NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING
EXAMINATION FOR
Semester 2, 2010/2011

CS 5201 - FOUNDATION IN THEORETICAL CS

April/May 2011 Time Allowed: 3 Hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **four**(**4**) long questions and comprises **seven** (**7**) pages, including this page.
2. *Answer* **three** *out of* **four** *questions.*
3. Each question should be answered in a **separate** answer book.
4. This is an *OPEN BOOK* examination.
5. Write your Matriculation number in **all** the answer books.

A: Algorithm ((5+10+10) = 25 marks)

Problem 1. AlgoQueues: "A better queue, or your money back." [5 points]

You've just started working for *AlgoQueues LTD*, a company specializing in fast algorithms. You've just begun work, and your boss says that your first task is to design a better priority queue. Recently, due to an amazing breakthrough, an engineer at AlgoQueues developed a new *QuantumString* microprocessor which can compare up to k integers in one step. The special MULTIC operation takes an array of k integers as input, and produces a sorted array of k integers as an output in just one step. Your boss wants you to build a priority queue, designed to take advantage of the MULTISC operation, with the following properties:

- Given an (unsorted) array A of n elements, you can build a priority queue in $O(n/k)$ steps.
- The `kExtractMin` operation returns the k smallest elements in the queue and runs in $O(\log \log n)$ steps.
- The `insertKey` operation adds a new element to the queue and runs in $O(\log n)$ steps.

Moreover, the new priority queue should be comparison-based, since it will be used to store arbitrary objects (like strings). Explain (briefly) why your boss's request is impossible, even when executing on the special QuantumString microprocessor, as long as $k = o(n)$.

Problem 2. Spinning in Circles [10 points]

(a) Assume you are given a connected, undirected graph $G = (V, E)$ as an adjacency list. The graph G contains n nodes and m edges. Give an algorithm (in pseudocode) that determines whether G has a cycle and runs in $O(n)$ time. Explain why your algorithm is correct. Note that. for full credit, the running time of your algorithm should not depend on m. (*Hint:* you do not necessarily need to find the cycle, if it exists.)

(b) Assume you are given a connected, undirected graph $G = (V, E)$ as an adjacency list. The graph G contains n nodes and m edges. For any two nodes $v, w \in V$, we define the distance $d(v, w)$ to be the length of the shortest path between v and w. We define the diameter of G to be the maximum distance between any two nodes in G. Assume that the diameter of G is $> n/2$. Prove that there exists some node $v \in V$ such that if you remove v from G, the graph is no longer connected, i.e., it is divided into (at least) two disconnected components.

Problem 3. Favorite Photos [10 points]

Your job is to build a data structure to maintain a set of photographs. Your photograph database should allow you to insert and search for photographs, as well as to designate some of the pho-

tographs as favorites by *marking* them. In more detail, your data structure should support the following operations:

- `Insert(x, t, m)`: inserts photograph x that was taken at time t. If $m = 1$, then the photograph is marked as a favorite; if $m = 0$, then it is not a favorite.
- `Search(t)`: find the next photograph taken after time t.
- `NextFavorite(t)`: find the next photograph taken after time t that is a favorite.

Give a data structure for solving this problem, and explain how it works. For full credit, your data structure should be both time and space efficient: more points will be given for more efficient solutions. (Remember that the photographs themselves may be quite large.) Give the performance (i.e., running time) of each operation and the space usage of the data structure. You may assume that at any given time t, your camera has taken at most one photograph x.

Describe your solution in words, or very high-level pseudocode. You do not need to provide complete details. You must provide enough detail, however, that your solution is clear.

B: Theory of Computation (10 Marks: 3+3+4)

(a) Let $|x|$ denote the length of a word x, xy be the concatenation of the strings x and y and consider for any two regular sets L, H the set $L \odot H$ given as

$$L \odot H = \{xy : |x| = |y| \wedge x \in L \wedge y \in H\}.$$

Which of the following three statements is true? Prove your answer.

1. For every regular sets L, H, the set $L \odot H$ is also regular.

2. For every regular sets L, H, the set $L \odot H$ is context-free; however, there are regular sets L, H such that $L \odot H$ is not regular.

3. There are regular sets L, H such that $L \odot H$ is not context-free.

(b) Give as a graphic or as a table a deterministic finite automaton which accepts the following language:

$$(\{0,1\}^* \cap (\{1\} \cdot \{0,2\}^* \cdot \{11,111\})) \cup (\{2,22,222,2222\} \cdot \{00,11\}^*).$$

(note that '$\cdot$' denote cancatenation)

Please mark the starting state and say which states accept or reject. For each state q and each symbol a there must be at most one state p such that the automaton can go from q to p on a.

(c) Let $\varphi_0, \varphi_1, \ldots$ be an acceptable numbering (= Gödel numbering) of all partial-recursive functions. Note that $\varphi_e(x)$ is undefined iff the e-th machine on input x does not halt. For the following sets A and B, state the choice (1, 2, or 3) for A and B:

1. decidable (= recursive),
2. recursively enumerable and undecidable,
3. not recursively enumerable?

Justify your answer for both sets A and B.

$A = \{e : 0 \notin \text{range}(\varphi_e)\}$;

$B = \{e : \varphi_e(e^2) \text{ is defined}\}$.

