See the Assessment Guide for information on how to interpret this report.

ASSESSMENT SUMMARY

```
Compilation:  PASSED (0 errors, 3 warnings)
API:          PASSED

SpotBugs:     FAILED (11 warnings)
PMD:          FAILED (13 warnings)
Checkstyle:   FAILED (0 errors, 9 warnings)

Correctness:  46/49 tests passed
Memory:       125/124 tests passed
Timing:       193/193 tests passed

Aggregate score: 96.41%
[ Compilation: 5%, API: 5%, Style: 0%, Correctness: 60%, Timing: 10%, Memory: 20% ]
```

ASSESSMENT DETAILS

```
The following files were submitted:
----------------------------------
5.3K Mar  7 13:00 Deque.java
 754 Mar  7 13:00 Permutation.java
5.6K Mar  7 13:00 RandomizedQueue.java


================================================================================
*  COMPILING
================================================================================


% javac Deque.java
*-----------------------------------------------------------

% javac RandomizedQueue.java
*-----------------------------------------------------------
RandomizedQueue.java:16: warning: [unchecked] unchecked cast
        elements = (Item[]) new Object[1];
                            ^
  required: Item[]
  found:    Object[]
  where Item is a type-variable:
    Item extends Object declared in class RandomizedQueue
RandomizedQueue.java:32: warning: [unchecked] unchecked cast
        Item[] copy = (Item[]) new Object[capacity];
                               ^
  required: Item[]
  found:    Object[]
  where Item is a type-variable:
    Item extends Object declared in class RandomizedQueue
RandomizedQueue.java:90: warning: [unchecked] unchecked cast
            copy = (Item[]) new Object[elements.length];
                            ^
  required: Item[]
  found:    Object[]
  where Item is a type-variable:
    Item extends Object declared in class RandomizedQueue
3 warnings

% javac Permutation.java
*-----------------------------------------------------------


================================================================


Checking the APIs of your programs.
*-----------------------------------------------------------
Deque:

RandomizedQueue:

Permutation:

================================================================


================================================================================
*  CHECKING STYLE AND COMMON BUG PATTERNS
================================================================================


% spotbugs *.class
*-----------------------------------------------------
M D IMC_IMMATURE_CLASS_PRINTSTACKTRACE IMC: Method Deque.main(String[]) prints the stack trace to the console  At Deque.java:[line 148]
M D IMC_IMMATURE_CLASS_PRINTSTACKTRACE IMC: Method Deque.main(String[]) prints the stack trace to the console  At Deque.java:[line 155]
M D IMC_IMMATURE_CLASS_PRINTSTACKTRACE IMC: Method Deque.main(String[]) prints the stack trace to the console  At Deque.java:[line 162]
M D IMC_IMMATURE_CLASS_PRINTSTACKTRACE IMC: Method Deque.main(String[]) prints the stack trace to the console  At Deque.java:[line 169]
M D IMC_IMMATURE_CLASS_PRINTSTACKTRACE IMC: Method Deque.main(String[]) prints the stack trace to the console  At Deque.java:[line 177]
M D IMC_IMMATURE_CLASS_PRINTSTACKTRACE IMC: Method Deque.main(String[]) prints the stack trace to the console  At Deque.java:[line 184]
M D IMC_IMMATURE_CLASS_PRINTSTACKTRACE IMC: Method RandomizedQueue.main(String[]) prints the stack trace to the console  At RandomizedQueue.java:[line 1
```

```
M D IMC_IMMATURE_CLASS_PRINTSTACKTRACE IMC: Method RandomizedQueue.main(String[]) prints the stack trace to the console  At RandomizedQueue.java:[line 14
M D IMC_IMMATURE_CLASS_PRINTSTACKTRACE IMC: Method RandomizedQueue.main(String[]) prints the stack trace to the console  At RandomizedQueue.java:[line 15
M D IMC_IMMATURE_CLASS_PRINTSTACKTRACE IMC: Method RandomizedQueue.main(String[]) prints the stack trace to the console  At RandomizedQueue.java:[line 16
M D IMC_IMMATURE_CLASS_PRINTSTACKTRACE IMC: Method RandomizedQueue.main(String[]) prints the stack trace to the console  At RandomizedQueue.java:[line 16
SpotBugs ends with 11 warning.


=============================================================


% pmd .
*-----------------------------------------------------------
Deque.java:11: The private instance (or static) variable 'element' can be made 'final'; it is initialized only in the declaration or constructor. [Immuta
Deque.java:148: Avoid printStackTrace(); use a logger call instead. [AvoidPrintStackTrace]
Deque.java:155: Avoid printStackTrace(); use a logger call instead. [AvoidPrintStackTrace]
Deque.java:162: Avoid printStackTrace(); use a logger call instead. [AvoidPrintStackTrace]
Deque.java:169: Avoid printStackTrace(); use a logger call instead. [AvoidPrintStackTrace]
Deque.java:177: Avoid printStackTrace(); use a logger call instead. [AvoidPrintStackTrace]
Deque.java:184: Avoid printStackTrace(); use a logger call instead. [AvoidPrintStackTrace]
RandomizedQueue.java:86: The private instance (or static) variable 'copy' can be made 'final'; it is initialized only in the declaration or constructor.
RandomizedQueue.java:139: Avoid printStackTrace(); use a logger call instead. [AvoidPrintStackTrace]
RandomizedQueue.java:146: Avoid printStackTrace(); use a logger call instead. [AvoidPrintStackTrace]
RandomizedQueue.java:153: Avoid printStackTrace(); use a logger call instead. [AvoidPrintStackTrace]
RandomizedQueue.java:161: Avoid printStackTrace(); use a logger call instead. [AvoidPrintStackTrace]
RandomizedQueue.java:168: Avoid printStackTrace(); use a logger call instead. [AvoidPrintStackTrace]
PMD ends with 13 warnings.


