COMPSCI330 Design and Analysis of Algorithms
Assignment 4

Due Date: Wednesday, Oct 4, 2017

Guidelines

- **Describing Algorithms** If you are asked to provide an algorithm, you should clearly define each step of the procedure, and then analyze its overall running time. There is no need to write pseudo-code; an unambiguous description of your algorithm in plain text will suffice. If the running time of your algorithm is worse than the suggested running time, you might receive partial credits.
- **Typesetting and Submission** Please submit each problem as an *individual* pdf file for the correct problem on Sakai. LaTeX is preferred, but answers typed with other software and converted to pdf is also accepted. Please make sure you submit to the correct problem, and your file can be opened by standard pdf reader. **Handwritten answers or pdf files that cannot be opened will not be graded.**
- **Timing** Please start early. The problems are difficult and they can take hours to solve. The time you spend on finding the proof can be much longer than the time to write. If you submit within one week of the deadline you will get half credit. Any submission after that will not receive any credit.
- **Collaboration Policy** Please check this page for the collaboration policy. You are **not** allowed to discuss homework problems in groups of more than 3 students. **Failure to adhere to these guidelines will be promptly reported to the relevant authority without exception.**

Problem 1 (Coffee Tasting)**.** (15 points) R and his wife W went to Seattle for a vacation. They visited a coffee shop where you can order a flight of three different kinds of coffee. To test whether R is able to taste the differences in the coffee, they decided to play a game: W will select a cup of coffee while R close his eyes. Then R will taste the coffee and try to decide which kind of coffee it is. R is not very good at telling the difference between different kinds of coffee, so to help him, once R made the decision, W will remove one of the incorrect answers (if R's current decision is correct, one of the other two is going to be removed with probability 1/2; if R's current decision is incorrect, then the other incorrect answer is going to be removed). R can then decide whether he wants to switch his choices.

(a) (5 points) If R really has no idea on what the coffee is (all three kinds of coffee taste the same to him), what should be his decision (does he switch or not)? What is the probability of getting the final answer correct?

(b) (10 points) R is not completely clueless. After tasting he believes the coffee is #1 with probability p_1, #2 with probability p_2 and #3 with probability p_3 (without loss of generality we will assume $p_1 < p_2 < p_3$, and of course $p_1 + p_2 + p_3 = 1$). Now, what is the best strategy for R? (A strategy would include which number to pick at the beginning, and whether to switch or not depending on the number that is removed.) What is the probability of getting the final answer correct? (In this problem you don't need to prove the strategy is optimal.)

Problem 2 (Quick Selection)**.** (20 points) In class we have described the Quick Selection algorithm

```
QuickSelection(A[], k)
If length(A) == 1 Then return A[1]
    Pick a random pivot p in array A
    Partition A such that A[i] = p, A[1..i-1] contains elements smaller than p
         and A[i+1..n] contains elements larger than p
    If k == i Then return p
    Else If k < i Then return QuickSelection(A[1..i-1], k)
    Else return QuickSelection(A[i+1..n], k-i)
```

Use induction to prove the expected running time of Quick Selection on an array of n elements is bounded by Cn for some constant C. You can assume the partition step takes exactly n time for an array of size n. You can also assume there are no two elements with the same value.

Let X_n be the running time of QuickSelection for n numbers, the recursion you should solve is

$$\mathbb{E}[X_n] = \frac{1}{n}\sum_{i=1}^{k-1}\mathbb{E}[X_{n-i}] + \frac{1}{n}\sum_{i=k+1}^{n}\mathbb{E}[X_{i-1}] + n.$$

Problem 3 (Hashing without Linked Lists)**.** (25 points) In class we talked about how to handle collisions in hashing by using a linked list for each location in your Hash table. However, in some cases you might want to make sure your hash table takes a *fixed* amount of memory. In those cases you can handle collisions in a different way. Suppose your hash table has n entries $0, 1, ..., n-1$, and your hash function f maps keys to $0, 1, ..., n-1$. In addition to f, for each possible *key* value you will also have a random "cycle" function g_{key} that maps $0, 1, ..., n-1$ to $0, 1, ..., n-1$. The property of g includes

1. For any $x \neq y \in \{0, 1, ..., n-1\}$, $g_{key}(x) \neq g(y)$.
2. For any x, let $x^1 = g_{key}(x)$, $x^2 = g_{key}(g_{key}(x))$, ..., $x^{t+1} = g_{key}(x^t)$, then $x^1, x^2, ..., x^{n-1}$ are all different from x and $x^n = x$.

Intuitively, you should think of g_{key} as defining a "cycle" between numbers 0,1,...,$n-1$. For example, if $n = 5$ one possible g is $g(0) = 3, g(3) = 2, g(2) = 4, g(4) = 1, g(1) = 0$.

The insert and find operations with a key key will first look at the location $x = f(key)$. If that location is not empty, the algorithm will continue to use function g_{key} to find the next location ($x = g_{key}(x)$) until it either finds the element or find an empty space for insertion. The implementations are given below:

```
A[0..n-1] is the hash table with (key,value) pairs

Insert(key, value)
    x = f(key)
    WHILE A[x] is not empty and A[x].key != key
       x = g(key, x)
    A[x].key = key
    A[x].value = value

Find(key)
    x = f(key)
    WHILE A[x] is not empty and A[x].key != key
       x = g(key, x)
    IF A[x] is empty THEN
        return "not found."
    ELSE
        return A[x].value
```

Suppose now the hash table contains αn elements (where α is a known fraction between $(0, 1)$), and we are inserting a random new element key such that $\Pr[f(key) = i] = 1/n$ for all $i = 0, 1, ..., n-1$. Suppose g is a uniformly random cycle.

(a) (20 points) Show that the expected number of cells ($A[x]$) that $INSERT$ operation looks at is at most $1/(1-\alpha)$.

(Hint: If a random variable X has value i with probability $p(1-p)^{i-1}$, then the expected value of this random variable is $1/p$.)

(b) (5 points) Show that the probability that $INSERT$ operation looks at more than $2/(1-\alpha)$ cells is at most $1/2$.