Set Interface API

```
public interface Set<E> {
    /** Adds the specified element to this set if it is not already present. */
    public void 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 and returns the specified element from this set if it is present. */
    public E 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 void 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> {
    /** Constructor that takes in a List and creates a new List copy with the same elements. */
    public List<E> (List<E> 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 void set(int index, E element);

    /** Removes and returns the element at the specified position in this list. */
    public E remove(int index);

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

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

    /** Returns a view of the portion of this list between the specified fromIndex, inclusive, and
    toIndex, exclusive. */
    public List<E> subList(int fromIndex, int toIndex);

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

    /** Returns true if this list contains the specified element. */
    public boolean contains(E element);

    /** Returns an immutable list containing an arbitrary number of elements. For example,
    List<Integer> example = List.of(1, 2, 3); is a valid assignment. */
    public static <E> List<E> of(E... elements);
}
// Implementations of Lists covered in class
public class ArrayList<E> implements List<E> {...}
public class LinkedList<E> implements List<E> {...}
```

Deque Interface API

```
public interface Deque<E> {
    /** Inserts the specified element at the front of this deque. */
    public void addFirst(E e);

    /** Inserts the specified element at the end of this deque. */
    public void addLast(E e);

    /** Retrieves, but does not remove, the first element of this deque. */
    public E getFirst();

    /** Retrieves, but does not remove, the last element of this deque. */
    public E getLast();

    /** Retrieves and removes the first element of this deque. */
    public E removeFirst();

    /** Retrieves and removes the last element of this deque. */
    public E removeLast();

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

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

```
// Implementations of Deques covered in class
public class ArrayDeque<E> implements Deque<E> {...}
public class LinkedListDeque<E> implements Deque<E> {...}
```

Integer Class API

```
public class Integer {
    /** A constant holding the minimum value an int can have, -2^31. */
    public static final int MIN_VALUE = -2147483648;

    /** A constant holding the maximum value an int can have, 2^31-1. */
    public static final int MAX_VALUE = 2147483647;
}
```

String Class API

```
public class String {
    /** Returns the char value at the specified index. */
    public char charAt(int index) { ... }

    /** Returns the length of this string. */
    public int length() { ... }
}
```

This is a piece of scratch paper.