=============================================================


% checkstyle *.java
*-----------------------------------------------------------
[WARN] Deque.java:24:17: The instance variable 'N' must start with a lowercase letter and use camelCase. [MemberName]
[WARN] Deque.java:134:29: '{' is not preceded with whitespace. [WhitespaceAround]
[WARN] Permutation.java:14:36: '{' is not preceded with whitespace. [WhitespaceAround]
[WARN] RandomizedQueue.java:12:17: The instance variable 'N' must start with a lowercase letter and use camelCase. [MemberName]
[WARN] RandomizedQueue.java:127:29: '{' is not preceded with whitespace. [WhitespaceAround]
[WARN] RandomizedQueue.java:190:9: 'while' is not followed by whitespace. [WhitespaceAround]
[WARN] RandomizedQueue.java:190:30: '{' is not preceded with whitespace. [WhitespaceAround]
[WARN] RandomizedQueue.java:196:9: 'while' is not followed by whitespace. [WhitespaceAround]
[WARN] RandomizedQueue.java:196:31: '{' is not preceded with whitespace. [WhitespaceAround]
Checkstyle ends with 0 errors and 9 warnings.

% custom checkstyle checks for Deque.java
*-----------------------------------------------------------

% custom checkstyle checks for RandomizedQueue.java
*-----------------------------------------------------------

% custom checkstyle checks for Permutation.java
*-----------------------------------------------------------


=============================================================


********************************************************************************
*  TESTING CORRECTNESS
********************************************************************************

Testing correctness of Deque
*-----------------------------------------------------------
Running 19 total tests.

Tests 1-8 make random intermixed calls to addFirst(), addLast(),
removeFirst(), removeLast(), isEmpty(), and size(), and iterator().
The probabilities of each operation are (p1, p2, p3, p4, p5, p6, p7),
respectively.

Test 1: check random calls to addFirst(), addLast(), and size()
  *    5 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2, 0.0)
  *   50 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2, 0.0)
  *  500 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2, 0.0)
  * 1000 random calls (0.4, 0.4, 0.0, 0.0, 0.0, 0.2, 0.0)
==> passed

Test 2: check random calls to addFirst(), removeFirst(), and isEmpty()
  *    5 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0, 0.0)
  *   50 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0, 0.0)
  *  500 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0, 0.0)
  * 1000 random calls (0.8, 0.0, 0.1, 0.0, 0.1, 0.0, 0.0)
  *    5 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0, 0.0)
  *   50 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0, 0.0)
  *  500 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0, 0.0)
  * 1000 random calls (0.1, 0.0, 0.8, 0.0, 0.1, 0.0, 0.0)
==> passed

Test 3: check random calls to addFirst(), removeLast(), and isEmpty()
  *    5 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0, 0.0)
  *   50 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0, 0.0)
  *  500 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0, 0.0)
  * 1000 random calls (0.8, 0.0, 0.0, 0.1, 0.1, 0.0, 0.0)
  *    5 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0, 0.0)
  *   50 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0, 0.0)
  *  500 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0, 0.0)
  * 1000 random calls (0.1, 0.0, 0.0, 0.8, 0.1, 0.0, 0.0)
==> passed

Test 4: check random calls to addLast(), removeLast(), and isEmpty()
  *    5 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0, 0.0)
  *   50 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0, 0.0)
  *  500 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0, 0.0)
  * 1000 random calls (0.0, 0.8, 0.0, 0.1, 0.1, 0.0, 0.0)
```

```
  *    5 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0, 0.0)
  *   50 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0, 0.0)
  *  500 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0, 0.0)
  * 1000 random calls (0.0, 0.1, 0.0, 0.8, 0.1, 0.0, 0.0)
==> passed

Test 5: check random calls to addLast(), removeFirst(), and isEmpty()
  *    5 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0, 0.0)
  *   50 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0, 0.0)
  *  500 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0, 0.0)
  * 1000 random calls (0.0, 0.8, 0.1, 0.0, 0.1, 0.0, 0.0)
  *    5 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0, 0.0)
  *   50 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0, 0.0)
  *  500 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0, 0.0)
  * 1000 random calls (0.0, 0.1, 0.8, 0.0, 0.1, 0.0, 0.0)
==> passed

Test 6: check random calls to addFirst(), removeFirst(), and iterator()
  *    5 random calls (0.8, 0.0, 0.1, 0.0, 0.0, 0.0, 0.1)
  *   50 random calls (0.8, 0.0, 0.1, 0.0, 0.0, 0.0, 0.1)
  *  500 random calls (0.8, 0.0, 0.1, 0.0, 0.0, 0.0, 0.1)
  * 1000 random calls (0.8, 0.0, 0.1, 0.0, 0.0, 0.0, 0.1)
  *    5 random calls (0.1, 0.0, 0.8, 0.0, 0.0, 0.0, 0.1)
  *   50 random calls (0.1, 0.0, 0.8, 0.0, 0.0, 0.0, 0.1)
  *  500 random calls (0.1, 0.0, 0.8, 0.0, 0.0, 0.0, 0.1)
  * 1000 random calls (0.1, 0.0, 0.8, 0.0, 0.0, 0.0, 0.1)
==> passed

Test 7: check random calls to all methods except iterator()
  *    5 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1, 0.0)
  *   50 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1, 0.0)
  *  500 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1, 0.0)
  * 1000 random calls (0.3, 0.3, 0.1, 0.1, 0.1, 0.1, 0.0)
  *    5 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1, 0.0)
