This homework will give you practice implementing linear-time sorting algorithms. This is an individual assignment; you may not share code with other students.

Copy the Homework 10 directory by doing the following, starting from your home directory. Don’t forget the "-r" switch in the cp command.

    mkdir hw10
    cd hw10
    cp -r $master/hw/hw10/*

Your job is to implement counting sort and radix sort for arrays of ints. All your code should appear in the file sort/Sorts.java. A skeleton is provided for you.

Part I (8 points)
----------------
Implement counting sort on int arrays.

    public static int[] countingSort(int[] keys, int whichDigit);

The most important difference between the counting sort you will implement here and the one presented in lecture is that this counting sort uses one base-16 digit in each int as its sort key, and ignores all the other digits. That way, your counting sort can be used as one pass of radix sort. (Make sure that your counting sort is stable!)

The parameter "whichDigit" tells the method which base-16 digit of each int to use as a sort key. If whichDigit is zero, sort on the least significant (ones) digit; if whichDigit is one, sort on the second least significant (sixteens) digit; and so on. An int is 32 bits long, so it has eight digits of four bits each.

The high bit of each int is a sign bit. To keep life simple, we will assume that all the numbers are positive, so the high bit is always zero. Don’t try to create an int whose most significant base-16 digit is greater than 7.

Hexadecimal Primer: Hexadecimal is a way of expressing a number in base-16. We use the usual digits 0...9, plus the additional digits a...f to represent eleven through fifteen. You can convert back and forth between an int and a hexadecimal string by using the Integer.toHexString(int) and Integer.parseInt(String, int) methods. (Look them up in the online Java API, and/or look at how they are used in Sorts.java.)

One of the best reasons to use base-16 digits is because they can be extracted very quickly from a key by using bit operations. This means that, in your countingSort method, you should use bit operations to extract the digits, and not throw away the speed advantage by using something silly like toString() to extract each digit. The bit operation that will serve you best is Java’s "&" operator. If you write "x & 15", it masks the int x against the bit pattern "0000...00001111", so only the least significant base-16 digit survives, and the others are set to zero. This allows you to extract the least significant digit.

Want to extract a different digit? Divide the int by some appropriate divisor first. Recall that integer division always rounds down, so you can eliminate low-order digits this way if you choose the right divisor. This moves the digit you’re looking for down to the least significant position, so you can mask it against a 15. (For faster performance, shift the bits to the right, if you know how to do so.)