Set Interface API

```
public interface Set<E> extends Collection<E> {
    /** Adds the specified element to this set if it is not already present. */
    public boolean add(E e) { ... }

    /** Returns true if this set contains the specified element. */
    public boolean contains(E e) { ... }

    /** Returns true if this set contains no elements. */
    public boolean isEmpty() { ... }

    /** Removes the specified element from this set if it is present. */
    public boolean remove(E e) { ... }

    /** Returns the number of elements in this set. */
    public int size() { ... }
}
```

WeightedQuickUnion Class API

```
public class WeightedQuickUnion {
    /** Creates a WeightedQuickUnion with n elements numbered 0 to n - 1 (inclusive). */
    public WeightedQuickUnion(int n) { ... }

    /** Checks whether x and y belong to the same set. */
    public boolean isConnected(int x, int y) { ... }

    /** Unions the set that x belongs to and the set that y belongs to. */
    public void union(int x, int y) { ... }
}
```

Math Class API

```
public class Math {
    /** Returns the value of the first argument raised to the power of the second argument.
    Runs in Θ(1) time. */
    public static double pow(double a, double b) { ... }

    /** Returns the positive remainder when dividing x by y. Runs in Θ(1) time. */
    public static long floorMod(long x, long y) { ... }

    /** Returns the positive square root of a double value.
    Runs in Θ(1) time. */
    public static double sqrt(double a) { ... }
}
```

Map Interface API

```
public interface Map<K,V> {
    /** Associates the specified value with the specified key in this map. */
    public V put(K key, V value) { ... }

    /** Returns true if this map contains a mapping for the specified key. */
    public boolean containsKey(K key) { ... }

    /** Returns true if this map contains no key-value mappings. */
    public boolean isEmpty() { ... }

    /** Removes the mapping for a key from this map if it is present. */
    public V remove(K key) { ... }

    /** Returns the number of key-value mappings in this map. */
    public int size() { ... }

    /** Returns the value to which the specified key is mapped, or null if this map contains no
    mapping for the key. */
    public V get(K key) { ... }
}
```

Random Class API

```
public class Random {
    /** Creates a new random number generator.  Runs in Θ(1) time. */
    public Random() { ... }

    /** Creates a new random number generator using a seed. If two instances of Random are created
    with the same seed, they will generate and return identical sequences of numbers. Runs in Θ(1)
    time. */
    public Random(long seed) { ... }

    /** Returns the next pseudorandom int from this random number generator's sequence. Runs in Θ(1)
    time. */
    public int nextInt() { ... }

    /** Returns a pseudorandom int value between 0 (inclusive) and the specified bound (exclusive),
    drawn from this random number generator's sequence. Runs in Θ(1) time. */
    public int nextInt(int bound) { ... }

    /** Returns a pseudorandom int value between origin (inclusive) and the specified bound (
    exclusive), drawn from this random number generator's sequence. Runs in Θ(1) time. */
    public int nextInt(int origin, int bound) { ... }
}
```

PriorityQueue Class API

```
public class PriorityQueue<E extends Comparable<E>> {
    /** Creates a PriorityQueue<E> with elements ordered according to their natural
    ordering as defined by the compareTo method of the elements. Runs in Θ(1) time. */
    public PriorityQueue() { ... }

    /** Inserts the specified element into this priority queue. Runs in O(log(N)) time. */
    public void add(E e) { ... }

    /** Retrieves and removes the head of this queue, or returns null if this queue is empty. The
    head of this queue is the element which is smallest according to the compareTo method.
    Runs in O(log(N)) time. */
    public E poll() { ... }

    /** Determines whether the queue is empty or not. Runs in Θ(1) time. */
    public boolean isEmpty() { ... }

    /** Returns the size of the queue. Runs in Θ(1) time. */
    public int size() { ... }

    /** Retrieves, but does not remove, the head of this queue, or returns null if this queue is
    empty. Runs in Θ(1) time. */
    public E peek() { ... }
}
```

List Interface API

```
public interface List<E> {
    /** Appends the specified element to the end of this list. Runs in constant time. */
    public void add(E e);

    /** Returns the element at the specified position in this list. */
    public E get(int index);

    /** Replaces the element at the specified position in this list with the specified element. */
    public E set(int index, E element);

    /** Returns the number of elements in this list. */
    public int size();

    /** Returns true if this list contains no elements. */
    public boolean isEmpty();

    /** Lists are defined to be equal if they contain the same elements in the same order. */
    public boolean equals(Object o);
}
```

This is a piece of scratch paper.