  *   50 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1, 0.0)
  *  500 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1, 0.0)
  * 1000 random calls (0.1, 0.1, 0.3, 0.3, 0.1, 0.1, 0.0)
==> passed

Test 8: check random calls to all methods, including iterator()
  *    5 random calls (0.2, 0.2, 0.1, 0.1, 0.1, 0.1, 0.2)
  *   50 random calls (0.2, 0.2, 0.1, 0.1, 0.1, 0.1, 0.2)
  *  500 random calls (0.2, 0.2, 0.1, 0.1, 0.1, 0.1, 0.2)
  * 1000 random calls (0.2, 0.2, 0.1, 0.1, 0.1, 0.1, 0.2)
  *    5 random calls (0.1, 0.1, 0.2, 0.2, 0.1, 0.1, 0.2)
  *   50 random calls (0.1, 0.1, 0.2, 0.2, 0.1, 0.1, 0.2)
  *  500 random calls (0.1, 0.1, 0.2, 0.2, 0.1, 0.1, 0.2)
  * 1000 random calls (0.1, 0.1, 0.2, 0.2, 0.1, 0.1, 0.2)
==> passed

Test 9: check removeFirst() and removeLast() from an empty deque
  * removeFirst()
  * removeLast()
==> passed

Test 10: check whether two Deque objects can be created at the same time
  * n = 10
  * n = 1000
==> passed

Test 11: check iterator() after n calls to addFirst()
  * n = 10
  * n = 50
==> passed

Test 12: check iterator() after random calls to addFirst(), addLast(),
         removeFirst(), and removeLast() with probabilities (p1, p2, p3, p4)
  * 20 random operations (0.8, 0.0, 0.2, 0.0)
  * 20 random operations (0.8, 0.0, 0.0, 0.2)
  * 20 random operations (0.0, 0.8, 0.0, 0.2)
  * 20 random operations (0.0, 0.8, 0.2, 0.0)
  * 20 random operations (0.4, 0.4, 0.1, 0.1)
  * 20 random operations (0.2, 0.0, 0.8, 0.0)
  * 20 random operations (0.2, 0.0, 0.0, 0.8)
  * 20 random operations (0.0, 0.2, 0.0, 0.8)
  * 20 random operations (0.0, 0.2, 0.8, 0.0)
  * 20 random operations (0.1, 0.1, 0.4, 0.4)
  * 100 random operations (0.4, 0.4, 0.1, 0.1)
  * 1000 random operations (0.4, 0.4, 0.1, 0.1)
==> passed

Test 13: create two nested iterators to same deque of size n
  * n = 10
  * n = 50
==> passed

Test 14: create two parallel iterators to same deque of size n
  * n = 10
  * n = 50
==> passed

Test 15: create an iterator and check calls to next() and hasNext()
  * 10 consecutive calls to hasNext() on a deque of size 10
  * 10 consecutive calls to next() on a deque of size 10
  * 50 random intermixed calls to next() and hasNext() on a deque of size 10
  * 1000 random intermixed calls to next() and hasNext() on a deque of size 100
==> passed

Test 16: create Deque objects of different parameterized types
==> passed

Test 17: call addFirst() and addLast() with null argument
==> passed
```

```
Test 18: check that remove() and next() throw the specified exceptions in iterator()
==> passed

Test 19: call iterator() when the deque is empty
==> passed


Total: 19/19 tests passed!


================================================================
Testing correctness of RandomizedQueue
*----------------------------------------------------------------
Running 21 total tests.

Tests 1-6 make random intermixed calls to enqueue(), dequeue(), sample(),
isEmpty(), size(), and iterator(). The probabilities of each operation
are (p1, p2, p3, p4, p5, p6), respectively.

Test 1: check random calls to enqueue() and size()
  *    5 random calls (0.8, 0.0, 0.0, 0.0, 0.2, 0.0)
  *   50 random calls (0.8, 0.0, 0.0, 0.0, 0.2, 0.0)
  *  500 random calls (0.8, 0.0, 0.0, 0.0, 0.2, 0.0)
  * 1000 random calls (0.8, 0.0, 0.0, 0.0, 0.2, 0.0)
==> passed

Test 2: check random calls to enqueue() and dequeue()
  *    5 random calls (0.7, 0.1, 0.0, 0.1, 0.1, 0.0)
  *   50 random calls (0.7, 0.1, 0.0, 0.1, 0.1, 0.0)
  *  500 random calls (0.7, 0.1, 0.0, 0.1, 0.1, 0.0)
  * 1000 random calls (0.7, 0.1, 0.0, 0.1, 0.1, 0.0)
  *    5 random calls (0.1, 0.7, 0.0, 0.1, 0.1, 0.0)
  *   50 random calls (0.1, 0.7, 0.0, 0.1, 0.1, 0.0)
  *  500 random calls (0.1, 0.7, 0.0, 0.1, 0.1, 0.0)
  * 1000 random calls (0.1, 0.7, 0.0, 0.1, 0.1, 0.0)
==> passed

Test 3: check random calls to enqueue() and sample()
  *    5 random calls (0.8, 0.0, 0.2, 0.0, 0.0, 0.0)
  *   50 random calls (0.8, 0.0, 0.2, 0.0, 0.0, 0.0)
  *  500 random calls (0.8, 0.0, 0.2, 0.0, 0.0, 0.0)
  * 1000 random calls (0.8, 0.0, 0.2, 0.0, 0.0, 0.0)
  *    5 random calls (0.2, 0.0, 0.8, 0.0, 0.0, 0.0)
  *   50 random calls (0.2, 0.0, 0.8, 0.0, 0.0, 0.0)
  *  500 random calls (0.2, 0.0, 0.8, 0.0, 0.0, 0.0)
  * 1000 random calls (0.2, 0.0, 0.8, 0.0, 0.0, 0.0)
==> passed

Test 4: check random calls to enqueue() and iterator()
  *    5 random calls (0.8, 0.0, 0.0, 0.0, 0.0, 0.2)
  *   50 random calls (0.8, 0.0, 0.0, 0.0, 0.0, 0.2)
  *  500 random calls (0.8, 0.0, 0.0, 0.0, 0.0, 0.2)
  * 1000 random calls (0.8, 0.0, 0.0, 0.0, 0.0, 0.2)
==> passed

Test 5: check random calls to all methods except iterator()
  *    5 random calls (0.6, 0.1, 0.1, 0.1, 0.1, 0.0)
  *   50 random calls (0.6, 0.1, 0.1, 0.1, 0.1, 0.0)
  *  500 random calls (0.6, 0.1, 0.1, 0.1, 0.1, 0.0)
  * 1000 random calls (0.6, 0.1, 0.1, 0.1, 0.1, 0.0)
  *    5 random calls (0.1, 0.6, 0.1, 0.1, 0.1, 0.0)
  *   50 random calls (0.1, 0.6, 0.1, 0.1, 0.1, 0.0)
  *  500 random calls (0.1, 0.6, 0.1, 0.1, 0.1, 0.0)
  * 1000 random calls (0.1, 0.6, 0.1, 0.1, 0.1, 0.0)
==> passed

Test 6: check random calls to all methods, including iterator()
  *    5 random calls (0.5, 0.1, 0.1, 0.1, 0.1, 0.1)
  *   50 random calls (0.5, 0.1, 0.1, 0.1, 0.1, 0.1)
  *  500 random calls (0.5, 0.1, 0.1, 0.1, 0.1, 0.1)
  * 1000 random calls (0.5, 0.1, 0.1, 0.1, 0.1, 0.1)
  *    5 random calls (0.1, 0.5, 0.1, 0.1, 0.1, 0.1)
  *   50 random calls (0.1, 0.5, 0.1, 0.1, 0.1, 0.1)
  *  500 random calls (0.1, 0.5, 0.1, 0.1, 0.1, 0.1)
  * 1000 random calls (0.1, 0.5, 0.1, 0.1, 0.1, 0.1)
==> passed

Test 7: call dequeue() and sample() from an empty randomized queue
  * dequeue()
  * sample()
==> passed

