached from my output without
     * going through a delay. */
    void traverse (HashSet<Device> marked) {
        for (Device d : connections) {
            if (marked.add (d)) /* Returns true if d wasn’t marked. */
                d.traverse (marked);
        }
    }
}

class DelayDevice extends Device {
    void traverse (HashSet<Device> marked) {
    }
}