COMPSCI330 Design and Analysis of Algorithms
Assignment 4

Due Date: Tuesday, Nov 15, 2016

Guidelines

- **Describing Algorithms** If you are asked to provide an algorithm, you should clearly define each step of the procedure, establish its correctness, 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 (Hearthstone Skill)**.** Alice is playing hearthstone, a card game that requires a lot of "skills". In particular, she needs to be able to compute probabilities and expectations.

In hearthstone, each player has a hero with 30 HP, and they combat by minions and spells. Suppose Alice's opponent has a Faerie Dragon which has 2 HP. Alice wants to kill the Faerie Dragon by playing Arcane Missiles. Arcane Missiles will fire 3 missiles, each missile will attack a random character for 1 HP (that is, initially it has a 50% chance of hitting either the opponent hero or the Faerie Dragon; but if the Faerie Dragon took two hits from the first two missiles, the third missile can only hit opponent hero). Let X be a random variable that is 1 if the first missile hits the Faerie Dragon, and 0 otherwise. Let Y be a random variable that is equal to the amount of damage Arcane Missiles dealt to the Faerie Dragon.

(a) (5 points) Compute $\mathbb{E}[Y|X = 1]$ and $\mathbb{E}[Y|X = 0]$. Then compute $\mathbb{E}[Y]$ using law of total expectations.

(b) (5 points) What is the probability that Arcane Missiles kill the Faerie Dragon (the Faerie Dragon dies as soon as it is hit by two missiles)?

(c) (10 points) If Arcane Missiles fire $k(3 < k \leq 29)$ missiles instead of 3, what is the probability that it kills the Faerie Dragon?

Problem 2 (Expensive Counters)**.** (20 points) In order to represent large integers, we can store them in a binary array. As an example, the number 11 can be represented by an array $A[] = [1, 1, 0, 1]$. The first element in the array represents the least significant bit, and in general the k-th element in the array represents 2^{k-1}. The array $A[]$ is a representation for 11 because $11 = 1+2+8$.

You are implementing a binary counter on a strange hardware. Your counter consists of n binary bits. The counter starts at 0, and has a function called "increase". If the counter is representing number x now, after calling "increase" it should represent the number $x + 1$ (counter resets to 0 if x becomes 2^n).

The hardware is very strange, so flipping the k-th bit in the counter actually takes 2^k time. Show that the amortized cost for the increase operation is $O(n)$.

(Hint: Look at the recitation material after the midterm to see how to solve the problem when flipping the counter takes only 1 unit of time.)

Problem 3 (Waiting in lines)**.** (20 points) Suppose there are n people that are trying to form lines to enter a basketball game. Initially, each person is in his/her own line. After that, two kinds of operations might happen: 1. Two lines might merge into one line. 2. A person enters the stadium, and is therefore not in any of the lines (and he/she will not come back and join any other lines).

You are now asked to design a data structure that can keep track of how many people are in each line. Your data structure should support the following operations:

- Merge(A, B). Merge the two lines where A and B are in. (A and B are not already in stadium, and they are guaranteed to be in different lines.)
- Enter(A). Person A enters the stadium.
- Count(A). Output the number of people in the same line with A. Person A is not already in the stadium.

As an example, after Merge(A, B), Merge(B, C), Enter(B), A and C will be in the same line. In this case Count(A) should return 2. Count(D) should return 1 because D has not joined the line with anyone else.

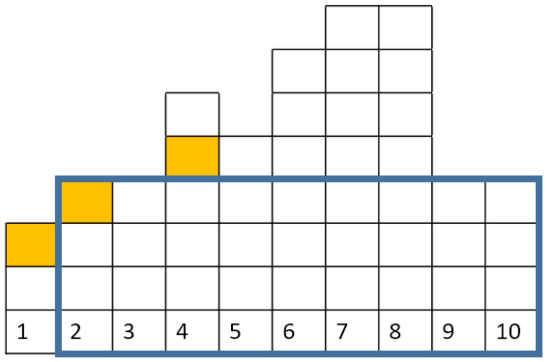
The amortized cost for all the operations should be $O(\alpha(n))$ where α is the inverse Ackermann function.

Problem 4 (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.

Problem 5 (Largest Rectangle)**.** (25 points) Bob got a piece of wood with a strange shape (see the figure below), and he wants to cut a largest rectangle out of it.



As you can see from the figure, one side of the wood is straight. The rectangle Bob wants must have one edge on this side of the wood. We can divide this side into n equal size intervals (in the figure $n = 10$). For each interval, we also know the height of the tree $h[i]$ (in the figure, $h[] = [3, 4, 4, 6, 5, 7, 8, 8, 4, 4]$).

In the figure, the largest rectangle is the blue rectangle, with an area of 36. Given n and $h[]$, you need to design an algorithm that finds the largest rectangle that Bob can get. Your algorithm should run in $O(n)$ time.

(Hint: You can sweep from left to right for the rightmost boundary of the rectangle. At each time step you need to keep track of possible upper-left corners. In the figure, the yellow squares are possible upper-left corners if the rightmost boundary is between 5 and 6.)