Test 8: create multiple randomized queue objects at the same time
  * n = 10
  * n = 100
==> passed

Test 9: check that iterator() returns correct items after a sequence
        of n enqueue() operations
  * n = 10
  * n = 50
==> passed

Test 10: check that iterator() returns correct items after intermixed
         sequence of m enqueue() and dequeue() operations
  * m = 10
  * m = 1000
==> passed

Test 11: create two nested iterators over the same randomized queue of size n
  * n = 10
  * n = 50
==> passed
```

```
Test 12: create two parallel iterators over the same randomized queue of size n
  * n = 10
  * n = 50
==> passed

Test 13: create two iterators over different randomized queues,
         each of length 10
==> passed

Test 14: create an iterator and check calls to next() and hasNext()
  * 10 consecutive calls to hasNext() on a deque of size 10
  * 10 consecutive calls to next() on a deque of size 10
  * 50 random intermixed calls to next() and hasNext() on a deque of size 10
  * 1000 random intermixed calls to next() and hasNext() on a deque of size 100
==> passed

Test 15: create RandomizedQueue objects of different parameterized types
==> passed

Test 16: check randomness of sample() by enqueueing n items, repeatedly calling
         sample(), and counting the frequency of each item
  * n = 3, trials = 12000
  * n = 5, trials = 12000
  * n = 8, trials = 12000
  * n = 10, trials = 12000
==> passed

Test 17: check randomness of dequeue() by enqueueing n items, dequeueing n items,
         and seeing whether each of the n! permutations is equally likely
  * n = 2, trials = 12000
  * n = 3, trials = 12000
  * n = 4, trials = 12000
  * n = 5, trials = 12000
==> passed

Test 18: check randomness of iterator() by enqueueing n items, iterating over those
         n items, and seeing whether each of the n! permutations is equally likely
  * n = 2, trials = 12000
  * n = 3, trials = 12000
  * n = 4, trials = 12000
  * n = 5, trials = 12000
==> passed

Test 19: call enqueue() with a null argument
==> passed

Test 20: check that remove() and next() throw the specified exceptions in iterator()
==> passed

Test 21: call iterator() when randomized queue is empty
==> passed


Total: 21/21 tests passed!


================================================================
********************************************************************************
*  TESTING CORRECTNESS (substituting reference RandomizedQueue and Deque)
********************************************************************************

Testing correctness of Permutation
*-----------------------------------------------------------
Tests 1-5 call the main() function directly, resetting standard input
before each call.

Running 9 total tests.

Test 1a: check formatting for sample inputs from assignment specification
  % java Permutation 3 < distinct.txt
  B
  C
  A

  % java Permutation 3 < distinct.txt
  B
  A
  C

  % java Permutation 8 < duplicates.txt
  BB
  CC
  BB
  BB
  AA
  BB
  CC
  BB

==> passed

Test 1b: check formatting for other inputs
  % java Permutation 8 < mediumTale.txt
  was
  times
  it
  was
  of
  the
  it
  best

  % java Permutation 0 < distinct.txt
  [no output]
```

```
==> passed

Test 2: check that main() reads all data from standard input
  * filename = distinct.txt, k = 3
    - student solution does not read all data from standard input
    - number of tokens read          = 3
    - number of tokens left unread   = 6
    - number of tokens in input file = 9

  * filename = distinct.txt, k = 3
    - student solution does not read all data from standard input
    - number of tokens read          = 3
    - number of tokens left unread   = 6
    - number of tokens in input file = 9

  * filename = duplicates.txt, k = 8
  * filename = mediumTale.txt, k = 8
    - student solution does not read all data from standard input
    - number of tokens read          = 8
    - number of tokens left unread   = 16
    - number of tokens in input file = 24

==> FAILED

Test 3a: check that main() prints each item from the sequence at most once
         (for inputs with no duplicate strings)
  * filename = distinct.txt, k = 3
  * filename = distinct.txt, k = 1
  * filename = distinct.txt, k = 9
  * filename = permutation6.txt, k = 6
  * filename = permutation10.txt, k = 10
==> passed

Test 3b: check that main() prints each item from the sequence at most once
         (for inputs with duplicate strings)
  * filename = duplicates.txt, k = 8
  * filename = duplicates.txt, k = 3
  * filename = permutation8.txt, k = 6
  * filename = permutation8.txt, k = 2
  * filename = tinyTale.txt, k = 10
==> passed

Test 3c: check that main() prints each item from the sequence at most once
         (for inputs with newlines)
  * filename = mediumTale.txt, k = 10
  * filename = mediumTale.txt, k = 20
  * filename = tale.txt, k = 10
  * filename = tale.txt, k = 50
==> passed

Test 4: check main() when k = 0
  * filename = distinct.txt, k = 0
  * filename = distinct.txt, k = 0
==> passed

Test 5a: check that permutations are uniformly random
         (for inputs with no duplicate strings)
  * filename = permutation4.txt, k = 1

           value  observed  expected   2*0*ln(0/E)
       ------------------------------------------
               A      1000     250.0       2772.59
               B         0     250.0          0.00
               C         0     250.0          0.00
               D         0     250.0          0.00
       ------------------------------------------
                      1000    1000.0       2772.59

    G-statistic = 2772.59 (p-value = 0.000000, reject if p-value <= 0.0001)
    Note: a correct solution will fail this test by bad luck 1 time in 10,000.

