EXERCISE 1: Maximum Flow

Enthusiastic celebration of a sunny day at a prominent northeastern university has resulted in the arrival at the university's medical clinic of `169` students in need of emergency treatment. Each of the `169` students requires a transfusion of one unit of whole blood. The clinic has supplies of `170` units of whole blood. The number of units of blood available in each of the four major blood groups and the distribution of patients among the groups is summarized below.

Blood Type	**AB**	**O**	**B**	**A**
Supply	45	45	34	46
Demand	50	42	38	39

Blood Type	**AB**	**O**	**A**	**B**
Can donate to	AB	*any*	*A, AB*	B, AB

How should the blood units be distributed such that the maximum number of patients receive blood units?

A. Formulate the problem as a **Max-Flow** problem. Show your work by drawing the flow network corresponding to the above instance of the problem. Mark the `source` and `sink` vertices and the `capacities` of the directed edges.

B. Show the flow through each edge in the network after applying ***Ford-Fulkerson***. What is the maximum number of patients that can receive blood units?

C. Find the ***minimum-cut*** in the max-flow network. Use the vertices on the *sink* side of the min-cut to explain why the demand of some patients can't be met.

EXERCISE 2: A Simplified MSD String Sort

Consider the following **MSD** code from the lecture slides. This code assumes that all the strings in the array of ***equal length.***

```
public static void sort(String[] a)
{
   aux = new String[a.length];
   sort(a, aux, 0, a.length - 1, 0);
}

private static void sort(String[] a, String[] aux, int lo, int hi, int d)
{
   if (hi <= lo) return;

   int[] count = new int[R+1];
   for (int i = lo; i <= hi; i++)
      count[a[i].charAt(d) + 1]++;
   for (int r = 0; r < R; r++)
      count[r+1] += count[r];
   for (int i = lo; i <= hi; i++)
      aux[count[a[i].charAt(d)]++] = a[i];
   for (int i = lo; i <= hi; i++)
      a[i] = aux[i - lo];

   for (int r = 0; r < R; r++)
      sort(a, aux, lo + count[r], lo + count[r+1] - 1, d+1);

}
```

recycles aux[] array but not count[] array

key-indexed counting

sort R subarrays recursively

Modify the code to use an ***array of queues*** for *key-indexed counting* instead of the `count` and `aux` arrays.

You are given a template of the code to modify in the next page. You can also use the online version of the exercise to run and test your code: http://bit.ly/array-of-queues-msd or download the IntelliJ project from the precepts page.

```java
private static final int R = 256;

public static void sort(String[] a) {
    if (a.length == 0) return;
    int w = a[0].length(); // all strings are assumed to be of the same length
    sort(a, 0, a.length-1, w, 0);
}

// Sort from a[lo] to a[hi], based on the dth character.
private static void sort(String[] a, int lo, int hi, int w, int d) {
    if (hi <= lo || d >= a[lo].length()) return;

     // The queue at bins[r] holds all the strings whose dth character is r.
    Queue<String>[] bins = (Queue<String>[]) new Queue[R];
    for (int r = 0; r < R; r++)
        bins[r] = new Queue<String>();

    // TODO: Add each string in the range a[lo ... hi] to its correct bin.









    // TODO: Use the bins array to distribute the strings
    // back to a[lo ... hi] sorted based on the dth character.









    // TODO: Recursively apply MSD to sort each bin.
    int from = lo;
    for (int r = 0; r < R; r++) {

        int to = _____________________;
        sort(a, from, to, w, d+1);

        from += _____________________;
    }
}
```