```
CS 6301.002.  Implementation of advanced data structures and algorithms
Spring 2016
Short Project 3
Tue, Feb 16, 2016

Ver 1.0: Initial description (Feb 16, 9:00 AM).
Ver 1.1: Added the statement "Solve at least one ...", p = 999953 (Fibonacci).
Elegant solutions to part f (Generic graph class) are eligible for 0.2 EC.

Due: 1:00 PM, Thu, Feb 25.
```

Solve at least one problem from the following list. First solution will be graded out of 10. Each additional problem will be considered for 1 extra point.

## a. Fibonacci numbers

Compare the running times of the O(n) and O(log n) algorithms for computing f_n % p, where f_n is the nth Fibonacci number. Write the following functions and compare their running times for large values of n. Use p = 999953.

```
public static long linearFibonacci(long n, long p);
public static long logFibonacci(long n, long p);
```

## b. Multipivot Quicksort

Implement generic versions of Quick sort using traditional partition and compare its performance with Quick sort that uses multipivot partition (2 and/or 3 pivots). What happens when the input array elements are not distinct?

## c. Merge sort vs Quick sort

Implement the best versions of Merge sort and Quick sort based on discussions in class (generic versions), and compare their performance. For convenience, use only sizes that are even powers of 2 (e.g., 1048576, 4194304, 16777216, 67108864). What happens when the input array elements are not distinct?

## d. Selection problem

Implement generics version of the selection problem (k largest elements of an unsorted array). Compare the performance of the O(n) algorithm with the heap-based algorithm discussed in class for the external version of the problem.

```
public static<T extends Comparable<? super T>> int select(T[] arr, int p, int r, int k)
// Find the k largest elements of arr[p..r].  Returns index q.
// The k largest elements are found in arr[q..r].
```

## e. kth quantiles problem

Implement an O(n log k) algorithm for the kth quantile problem (Ex. 9.3-6, p. 223 in Cormen et al's book, 3rd ed.).

```
public static<T extends Comparable<? super T>> void kthQuantiles(T[] A, T[] result, int k)
// The kth quantiles are returned in the array "result", of size k-1, supplied by the caller.
```

## f. Generic graph class

This problem is for training in OO methodology. The classes we have created for Graph, Vertex, and Edge are useful, but they are unsatisfactory for the following reason. For each application, we have had to modify the classes and add additional fields. Ideally, we should just extend the Vertex and Edge classes, as needed, for each problem. The question is how.

Write a generic version of the Graph class. Header:

```
public class Graph<V extends Vertex,E extends Edge> implements Iterable<V> {
```

Implement the vertices using an array (not array list). Rewrite the readGraph method so that it can be used to read input and construct a graph.

```
public static<V extends Vertex,E extends Edge> Graph<V,E> readGraph(Scanner in, boolean directed)
```

Add additional parameters to readGraph, as needed. We should be able to call the method as:

```
Graph<EulerVertex,EulerEdge> g = Graph.readGraph(in, false);
```

This will create an undirected graph, whose nodes are of type EulerVertex and edges are of type EulerEdge. Those classes are defined as:

```
class EulerVertex extends Vertex { ... }
class EulerEdge extends Edge { ... }
```