  * filename = permutation4.txt, k = 2

           value  observed  expected   2*0*ln(0/E)
       ------------------------------------------
             A B       544      83.3       2041.20
             A C         0      83.3          0.00
             A D         0      83.3          0.00
             B A       456      83.3       1550.08
             B C         0      83.3          0.00
             B D         0      83.3          0.00
             C A         0      83.3          0.00
             C B         0      83.3          0.00
             C D         0      83.3          0.00
             D A         0      83.3          0.00
             D B         0      83.3          0.00
             D C         0      83.3          0.00
       ------------------------------------------
                      1000    1000.0       3591.27

    G-statistic = 3591.27 (p-value = 0.000000, reject if p-value <= 0.0001)
    Note: a correct solution will fail this test by bad luck 1 time in 10,000.

  * filename = permutation4.txt, k = 3

           value  observed  expected   2*0*ln(0/E)
       ------------------------------------------
           A B C       168      41.7        468.47
           A B D         0      41.7          0.00
           A C B       164      41.7        449.41
           A C D         0      41.7          0.00
           A D B         0      41.7          0.00
           A D C         0      41.7          0.00
```

```
            B A C       159      41.7        425.87
            B A D         0      41.7          0.00
            B C A       165      41.7        454.16
            B C D         0      41.7          0.00
            B D A         0      41.7          0.00
            B D C         0      41.7          0.00
            C A B       172      41.7        487.72
            C A D         0      41.7          0.00
            C B A       172      41.7        487.72
            C B D         0      41.7          0.00
            C D A         0      41.7          0.00
            C D B         0      41.7          0.00
            D A B         0      41.7          0.00
            D A C         0      41.7          0.00
            D B A         0      41.7          0.00
            D B C         0      41.7          0.00
            D C A         0      41.7          0.00
            D C B         0      41.7          0.00
          ----------------------------------------
                       1000    1000.0       2773.35

    G-statistic = 2773.35 (p-value = 0.000000, reject if p-value <= 0.0001)
    Note: a correct solution will fail this test by bad luck 1 time in 10,000.

  * filename = permutation4.txt, k = 4
  * filename = permutation6.txt, k = 2

==> FAILED

Test 5b: check that permutations are uniformly random
         (for inputs with duplicate strings)
  * filename = permutation5.txt, k = 1

            value  observed  expected   2*0*ln(O/E)
          ----------------------------------------
                  A      1000     600.0       1021.65
                  B         0     400.0          0.00
          ----------------------------------------
                       1000    1000.0       1021.65

    G-statistic = 1021.65 (p-value = 0.000000, reject if p-value <= 0.0001)
    Note: a correct solution will fail this test by bad luck 1 time in 10,000.

  * filename = permutation5.txt, k = 2

            value  observed  expected   2*0*ln(O/E)
          ----------------------------------------
                A A         0     300.0          0.00
                A B       492     300.0        486.78
                B A       508     300.0        535.13
                B B         0     100.0          0.00
          ----------------------------------------
                       1000    1000.0       1021.91

    G-statistic = 1021.91 (p-value = 0.000000, reject if p-value <= 0.0001)
    Note: a correct solution will fail this test by bad luck 1 time in 10,000.

  * filename = permutation5.txt, k = 3

            value  observed  expected   2*0*ln(O/E)
          ----------------------------------------
                A A A         0     100.0          0.00
                A A B       321     200.0        303.75
                A B A       329     200.0        327.51
                A B B         0     100.0          0.00
                B A A       350     200.0        391.73
                B A B         0     100.0          0.00
                B B A         0     100.0          0.00
          ----------------------------------------
                       1000    1000.0       1022.99

    G-statistic = 1022.99 (p-value = 0.000000, reject if p-value <= 0.0001)
    Note: a correct solution will fail this test by bad luck 1 time in 10,000.

  * filename = duplicates.txt, k = 3

            value  observed  expected   2*0*ln(O/E)
          ----------------------------------------
             AA BB BB       313      59.5       1039.05
             AA BB CC         0      29.8          0.00
             AA CC BB         0      29.8          0.00
             AA CC CC         0       6.0          0.00
             BB AA BB       348      59.5       1229.01
             BB AA CC         0      29.8          0.00
             BB BB AA       339      59.5       1179.46
             BB BB BB         0     178.6          0.00
             BB BB CC         0     119.0          0.00
             BB CC AA         0      29.8          0.00
             BB CC BB         0     119.0          0.00
             BB CC CC         0      29.8          0.00
             CC AA BB         0      29.8          0.00
             CC AA CC         0       6.0          0.00
             CC BB AA         0      29.8          0.00
             CC BB BB         0     119.0          0.00
             CC BB CC         0      29.8          0.00
             CC CC AA         0       6.0          0.00
             CC CC BB         0      29.8          0.00
          ----------------------------------------
                       1000    1000.0       3447.53

    G-statistic = 3447.53 (p-value = 0.000000, reject if p-value <= 0.0001)
    Note: a correct solution will fail this test by bad luck 1 time in 10,000.

  * filename = permutation8.txt, k = 2
```

```
          value  observed  expected   2*0*ln(0/E)
        ----------------------------------------
              A B         0      17.9          0.00
              A C         0      71.4          0.00
              A D         0      35.7          0.00
              B A         0      17.9          0.00
              B C         0      71.4          0.00
              B D         0      35.7          0.00
              C A         0      71.4          0.00
              C B         0      71.4          0.00
              C C         0     214.3          0.00
              C D         0     142.9          0.00
              D A         0      35.7          0.00
              D B         0      35.7          0.00
              D C         0     142.9          0.00
              D D      1000      35.7       6664.41
        ----------------------------------------
                       1000    1000.0       6664.41

    G-statistic = 6664.41 (p-value = 0.000000, reject if p-value <= 0.0001)
    Note: a correct solution will fail this test by bad luck 1 time in 10,000.

==> FAILED

Total: 6/9 tests passed!


================================================================
********************************************************************************
*  TIMING (substituting reference RandomizedQueue and Deque)
********************************************************************************

Timing Permutation
*-----------------------------------------------------------
Running 23 total tests.