C: Principles of Programming Languages (20 marks)

(a) Consider the following polymorphic list data declaration:

$$data\ List\ a\ =\ Nil\ |\ Cons\ a\ (List\ a)$$

Give the most general types for each of the following functions. If a type error occurs, discuss the cause(s) of the type error. (6 marks)

(i) $list1\ =\ Cons\ 1\ (Cons\ 2\ (Cons\ 3\ Nil))$

(ii) $list2\ =\ Cons\ 'a'\ (Cons\ 'b'\ Nil)$

(iii) $list3\ =\ Cons\ (Cons\ 'a'\ Nil)\ (Cons\ (Cons\ 3\ Nil)\ Nil)$

(iv) $fn1\ xs\ y\ =$ case xs of $\{\ Cons\ x\ xs\ \rightarrow\ Cons\ y\ (Cons\ x\ Nil)\ \}$

(v) $fn2\ xs\ r\ f\ =$ case xs of $\{\ Nil \rightarrow r\ ;\ Cons\ x\ xs \rightarrow\ f\ x\ xs\ \}$

iv $fn3\ g\ x\ =\ g\ x\ (fn3\ g\ x)$

(b) Consider a generic higher-order function that is defined as follows:

```
// type declaration
goo :: (a → r → r) → (a → r → r) → r → (List a) → r

// method declaration
goo f g w xs = case xs of { Nil       → w;
                            Cons a as → f a (goo f g (g a w) as) }
```

(i) Consider the following function to compute the length of a list. (2 marks)

```
len :: List a → Int
len xs = case xs of { Nil       → 0;
                      Cons a as → 1 + (len as) }
```

Both *len* and *goo* share a similar algorithmic structure. Show that *len* is a special instance of *goo* by implementing the former in terms of the latter.

(ii) With the use of the *goo* method, provide a function that would reverse a list of elements in *linear time*. This function should have the following features. (2 marks)

$reverse :: List\ a \rightarrow List\ a$
e.g. $reverse\ (Cons\ 'h'\ (Cons\ 'i'\ Nil))$
$\Rightarrow$ (evaluates to) $(Cons\ 'i'\ (Cons\ 'h'\ Nil))$

(iii) Using *goo*, implement a method that would find the second largest number in a given list. In case such a number cannot be found, you may throw an exception. (3 marks)

$secondmax :: List\ Int \rightarrow Int$
e.g. $secondmax\ (Cons1\ (Cons\ 2\ Nil)) \Rightarrow 1$
$secondmax\ (Cons\ 1\ Nil) \Rightarrow$ ..exception..

(c) Consider the following definition to join two lists together

$$
\begin{array}{l}
app :: List\ a \rightarrow List\ a \rightarrow List\ a \\
app\ xs\ ys = \text{case } xs \text{ of } \{\ Nil \rightarrow ys; \\
\qquad\qquad\qquad\quad Cons\ a\ as \rightarrow Cons\ a\ (app\ as\ ys)\ \}
\end{array}
$$

Prove (by induction) that the following propery holds. (3 marks)

$$app\ (app\ xs\ ys)\ zs \equiv app\ xs\ (app\ ys\ zs)$$

(d) Briefly, *compare* and *contrast*, the following parameter passing mechanisms. In each instance, describe one of more languages that support the corresponding mechanisms. (4 marks)

- Pass by value.
- Pass by reference.
- Pass by value and result.
- Pass by name.

D: Logic and AI (45 marks)

1 (Propositional logic; 5 + 5 + 5 = 15 marks)

a Determine whether or not the following formula is valid. Use the truth table or the semantic tableau method to do so:
$(p \supset q) \supset (q \supset p)$

b Show that the following inference rule is sound.

$$\frac{\vdash \sim (A \wedge \sim B) \quad \vdash \sim (B \wedge \sim A)}{\vdash A \equiv B}$$

c Give an example of a formula A in which (i) the atomic propositions p, q and r occur at least once each and (ii) A is in Conjunctive Normal Form (CNF).

2 (First order logic; 10 + 5 + 10 = 25 marks)

a In the formula given below underline or circle all the free occurrences of the variables. Which variables, if any, have both a free and bound occurrence? Which variables, if any, have only free occurrences?
$\exists x[P(y,z) \wedge (\forall y[\sim Q(y,x) \vee P(y,z)])]$

b Show that the following closed formula φ is satisfiable by exhibiting a model for it.

$\varphi = \varphi_1 \wedge \varphi_2, \wedge \varphi_3$ where:
- $\varphi_1 = \forall x \sim S(x,x)$
- $\varphi_2 = \forall x \exists y S(x,y)$
- $\varphi_3 = \forall x \forall y \exists z (S(x,y) \supset (S(x,z) \wedge S(z,y))$

c Determine if the following two atomic formulas are unifiable. If yes, compute an mgu for the pair. If not, show where the unification algorithm will fail.
$p(a, x, f(g(y)))$, $p(z, h(z,u), f(u))$

3 (Program logics; 1 + 4 = 5 marks)
In answering this question please ensure that your answers are *succinct*.

a What is the case for attempting the formal verification of programs?

b Comment on the pros and cons of using the following logical systems to reason about the behavior of programs (say, written in C).
- Propositional logic.
- First order logic.
- (Floyd-) Hoare logic.
- Temporal logics.