UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL 2014 EXAMINATIONS

CSC236H1S

Instructor: David Liu

Duration – 3 hours
Total Marks: 74

Examination Aids: one *double*-sided, *handwritten* aid sheet

Last Name:

First Name:

Student Number:

Please read the following guidelines carefully!

1. This examination has **9** questions. There are a total of **17** pages, not including this title page. Blank pages are included at the end of the exam for rough work.
2. Answer questions clearly and completely. Give formal proofs unless explicitly asked not to. You may use any claim/result from class, unless you are being asked to prove that claim/result, or explicitly told not to.
3. Any question you leave blank or cross out your work and write "I don't know" is worth 10% of the marks.
4. **You must achieve a mark of at least 40% on this exam to pass this course.**

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Total
Grade										
Out Of	6	7	6	12	6	12	13	5	7	74

Take a deep breath.

This is your chance to show us

How much you've learned.

We **WANT** to give you the credit

That you've earned.

A number does not define you.

It's been a real pleasure teaching you.

Good luck!

1. [6] You might find the following recursively-defined function $g : \mathbb{N} \to \mathbb{N}$ familiar:

$$g(n) = \begin{cases} 0, & \text{if } n \leq 1 \\ g(n-1) + g(n-2) + 2^{n-2}, & \text{if } n > 1 \end{cases}$$

Prove using complete induction that for all $n \in \mathbb{N}$, $g(n) = 2^n - f_{n+2}$, where f_n denotes the n-th Fibonacci number: $f_1 = f_2 = 1$, and for all $n \geq 3$, $f_n = f_{n-1} + f_{n-2}$.

2. Consider the following mysterious divide-and-conquer algorithm.

```
1 def mys(A, x):
2   if len(A) <= 1:
3     if A[0] < x:
4       return 1
5     else:
6       return 0
7   else:
8     c = len(A) // 3 # Integer division, rounds down. Equivalent to floor(len(A)/3).
9     return mys(A[0..c-1], x) + mys(A[c..2*c-1]) + mys(A[2*c..len(A)-1])
```

(a) [5] Assume that the length of A is a power of three. Find a recursive definition for the worst-case runtime of this function (justifying it based on the code), and find a tight asymptotic bound for your recurrence using the Master Theorem.

(b) [2] In fact, if $len(A)$ is positive but *not* a power of three, this function is not guaranteed to terminate! Explain why. (Hint: take a small non-empty list of length that isn't a power of 3, and examine the recursive calls very carefully.)

3. Consider the function defined recursively below:

$$T(n) = \begin{cases} 10, & \text{if } n = 0 \\ T(n-1) + 4 \cdot 3^n, & \text{if } n \geq 1 \end{cases}$$

(a) [2] Give *two* reasons you cannot use the Master Theorem here.

(b) [4] Find an exact closed form for $T(n)$ using repeated substitution (no proof necessary). You may find the following formula helpful:

$$\sum_{i=0}^{n} x \cdot y^i = x \cdot \frac{y^{n+1} - 1}{y - 1}.$$

4. Consider the following function.

```
1 def someop(n, d):
2 '''
3 Pre: n, d are natural numbers, with d > 0
4 Post: ???
5 '''
6   a = 0
7   b = n
8   while b >=d:
9     b -= d
10    a += 1
11  return (a,b)
```

(a) [2] State a postcondition for this function that fully describes what it does (so writing something simple like "returns two natural numbers" receives no marks).

(b) [4] Prove that the `while` loop terminates by finding a suitable loop measure. Note that you might need to find and prove a suitable loop invariant, depending on how you choose your loop measure.

(c) [6] Prove that this function is correct, according to the given precondition, and your postcondition from part (a). Note that you will need find and prove a good loop invariant in order to do this.

5. [6] Regular languages are built by taking the fundamental unit languages $\emptyset$, $\{\epsilon\}$, and $\{a\}$ ($a \in \Sigma$) and repeatedly applying union, concatenation, and star operations. You might wonder – are all three of these operations necessary in defining regular languages? In this question, we'll prove that all three are, in fact, necessary. Define the following three sets of regular expressions:

$$R_+ = \{\text{regexes that do not use the } + \text{ operator}\}$$
$$R_\times = \{\text{regexes that do not use concatenation}\}$$
$$R_* = \{\text{regexes that do not use the } * \text{ operator}\}$$

E.g., the regular expression $01 + \epsilon + 1$ is in R_*, but not R_+ or $R_\times$.

For *each* of the three sets R_+, $R_\times$, and R_*, give a regular language that is *not* matched by any regex in the set, and explain why. You must also justify why each of the languages you used are indeed regular.

6. For each of the following regular languages, (i) give a regular expression matching the language, and (ii) give a DFA or NFA accepting the language. You do not need to provide justification for your answers, but if your answer is incorrect, you are more likely to receive part marks if we see evidence of your thinking.

   (a) [4] $L = \{w \in \{a, b\}^* \mid w \text{ starts with } a \text{ and ends with } bb\}$

   (b) [4] $L = \{w \in \{0, 1\}^* \mid w \text{ contains exactly two 0's}\}$

   (c) [4] $L = \{w \in \{a, b, c\}^* \mid w \text{ contains at least two occurrences of the string } abc\}$

7. Consider the language $L = \{w \in \{1,2\}^* \mid$ the sum of the digits in w is one less than a multiple of 4$\}$. Note that $\epsilon \notin L$, as we'll assume the sum of the digits in ϵ is 0.

   (a) [4] As a warmup, prove that every DFA accepting L must contain at least 4 states. (Hint: consider the possible remainders when you divide a number by 4.)

(b) [2] Design a 4-state DFA accepting this language, and **label each of your states** to refer to in part (c).

(c) [4] Give state invariants for your DFA.

(d) [3] Consider the initial state of your DFA. Prove that ϵ satisfies the invariant you gave for the initial state, and then prove that the two transitions out of the initial state respect your state invariants.

8. [5] Let $L = \{0^{n^2} \mid n \in \mathbb{N}\}$; for example, $\epsilon = 0^0, 0 = 0^1, 0000 = 0^4, 000000000 = 0^9$ are all in L.
   Prove that L is not regular.
   Hint: compute the difference between consecutive perfect squares, $(n+1)^2 - n^2$.

9. Consider the following problem. You are given a set of n coins, all of which are identical except one. This one fake coin looks the same as the others, but weighs different (either lighter or heavier, you don't know). You have a balance that can compare two (arbitrary size) sets of coins and tell you which is heavier (or tell you they are both the same weight).

   (a) [4] Assume $n = 3^k$, i.e., a power of 3. Give a divide-and-conquer algorithm that will let you identify the fake coin using as few weighings as possible (though you don't need to prove this). Feel free to use only English to describe your steps, but also to use "pseudocode"-like structure for things like if statements. Be explicit in where you use recursion.

   Hint: The only operations you can do with the coins are divide them into sets, and use the balance to compare the weights to two different sets.

(b) [2] Give a recurrence for the *exact* number of weighings (times you use the balance), in the **worst case**.

(c) [1] Suppose you are asked to calculate a closed form expression for the *exact* number of weighings your algorithm uses in the worst case. Why can you not use the Master Theorem?

Use this page for rough work.

Use this page for rough work.

Use this page for rough work.

Use this page for rough work.