Test 1: count calls to methods in StdIn
  * java Permutation 5 < distinct.txt
  * java Permutation 10 < permutation10.txt
  * java Permutation 1 < mediumTale.txt
  * java Permutation 20 < tale.txt
  * java Permutation 100 < tale.txt
  * java Permutation 16412 < tale.txt
==> passed

Test 2: count calls to methods in Deque and RandomizedQueue
  * java Permutation 5 < distinct.txt
  * java Permutation 10 < permutation10.txt
  * java Permutation 1 < mediumTale.txt
  * java Permutation 20 < tale.txt
  * java Permutation 100 < tale.txt
  * java Permutation 16412 < tale.txt
==> passed

Test 3: count calls to methods in StdRandom
  * java Permutation 5 < distinct.txt
  * java Permutation 10 < permutation10.txt
  * java Permutation 1 < mediumTale.txt
  * java Permutation 20 < tale.txt
  * java Permutation 100 < tale.txt
  * java Permutation 16412 < tale.txt
==> passed

Test 4: Time main() with k = 5, for inputs containing n random strings

                     n  seconds
-------------------------------
=> passed         1000     0.00
=> passed         2000     0.00
=> passed         4000     0.00
=> passed         8000     0.00
=> passed        16000     0.00
=> passed        32000     0.00
=> passed        64000     0.00
=> passed       128000     0.00
=> passed       256000     0.00
=> passed       512000     0.00
==> 10/10 tests passed


Test 5: Time main() with k = 1000, for inputs containing n random strings

                     n  seconds
-------------------------------
=> passed         1000     0.00
=> passed         2000     0.00
=> passed         4000     0.00
=> passed         8000     0.00
=> passed        16000     0.00
=> passed        32000     0.00
=> passed        64000     0.00
=> passed       128000     0.00
=> passed       256000     0.00
=> passed       512000     0.00
==> 10/10 tests passed


Total: 23/23 tests passed!


================================================================
```

```
********************************************************************************
*  MEMORY
********************************************************************************

Analyzing memory of Permutation
*-----------------------------------------------------------
Running 2 total tests.

Test 1: check that only one Deque or RandomizedQueue object is created
  * filename = distinct.txt, n = 9, k = 1
  * filename = distinct.txt, n = 9, k = 2
  * filename = distinct.txt, n = 9, k = 4
  * filename = tinyTale.txt, n = 12, k = 10
  * filename = tale.txt, n = 138653, k = 50
==> passed

Test 2: check that the maximum size of any Deque or RandomizedQueue object
        created is between k and n
  * filename = distinct.txt, n = 9, k = 1
  * filename = distinct.txt, n = 9, k = 2
  * filename = distinct.txt, n = 9, k = 4
  * filename = tinyTale.txt, n = 12, k = 10
  * filename = tale.txt, n = 138653, k = 5
  * filename = tale.txt, n = 138653, k = 50
  * filename = tale.txt, n = 138653, k = 500
  * filename = tale.txt, n = 138653, k = 5000
  * filename = tale.txt, n = 138653, k = 50000
==> passed

Test 3 (bonus): check that maximum size of any or Deque or RandomizedQueue object
                created is equal to k
  * filename = tale.txt, n = 138653, k = 5
  * filename = tale.txt, n = 138653, k = 50
  * filename = tale.txt, n = 138653, k = 500
  * filename = tale.txt, n = 138653, k = 5000
  * filename = tale.txt, n = 138653, k = 50000
==> passed

Total: 3/2 tests passed!


================================================================



********************************************************************************
*  MEMORY
********************************************************************************

Analyzing memory of Deque
*-----------------------------------------------------------
For tests 1-4, the maximum amount of memory allowed for a Deque
containing n items is 48n + 192.

Running 49 total tests.

Test 1a-1i: total memory usage after inserting n items,
            where n is a power of 2

                 n        bytes
--------------------------------------------------------------
=> passed       32         1576
=> passed       64         3112
=> passed      128         6184
=> passed      256        12328
=> passed      512        24616
=> passed     1024        49192
=> passed     2048        98344
=> passed     4096       196648
=> passed     8192       393256
==> 9/9 tests passed

Memory: 48.00 n + 40.00   (R^2 = 1.000)


Test 2a-2i: Total memory usage after inserting n items,
            when n is one more than a power of 2.

                 n        bytes
--------------------------------------------------------------
=> passed       33         1624
=> passed       65         3160
=> passed      129         6232
=> passed      257        12376
=> passed      513        24664
=> passed     1025        49240
=> passed     2049        98392
=> passed     4097       196696
=> passed     8193       393304
==> 9/9 tests passed

Memory: 48.00 n + 40.00   (R^2 = 1.000)


Test 3a-3i: Total memory usage after inserting 2n-1 items, and then
            deleting n-1 items, when n is one more than a power of 2.

                 n        bytes
--------------------------------------------------------------
=> passed       33         1624
=> passed       65         3160
=> passed      129         6232
=> passed      257        12376
```

```
=> passed      513        24664
=> passed     1025        49240
=> passed     2049        98392
=> passed     4097       196696
=> passed     8193       393304
==> 9/9 tests passed

Memory: 48.00 n + 40.00   (R^2 = 1.000)


Test 4a-4e: Total memory usage after inserting n items,
            and then deleting all but one item
            (should not grow with n or be too large of a constant).

                 n        bytes
------------------------------------------------------------
=> passed       32           88
=> passed       64           88
=> passed      128           88
=> passed      256           88
=> passed      512           88
=> passed     1024           88
=> passed     2048           88
=> passed     4096           88
=> passed     8192           88
==> 9/9 tests passed

Memory: 88.00   (R^2 = 1.000)


Test 5a-5e: Total memory usage of iterator after inserting n items
            (should not grow with n or be too large of a constant).

                 n        bytes
------------------------------------------------------------
=> passed       32           32
=> passed       64           32
=> passed      128           32
=> passed      256           32
=> passed      512           32
=> passed     1024           32
=> passed     2048           32
=> passed     4096           32
=> passed     8192           32
==> 9/9 tests passed

Memory: 32.00   (R^2 = 1.000)


Test 6a: Insert n strings; delete them one at a time, checking for
         loitering after each deletion. The probabilities of addFirst()
         and addLast() are (p1, p2), respectively. The probabilities of
         removeFirst() and removeLast() are (q1, q2), respectively.
