CMPSC 463 Problem Set #5 (100 points)

1. Consider the partition algorithm from quicksort. (30 points)

   a) Using this algorithm, write a method that finds the kth largest element in an array of integers. This element is defined as with minimum value out of all elements whose value is greater than or equal to at least k-1 elements in the array.

```
Note: If you are using a 0-based array, you would need to make
appropriate adjustments.

findKthLargest(array[1..n],k)
    return helper(array,1,n,k)
end findKthLargest

helper(array[1..n],p,q,k)
    r = partition(array,p,q)
    if r == k then
        return array[k]
    else if k < r then
        return helper(array,p,r-1,k)
    else
        return helper(array,r+1,q,k)
    end if
end helper
```

   b) Suppose that your algorithm always picks a "good enough" pivot, defined as one that reduces the size of the array that contains the desired element by at least 3/4. Write a recurrence relation that expresses the worst case running time of your algorithm.

$$T(n) = T\left(\frac{3}{4}n\right) + c_1 n$$

   c) Solve the recurrence relation.

Let's see if we can apply the Master Theorem:
$a = 1, b = \frac{4}{3}, f(n) = c_1 n, h(n) = n^{\log_{\frac{4}{3}} 1} = 1$
$\frac{f(n)}{h(n)} = c_1 n = \Omega(n)$
Checking the regularity condition:
$af\left(\frac{n}{b}\right) = \frac{c_1 n}{4/3} = \frac{3}{4}c_1 n = \frac{3}{4}f(n)$. Choosing c=3/4, the regularity condition holds.
Thus $T(n) = \theta\big(\mathrm{f(n)}\big) = \theta(\mathrm{n})$

   d) If we can choose a "good enough pivot" at least half of the time, what would be the average case running time of your algorithm?

We would expect that the average case running time would be twice this calculated amount. Thus, $T_A(n) = \theta(2\mathrm{n}) = \theta(\mathrm{n})$

2. The *One Size Fits One* t-shirt company that makes unique sizes for its customers. Unfortunately, in the latest batch of t-shirts, the customer assignments for each t-shirt have been lost. Moreover, it is impossible to sort the shirts (or the people) just by looking at them. The customers have been invited to try on the shirts to try to figure out how to assign shirts. When a customer tries on a shirt, you will be only tell if the shirt is too small, fits, or is too large.

Your task is to write two methods in Java to help with the assignment given n unique customers and n unique shirts. You should use the following template when writing your code.

a) The first method should be an $\theta(n^2)$ expected time algorithm that rearranges the people and/or the shirts such that such that `persons.get(i)` fits `shirts.get(i)` for all $1 \leq i \leq n$. (30 points)

```
// Idea:  For each shirt we scan the persons array to find a matching person
static void methodA(ArrayList<Shirt> shirts, ArrayList<Person> persons) {
  for (int i = 0; i < shirts.size(); i++) {
    boolean foundMatch = false;
    for (int j = i; !foundMatch && j < persons.size(); j++) {
      if (shirts.get(i).equals(persons.get(j))) {
        Collections.swap(persons, i, j);
        foundMatch = true;
      }
    }
  }
}
```

b) The second method should be a $\theta(n \log n)$ expected time algorithm that rearranges the nuts and/or the bolts such that such that `persons.get(i)` fits `shirts.get(i)` for all $1 \leq i \leq n$. (40 points)

Note: I think most people will create two different partition methods, which is fine.

```
static void methodB(ArrayList<Shirt> shirts, ArrayList<Person> persons) {
  methodBHelper(shirts, persons, 0, shirts.size()-1);
}

// Sort items using a quicksort-like approach
static void methodBHelper(ArrayList<Shirt> shirts, ArrayList<Person> persons,
                          int p, int q) {
  if (p < q) {
    // Partition shirts[p..q] based on persons[q]
    int r = partitionItems(shirts, persons.get(q), p, q);
    // Partition persons[p..q] based on shirts[r]
    partitionItems(persons, shirts.get(r), p, q);

    methodBHelper(shirts,persons,p,r-1);
    methodBHelper(shirts,persons,r+1,q);
  }
}
```

```java
/** Partition the items[p..q] using pivot, such that:
 *  each item in items[p..(r-1)] is too small item[q]
 *  each item in items[(r+1)..q] is too big for item[q]
 *  item[r] fits item[q]
 *
 *  Precondition: The arraylist and the pivot must be of different
 *                subclasses
 *
 * @return r
 */
static int partitionItems(ArrayList<? extends Item> items, Item pivot, int p,
                          int q) {
  int r = p;

  // Loop invariant:
  //   All items in items[p..(r-1)] are smaller than pivot
  //   All items in items[r..(i-1)] are greater than pivot
  for (int i = p; i < q; i++) {
    if (items.get(i).equals(pivot)) {
      // We have a match - move it to end, an decrement i
      Collections.swap(items, i, q);
      i--;
    }
    else {
      // Cast items so that we can compare
      boolean tooSmall = false;
      if (items.get(i) instanceof Person) {
        tooSmall = ((Person) items.get(i)).isTooSmallFor((Shirt) pivot);
      }
      else {
        tooSmall = ((Shirt) items.get(i)).isTooSmallFor((Person) pivot);
      }

      // Move small items to beginning
      if (tooSmall) {
        Collections.swap(items, i, r);
        r++;
      }
    }
  }
  Collections.swap(items, r, q);
  return r;
}
```

The algorithms should use the classes in the template, but you may only call the `equals` (aka fits) and `isTooSmall` methods in your algorithm, i.e. you may not access the protected size variables.

Please submit your answer to problem 1 as a pdf file, and your answer to problem 2 as a .java file. You may change the methodA and methodB methods, and add methods to the Solution class, but otherwise the template code should be left unchanged.