/* Definition of java.util.Enumeration */
public interface Enumeration {
    /** True if this Enumeration has more elements to produce. */
    public abstract boolean hasMoreElements();
    /** Returns the next element to be produced by this Enumeration,
     * and advances the Enumeration to the following element. */
    public abstract Object nextElement();
}

class List {
    public Object head; /* First element. */
    public List tail; /* Rest of list. */

    public List(Object head, List tail)
    {
        this.head = head; this.tail = tail;
    }
}

Problem #1 [6 points]

For each of the following, either indicate that it is true and briefly tell why or give a counter-example.

a. A binary tree of height $h$ contains $O(2^{(2h)})$ nodes.

b. A binary tree containing $N$ nodes has height $\Omega(\log N)$.

c. A complete binary tree of height $h$ contains $\Theta(2^{(h-3)})$ nodes.

d. If I add $K$ items to the beginning of a list of $N$ items, and if those $K$ new items are all smaller than the remaining $N$, then the number of inversions in the list increases by $O(K^2)$.

e. A $\Theta(N)$ algorithm is always preferable to a $\Theta(N^2)$ algorithm.

f. A certain hash table contains $N$ integer keys, all distinct, and each of its bins contains at most $K$ elements. Assuming that the hashing function and the equality test require constant time, the time required to find all keys in the hash table that are between $L$ and $U$ is $O(K^*(U-L))$ in the worst case.
Problem #2 [1 point]

If \(a\) and \(m\) are relatively prime, and \(\phi(m)\) denotes the number of positive integers less than and relatively prime to \(m\), what is \(a^{\phi(m)} \mod m\)?

Problem #3 [6 points]

You are given a set of ordered pairs of real numbers, \((A_i, B_i)\), satisfying \(0 \leq A_i \leq B_i \leq 1\) for \(0 \leq i < N\). Consider each pair as representing an interval on the real line:

\((A_i, B_i)\) represents \([A_i..B_i] = \{x | A_i \leq x \leq B_i\}\)

As quickly as possible, I would like to find and print the endpoints of all intervals between 0 and 1 of numbers that are not contained in any of these intervals. For example, if the inputs are

\[(0.1, 0.25), (0.0, 0.2), (0.8, 0.9), (0.4, 0.6), (0.35, 0.65)\]

the output should be

\[(0.25, 0.35), (0.65, 0.8), (0.9, 1.0)\]

in any order. For the purposes of this problem, we'll ignore the distinction between closed intervals--those that include their endpoints--and open intervals--those that do not. Present the fastest algorithm you can think of to do this. Keep things at a reasonably high level. You may assume that you have available to you implementations of any of the data structures and algorithms we've discussed, without going into details of how they are implemented. You may use pseudo-code--that is, English sentences such as "store the \(A_i\) in has table \(H\)..." or "heapify the array of \(B_i\)'s" without going into further detail.

Problem #4 [7 points]

The data type \(KTree\) is a positional tree in which all labels are at the leaves. It has the following (partial) definition:

```java
class KTree {
    /** The number of non-empty children of this node. */
    public int degree() ...
    /** Child number \(K\) of this (null indicates an empty child). */
    public KTree child(int k) ...
    /** The label of this node. Valid iff degree() == 0. */
    public Object label() ...
    /** An enumeration of all labels in this tree, in any order. */
    public Enumeration allLabels() ...
}
```

Assume that \(degree\), \(child\), and \(label\) are already written. You are to supply a definition of \(allLabels\). For example, if \(T\) is a \(KTree\), I want to be able to write

```java
Enumeration p;
p = T.allLabels();
while (p.hasMoreElements())
    System.out.println(p.nextElement().toString());
```
and have (the string representations of) all labels at the leaves of T printed. [For your reference, there is a definition of Enumeration on page 2.] Fill in allLabels to make this happen, adding any additional classes and methods you want.

/** An enumeration of all labels in this tree, in any order. */
public Enumeration allLabels() {
    /* FILL IN */
}

Problem #5 [6 points]

Assume that the type List is defined as on page 2. The procedure mergeSet, described below, is supposed to merge an arbitrary number (M) of sorted lists, as opposed to a simple pair of lists. It is possible to do this in O(NlgM), where N is the total number of elements in all lists. It does not suffice to concatenate all the lists together and then sort them, since that will require time theta(NlgN) in the worst case (we assume that we can only compare items on the list, so that radix sorts and distribution sorts are out).

a. The following also does not meet the desired time constraint:

    static List mergeSet (List[] L)
    {
        List result = null;
        for (int i = 0; i < L.length; i += 1)
            result = merge2(result, L[i]);
        return result;
    }
(where \texttt{merge2} is the familiar procedure for merging two lists in time proportional to the total number of items in the two lists). What is the worst-case time bound on this implementation and why?

b. Fill in the body below to fulfill the comment. Keep your solution at as high a level as possible. You may assume that doubly-linked lists, queues, deques, stacks, hash tables, balanced binary search trees, and priority queues are provided, if any of these are useful to you; again, just state your assumptions. Again, your program must run in \(O(N\lg M)\) time. Assume that the operation \(\lt\) compares two items in the lists for /** Assumes that lists, \(L[0],\ldots\) are sorted. Returns the result of * merging the lists \(L[0],\ldots,L[M-1]\) into a single sorted list. * The operation may be destructive; it may destroy the original lists. */ static List mergeSet(List[] L) { // FILL IN } */