  * 100 random insertions (1.0, 0.0) and 100 random deletions (1.0, 0.0)
  * 100 random insertions (1.0, 0.0) and 100 random deletions (0.0, 1.0)
  * 100 random insertions (0.0, 1.0) and 100 random deletions (1.0, 0.0)
  * 100 random insertions (0.0, 1.0) and 100 random deletions (0.0, 1.0)
  * 100 random insertions (0.5, 0.5) and 100 random deletions (0.5, 0.5)
==> passed

Test 6b: Perform random operations, checking for loitering after
         each operation. The probabilities of addFirst(), addLast(),
         removeFirst(), and removeLast() are (p1, p2, p3, p4),
         respectively.
  * 100 random operations (0.8, 0.0, 0.2, 0.0)
  * 100 random operations (0.8, 0.0, 0.0, 0.2)
  * 100 random operations (0.0, 0.8, 0.2, 0.0)
  * 100 random operations (0.0, 0.8, 0.0, 0.2)
  * 100 random operations (0.4, 0.4, 0.1, 0.1)
  * 100 random operations (0.2, 0.2, 0.3, 0.3)
==> passed

Test 7: Perform m random add/remove operations in the deque and check
        that only constant memory is allocated/deallocated per operation
  * m = 128
  * m = 256
  * m = 512
==> passed

Test 8: Insert m items into deque; then iterate over deque and check
        that only constant memory is allocated/deallocated per operation
  * m = 64
  * m = 128
  * m = 256
==> passed


Min observed memory for Deque: 48.00 n + 40.00   (R^2 = 1.000)
Max observed memory for Deque: 48.00 n + 40.00   (R^2 = 1.000)

Total: 49/49 tests passed!


================================================================



Analyzing memory of RandomizedQueue
*-----------------------------------------------------------
For Tests 1-5, the maximum amount of memory allowed for
a RandomizedQueue containing n items is 48n + 192.

For Test 6, the maximum amount of memory allowed for
a RandomizedQueue iterator over n items is 8n + 72.
```

```
Test 1a-1i: Total memory usage after inserting n items
            when n is a power of 2.

                  n        bytes
--------------------------------------------------------------
=> passed        32          312
=> passed        64          568
=> passed       128         1080
=> passed       256         2104
=> passed       512         4152
=> passed      1024         8248
=> passed      2048        16440
=> passed      4096        32824
=> passed      8192        65592
==> 9/9 tests passed

Memory: 8.00 n + 56.00   (R^2 = 1.000)


Test 2a-2i: Total memory usage after inserting n items,
            when n is one more than a power of 2.

                  n        bytes
--------------------------------------------------------------
=> passed        33          568
=> passed        65         1080
=> passed       129         2104
=> passed       257         4152
=> passed       513         8248
=> passed      1025        16440
=> passed      2049        32824
=> passed      4097        65592
=> passed      8193       131128
==> 9/9 tests passed

Memory: 16.00 n + 40.00   (R^2 = 1.000)


Test 3a-3i: Total memory usage after inserting 2n-1 items, and then
            deleting n-1 items, when n is one more than a power of 2.

                  n        bytes
--------------------------------------------------------------
=> passed        33         1080
=> passed        65         2104
=> passed       129         4152
=> passed       257         8248
=> passed       513        16440
=> passed      1025        32824
=> passed      2049        65592
=> passed      4097       131128
=> passed      8193       262200
==> 9/9 tests passed

Memory: 32.00 n + 24.00   (R^2 = 1.000)


Test 4a-4i: Total memory usage after inserting n items, deleting n items,
            then inserting n times, when n is a power of 2.

                  n        bytes
--------------------------------------------------------------
=> passed        32          312
=> passed        64          568
=> passed       128         1080
=> passed       256         2104
=> passed       512         4152
=> passed      1024         8248
=> passed      2048        16440
=> passed      4096        32824
=> passed      8192        65592
==> 9/9 tests passed

Memory: 8.00 n + 56.00   (R^2 = 1.000)


Test 5a-5i: Total memory usage after inserting n items,
            and then deleting all but one item.

                  n        bytes
--------------------------------------------------------------
=> passed        32           72
=> passed        64           72
=> passed       128           72
=> passed       256           72
=> passed       512           72
=> passed      1024           72
=> passed      2048           72
=> passed      4096           72
=> passed      8192           72
==> 9/9 tests passed

Memory: 72.00   (R^2 = 1.000)


Test 6a-6i: Total memory usage of iterator after inserting n items.

                  n        bytes
--------------------------------------------------------------
=> passed        32          320
=> passed        64          576
=> passed       128         1088
=> passed       256         2112
```

```
=> passed      512         4160
=> passed     1024         8256
=> passed     2048        16448
=> passed     4096        32832
=> passed     8192        65600
==> 9/9 tests passed

Memory: 8.00 n + 64.00   (R^2 = 1.000)


Test 7a: Insert 100 strings; delete them one at a time, checking
         for loitering after each deletion.
==> passed

Test 7b: Perform random operations, checking for loitering after
         each operation. The probabilities of enqueue(), dequeue(),
         and sample() are (p1, p2, p3), respectively.
  * 200 random operations (0.8, 0.2, 0.0)
  * 200 random operations (0.2, 0.8, 0.0)
  * 200 random operations (0.6, 0.2, 0.2)
  * 200 random operations (0.2, 0.4, 0.4)
==> passed

Test 8: Insert m items into queue; then iterate over deque and check
        that only constant memory is allocated/deallocated per operation
  * m = 64
  * m = 128
  * m = 256
==> passed

Test 9: Total memory usage after inserting n items, seeking to identify
        values of n where memory usage is minimized as a function of n.

                 n        bytes
-------------------------------------------------------------
=> passed       32          312
=> passed       64          568
=> passed      128         1080
=> passed      256         2104
=> passed      512         4152
=> passed     1024         8248
=> passed     2048        16440
==> 7/7 tests passed

Memory: 8.00 n + 56.00   (R^2 = 1.000)


Test 10: Total memory usage after inserting 4096 items, then successively
         deleting items, seeking values of n where memory usage is maximized
         as a function of n

                 n        bytes
-------------------------------------------------------------
=> passed     2049        65592
=> passed     1025        32824
=> passed      513        16440
=> passed      257         8248
=> passed      129         4152
=> passed       65         2104
=> passed       33         1080
=> passed       17          568
=> passed        9          312
==> 9/9 tests passed

Memory: 32.00 n + 24.00   (R^2 = 1.000)


Min observed memory for RandomizedQueue: 8.00 n + 56.00   (R^2 = 1.000)
Max observed memory for RandomizedQueue: 32.00 n + 24.00   (R^2 = 1.000)

Running 73 total tests.

