CS 331: Algorithms and Complexity (Fall 2016)
Unique numbers: 51420, 51425, 51430, 51435

Assignment 1

Due on Wednesday, September 7, by 11.59pm

Matt Gmitro 51430 MTG759

Problem 1

(a) (5 points) Arrange the following list of functions in increasing order of growth rate. That is, if function $g(n)$ follows function $f(n)$, then it should be the case that $f(n)$ is $O(g(n))$ (the base of logarithms is 2). Justify.

$$f_1(n) = 2^{2^n}$$
$$f_2(n) = 2^{n^3}$$
$$f_3(n) = n^{\log n}$$
$$f_4(n) = n(\log n)^3$$
$$f_5(n) = n^4$$
$$f_6(n) = 2^{2^{\log(\log n)}}$$
$$f_7(n) = \sqrt{n}$$

Answer: 7, 4, 6, 5, 3, 2, 1. 7 is a square root of n which is always less than n unlike all the others. 4 will grow slower than multiplying n by itself or any exponential. 6 grows slower than 3 because log(log n) has extremely slow growth and the n will grow faster. 5 goes before 3 because after n = 10000, log n = 4, which means 5 will exceed 3. 3 before 2 because 2 is an exponential with cubic growth in the exponent. And 2 before 1 because 1 is an exponent with exponential growth in the exponent.

(b) (5 points) Prove the following formula using induction

$$\sum_{r=1}^{n} r(r+1) = n(n+1)(n+2)/3.$$

Answer: First, the base case. If n = 1, then the left hand side gives 2. The right hand side gives 1(2)(3)/3 = 2. Thus, the base case is true. For the recursive case, we have

$$\sum_{r=1}^{k} r(r+1) = k(k+1)(k+2)/3,$$

and we can add (k+1)(k+2) to both sides. Then

$$\sum_{r=1}^{k+1} r(r+1) = (k(k+1)(k+2)/3) + (k+1)(k+2),$$

$$= (3(k+1)(k+2) + k(k+1)(k+2))/3,$$

$$= (k+1)(k+2)(k+3)/3,$$

since we can factor out (k+1)(k+2). This proves the recursive case, and by induction proves the formula.

(c) (5 points) Alice and Bob, two students of CS 331, were asked to provide algorithms for a simple program. The program is supposed to receive a stream of integers as input. After each input integer, the program must report the average of the integers it has seen so far. Alice and Bob came up with the following algorithms to do this simple task, as in Algorithm 1 and 2, respectively.

Algorithm 1: Alice's Algorithm

```
Initially sum is 0 and count of integers read is 0;
while the input stream is not empty do
    Read the integer i;
    Add i to the sum;
    Report average as sum/count;
    Increase count by 1;
```

Algorithm 2: Bob's Algorithm

```
Initially average is 0 and count of integers read is 0;
while the input stream is not empty do
    Read the integer i;
    Modify average as (average * count + i)/(count + 1);
    Increase count by 1;
    Report average;
```

State whether Alice and Bob have provided correct algorithms, ignoring truncation due to integer division. Also, either prove the algorithm to be right, or show how it is wrong. Proof

should mathematically verify the correctness of the steps and invariants, and incorrect steps in the algorithms should be mentioned with rectifications.

Answer: A.) Alice's algorithm is incorrect because she initializes count to 0 and divides by count on the first run before incrementing count. This can be rectified by incrementing count before calculating the average or by initializing count at 1.

B.) Bob's algorithm is correct. Proof: first, the base cares where the first integer i comes in. Since average and count are initialized at 0, the modified average is i / 1 which is the correct average for a single integer. Assuming the inductive hypothesis that the average is correct for the kth integer we get kaverage = (average * k + i) / (k + 1). To get the average of this we would multiply the average by k to get the sum of the first k integers, add k+1, and divide by a newly iterated count. So it's sufficient to show,

$$(\sum_{j=0}^{k+1} i_j)/(k+1) = ((kaverage) * k + (nexti))/(k+1+1),$$

and since this is the definition of an average, it is true. Therefore, by induction we conclude the algorithm is correct.

Problem 2

You are seated on an electric chair, and there is a two-pan balance before you. A two-pan balance is something that is used to compare the weight of two objects. It's one of the things Lady justice has. On the right pan, is an object of unknown weight W. The left pan is empty. You are given N tokens and all of these tokens have weight w. Using these N tokens and the two-pan balance, you have to approximate W as efficiently as possible. I.e. Provide an algorithm to find a and b such that $a \leqslant W \leqslant b$, $b - a = w$, and a and b are in terms of w (If W is heavier than Nw, then report that). There is a catch, however. Below the left pan is a switch. If the weight of the tokens in the left pan is greater than W, the switch will be hit, and you will be electrocuted to death. Moreover, each time you wish to change the number of tokens on the left pan, the left pan must be emptied before the desired number of tokens is placed on the left pan. I.e. you are not allowed to add or remove tokens from the left pan one by one. Given this scenario, approximate W when:

- You have one life to spare.
- You have an unlimited number of lives to spare. ("God I wish I was this person.")
- You have two lives to spare.

Note that in this problem, an algorithm A is said to be more efficient than another algorithm B if and only if the number of steps taken in the worst case scenario for A is smaller than the number of steps taken in the worst case scenario for B.

Problem 2 continued on next page...

Hint: For this question, you don't need to come up with a pseudo code. You just need to precisely describe your approach for each part in a small paragraph.

Answer:
a.) Add one coin on each weighing. If you die, you know that a = number before death and b = number at death with w = 1.
b.) Let a = half of the tokens, and start by weighing W against a. (i) If you die, take b = a, a = (1/2)a and try again. Repeat until you are alive, then you have a = most recent weighing and b = previous weighing. From here you can halve the distance between a and b and try adding that amount to a. If you die take this new value to be b and repeat until the width w = b - a is solidified. (ii) If you don't die immediately, go the opposite way and take a = (1.5)a. Once you die, set b = a, a = (a/1.5) and find the width in a similar manner as in (i).
c.) Let a = (1/4) of the tokens and weigh W against a. (i) If you die, iterate from 1 until you die again and you'll have the a = iteration before death, b = iteration at death. (ii) If you don't die, let a = half of the tokens and repeat the process in (i). If you don't die in (ii) repeat with a = (3/4) of the tokens and so on with a = all tokens. You will only need to sort through a quarter of the tokens max.