Department of Computer Science **University of San Francisco**

Computer Science 673
Fall 2016
Homework 3: Sorting
Due Friday, September 16th

All problem / exercise numbers are from the **3rd edition** of Introduction to Algorithms (the 1st and 2nd editions are different!) Note the difference between problems and exercises!

1. Exercise 5.3-2 (4 points)

   Professor Kelp decides to write a procedure that will produce at random any permutation besides the identity permutation. He proposes the following procedure:

   ```
   PERMUTE–WITHOUT–IDENTITY(A)
   1.   n ← length(A)
   2.   for i ← 1 to n do
   3.        swap A[i] ↔ A[RANDOM(i+1,n)]
   ```

   Prof Kelp wants every permuation except the identity permutation to have the same probability, while he wants identity permutation (that is, no changes to the original array) to have the probability 0. Does this code do what professor Kelp intends?

2. Consider the original HIRE-ASSISTANT problem, where we interview canditates one at a time, and always hire the next person if he/she is better than who we currently have:

   ```
   HIRE-ASSISTANT(n)
     Hire Candidate[1]
     Best ← 1
     for i ← 2 to n do
       if Candidate[i] is better than Candidate[Best]
         Best ← i
         Hire Candidate[i]
   ```

   (a) (1 points) What is the probability that you will hire exactly once (as a function of n). Explain you answer!

   (b) (1 points) What is the probability that you will hire exactly n times (as a function of n)? Explain your answer!

   (c) (4 points) What is the probabilty that you will hire exactly twice (as a function of n)? Explain your answer! This one is much harder than the first 2 – be sure you don't miss any cases!

3. (4 points) In Randomized-Quicksort of a list of length n, what is the largest number of times that RANDOM will be called? What is the smallest possible number of times that RANDOM will be called? Be as exact as possible.

4. Problem 7-4 **Stack Depth for Quicksort**

```
Tail-Recursive-Quicksort($A, p, r$)
    while $p < r$
    do
        $q \leftarrow$ Partition($A, p, r$)
        Tail-Recursive-Quicksort($A, p, q - 1$)
        $p \leftarrow q + 1$
```

   (a) (3 points) Argue that Tail-Recursive-Quicksort(A, 1, A.length) correctly sorts the array A.

   (b) (3 points) Describe a scenario in which the stack depth of Tail-Recursive-Quicksort is $\Theta(n)$ on an n-element input array.

   (c) (3 points) Modify the code for Tail-Recursive-Quicksort so that the worst-case stack depth is $\Theta(\lg n)$. Maintain the $O(n \lg n)$ expected running time of the algorithm.

5. Exercise 8.3-2 (6 points)

   A sorting algorithm is *stable* if it preserves the order of duplicate elements. (Consider the case where we are sorting an arrray records based on a key field. Two different records could have the same key. If the sort is stable, then for any two records x and y with the same key field, x would appear before y in the sorted list if and only if x appeared before y in the original list.

   (a) (2 points) Which of the following sorting algorithms are stable: insertion, merge, heap, quick.

   (b) (4 points) Give a simple scheme that makes *any* sorting algorithm stable. How much additional time and space does your sorting algorithm require?