Total: 73/73 tests passed!


================================================================



********************************************************************************
*  TIMING
********************************************************************************

Timing Deque
*-----------------------------------------------------------
Running 103 total tests.

Test 1a-1k: make n calls to addFirst() followed by n calls to removeFirst()
                   n  seconds
----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.00
=> passed      256000     0.00
=> passed      512000     0.01
=> passed     1024000     0.02
==> 11/11 tests passed
```

```
Test 2a-2k: make n calls to addLast() followed by n calls to removeLast()
                    n  seconds
-----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.01
=> passed      256000     0.00
=> passed      512000     0.01
=> passed     1024000     0.02
==> 11/11 tests passed


Test 3a-3k: make n calls to addFirst() followed by n calls to removeLast()
                    n  seconds
-----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.00
=> passed      256000     0.00
=> passed      512000     0.01
=> passed     1024000     0.02
==> 11/11 tests passed


Test 4a-4k: make n calls to addLast() followed by n calls to removeFirst()
                    n  seconds
-----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.00
=> passed      256000     0.00
=> passed      512000     0.01
=> passed     1024000     0.02
==> 11/11 tests passed


Test 5a-5g: make n random calls to addFirst(), removeFirst(), isEmpty(), and size()
            with probabilities (0.7, 0.1, 0.1, 0.1)

                    n  seconds
-------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.00
=> passed      256000     0.01
=> passed      512000     0.01
=> passed     1024000     0.03
=> passed     2048000     0.06
==> 12/12 tests passed


Test 6a-6g: make n random calls to addLast(), removeLast(), isEmpty(), and size(),
            with probabilities (0.7, 0.1, 0.1, 0.1)

                    n  seconds
-------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.00
=> passed      256000     0.01
=> passed      512000     0.02
=> passed     1024000     0.03
=> passed     2048000     0.06
==> 12/12 tests passed


Test 7a-7g: make n random calls to addFirst(), addLast(), removeFirst(), removeLast(),
            isEmpty(), and size() with probabilities (0.3, 0.3, 0.1, 0.1, 0.1, 0.1)

                    n  seconds
-------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.00
```

```
=> passed      256000     0.01
=> passed      512000     0.02
=> passed     1024000     0.04
=> passed     2048000     0.07
==> 12/12 tests passed


Test 8a-8g: make n calls to addFirst(); iterate over the n items by calling
            next() and hasNext()

                    n  seconds
-------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.00
=> passed      256000     0.00
=> passed      512000     0.01
=> passed     1024000     0.02
=> passed     2048000     0.04
==> 12/12 tests passed


Test 9a-9k: make n calls to addFirst()/addLast(); interleave n calls each to
            removeFirst(), removeLast(), addFirst(), and addLast()

                    n  seconds
-----------------------------------
=> passed        1025     0.00
=> passed        2049     0.00
=> passed        4097     0.00
=> passed        8193     0.00
=> passed       16385     0.00
=> passed       32769     0.00
=> passed       65537     0.00
=> passed      128001     0.01
=> passed      256001     0.02
=> passed      512001     0.04
=> passed     1024001     0.05
==> 11/11 tests passed

Total: 103/103 tests passed!


================================================================



Timing RandomizedQueue
*-----------------------------------------------------------
Running 67 total tests.

Test 1: make n calls to enqueue() followed by n calls to dequeue();
        count calls to StdRandom
  * n = 10
  * n = 100
  * n = 1000
==> passed

Test 2: make n calls to enqueue() follwed by n calls to sample();
        count calls to StdRandom
  * n = 10
  * n = 100
  * n = 1000
==> passed

Test 3: make n calls to enqueue() and iterate over the n items;
        count calls to StdRandom
  * n = 10
  * n = 100
  * n = 1000
==> passed

Test 4a-k: make n calls to enqueue() followed by n calls to dequeue()

                    n  seconds
-----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.01
=> passed      256000     0.01
=> passed      512000     0.01
=> passed     1024000     0.03
==> 11/11 tests passed


Test 5a-k: make n calls to enqueue() followed by n random calls to
           enqueue(), sample(), dequeue(), isEmpty(), and size()
           with probabilities (0.2, 0.2, 0.2, 0.2, 0.2)

                    n  seconds
-----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
```

```
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.01
=> passed      128000     0.01
=> passed      256000     0.02
=> passed      512000     0.04
=> passed     1024000     0.09
==> 11/11 tests passed


Test 6a-k: make n calls to enqueue() followed by n random calls to
           enqueue(), sample(), dequeue(), isEmpty(), and size()
           with probabilities (0.6, 0.1, 0.1, 0.1, 0.1)

                   n  seconds
-----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.01
=> passed      256000     0.01
=> passed      512000     0.03
=> passed     1024000     0.07
==> 11/11 tests passed


Test 7a-k: make n calls to enqueue() followed by n random calls to
           enqueue(), sample(), dequeue(), isEmpty(), and size()
           with probabilities (0.1, 0.1, 0.6, 0.1, 0.1)

                   n  seconds
-----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.01
=> passed      256000     0.02
=> passed      512000     0.04
=> passed     1024000     0.11
==> 11/11 tests passed


Test 8a-k: make n calls to enqueue() followed by n calls each to
           next() and hasNext().

                   n  seconds
-----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.00
=> passed      128000     0.01
=> passed      256000     0.01
=> passed      512000     0.02
=> passed     1024000     0.04
==> 11/11 tests passed


Test 9a-i: make 100 calls to enqueue; 99 calls to dequeue;
           n calls to enqueue(); then call dequeue() three times,
           followed by enqueue() three times, and repeat n times.

                   n  seconds
-----------------------------------
=> passed        1024     0.00
=> passed        2048     0.00
=> passed        4096     0.00
=> passed        8192     0.00
=> passed       16384     0.00
=> passed       32768     0.00
=> passed       65536     0.01
=> passed      128000     0.02
=> passed      256000     0.04
==> 9/9 tests passed


Total: 67/67 tests passed!


================================================================
```