Problem 1

For the circular array implementation of FIFO queues in CircularArray.java, implement a method size() which returns the number of elements in the queue as an int.

Answer:

```java
public int size() {
    if (end >= start) return end-start;
    else return elements.length+end - start;
}
```

Problem 2

For the generic ordered singly linked list in GOrderedList.java, write a generic method L.numBetween(T Lower, T Upper) which returns the number of elements in L that are strictly greater than Lower and strictly less than Upper. For instance if L is the list [2,3,5,6,10,14] then L.numBetween(5,11) should return 2, corresponding to the elements 6 and 10.

Answer:

```java
public int numBetween(T Lower, T Upper) {
    GOrderedList<T> N = next;
    while (N != null && N.value.compareTo(Lower) <= 0)
        N = N.next;
    if (N == null) return 0;
    int count = 1;
    N = N.next;
    while (N != null && N.value.compareTo(Upper) < 0) {
        count++;
        N = N.next;
    }
    return count;
}
```
Problem 3

In the hash table implementation in MyHashTable.java, if a key KEY is already in the table, and you execute the method put(KEY, VALUE), it does nothing at all. Obviously, it would be better to have it change the value associated with KEY to be VALUE. Rewrite the method put so that it does this.

Answer:

```java
public void put(String Key, Person PP) {
    int Index = MyHash(Key);
    MyNode N = FindNode(Key, Index);
    if (N == null) {
        N = new MyNode(Key, PP);
        N.next = table[Index];
        table[Index] = New;
    }
    else N.value = P;
}
```

Problem 4

In the method MyHash in MyHashTable.java, if the table size is close to a multiple of 37, it chooses the multiplicative factor Mult to be 43 rather than 37. You get into particularly bad trouble if the table size is an exact power of Mult. Explain why. Hint: Delete this line from the method, create a table of length 37, and see what hash values you get for the strings “aaaa”, “aazz” “zzaa” “zzzz”, “abcd”, “zycd”, “abxw”, and “zyxw”. Do the same experiment with a table of size $37^2 = 1369$. Describe what is going on, explain why this happens, and explain why it would be a very bad thing for a hash function.

Answer: Suppose that Size = Mult$^k$. Then for a given string S with n characters,

$$S.myhash() = (S[0] * Mult^{n-1} + S[1] * Mult^{n-2} + \ldots + S[n-2] * Mult + S[n-1]) \mod Mult^k =$$

$$S[0] * Mult^{n-1} \mod Mult^k + S[1] * Mult^{n-2} \mod Mult^k + \ldots + S[n-2] * Mult \mod Mult^k + S[n-1] \mod Mult^k$$

But in the second sum, all of the terms involving powers of Mult greater than k are 0. That is, the hash function ignores all but the last k characters; any two strings with the same final k characters hash to the same value.
Honors Problem

Continuing Problem 4, it is also a bad thing if the table size is a power of $\text{Mult}$ minus 1; e.g. $\text{Mult}=37$ and the table size is 36 or $1368 = 37^2 - 1$. Explain.

**Answer:** Suppose that $\text{Size} = \text{Mult}^k - 1$. Then $\text{Mult}^k \% \text{Size} = 1$. Likewise

$$\text{Mult}^{2k} \% \text{Size} = (\text{Mult}^k)^2 \% \text{Size} = (\text{Mult}^k \% \text{Size}) \cdot (\text{Mult}^k \% \text{Size}) = 1 \cdot 1 = 1$$

In general

$$\text{Mult}^{nk} \% \text{Size} = (\text{Mult}^k)^n \% \text{Size} = 1^n = 1$$

Therefore what the hash function does is to break the string into groups of $k$ characters, view each group as a numeral base $\text{Mult}$, and add these up. For instance if $k = 3$, $\text{Mult}=37$, and $\text{Size} = 37^3 - 1 = 50,652$, then the string “Constantinople” is hashed to the following sum (interpreting characters as their numeric equivalents).

\[
\begin{align*}
'c' \cdot 37^2 &+ 'o' + \\
'n' \cdot 37^2 &+ 's' \cdot 37 + 't' + \\
'a' \cdot 37^2 &+ 'n' \cdot 37 + 't' + \\
'i' \cdot 37^2 &+ 'n' \cdot 37 + 'o' + \\
'p' \cdot 37^2 &+ 't' \cdot 37 + 'c'
\end{align*}
\]

and then reduced mod $\text{Size}$. This is equal to

$$\left[(n' + a' + t' + p') \cdot 37^2 + (C' + s' + n' + t') \cdot 37 + o' + t' + e' \right] \% \text{Size}$$

This is likely to lead to a lot of collisions. For example if you can change one word to another by swapping pairs of letters $k$ spaces apart such as “pear” and “reap”, or “loot” and “tool”, then they hash to the same value. Likewise if you can turn one word into another by shifting one letter up $g$ spots in the alphabet and shifting a letter $k$ places away down $q$ spots, they will hash to the same value, such as “pear” and “neat” (shift ‘p’ down 2 and ‘r’ up 2). Because of the regularities of the construction of words, these operations tend to work disproportionately often, and therefore give many collisions.

If you didn’t get this problem, don’t worry about it; this is a more difficult problem than most.