University of Toronto, Faculty of Arts and Science
August 2014 Examinations
CSC373H1Y Final Examinations
Lalla Mouatadid
Duration: 3 hours
No Aids Allowed
YOU MUST GET AT LEAST 40% ON THIS EXAM IN ORDER TO PASS THE COURSE

PLEASE COMPLETE THE SECTION BELOW:

First Name: ______________________

Last Name: ______________________

Exam Instructions

- **Check that your exam book has 20 pages** (including this cover page and 3 blank pages at the end). The last 3 pages are for rough work only, *they will* **not** *be marked*. Please bring any discrepancy to the attention of an invigilator.
- There are 6 questions worth a total of 104 points. Answer all questions on the question booklet.
- For Questions C, D, E, and F, if you do not know how to answer any part of the question then you can leave that part blank or write "I DON'T KNOW." to receive 10 percent of the points of the question.

Course Specific Notes

- Unless stated otherwise, you can use the standard data structures and algorithms discussed in CSC263 and in the lectures without describing their implementation by simply stating their standard name (e.g. min-heap, merge- sort, DFS, Dijkstra). Suggest a one must-do activity in Toronto for one extra bonus mark. You do not need to provide any explanation or pseudo-code for the data structures implementation. You can also use their running time without proof. For example, if you are using the merge-sort in your algorithm you can simply state that merge-sort's worst-case running time is $\mathcal{O}(n \log n)$. If you modify a data structure or an algorithm from class, you must describe the modification and its effects.
- In some questions you will be given a computational problem and then asked to design an efficient algorithm for it. Unless stated otherwise, for data structures and algorithms that you design you should provide a short high-level explanation of how your algorithm works in plain English, and the pseudo-code for your algorithm in a style similar to those we have seen in the lectures. If you miss any of these the answer might not be marked. Your answers will be marked based on the efficiency of your algorithms and the clarity of your explanations. State the running time of your algorithm with a brief argument supporting your claim and prove that your algorithm works correctly (e.g. finds an optimal solution).

PLEASE PRINT YOUR STUDENT NUMBER AND YOUR NAME

Student Number: ______________________

First Name: ______________________

Last Name: ______________________

The section below is for marker's use only. Do NOT use it for answering or as scratch paper.

Questions	**Points**
A	/12
B	/12
C	/20
D	/20
E	/20
F	/20
Total	/104

A. Definitions [12]

Give the inputs and outputs of the following problems.

1. **Minimum Vertex Cover** [2]
   Input:

   Output:

2. **Circuit SAT** [2]
   Input:

   Output:

3. **Maximum Bipartite Matching** [2]
   Input:

   Output:

Give a formal definition for each of the following topics/mathematical objects that we examined in complexity theory.

1. **A verifier** [2]

2. **A decision problem** [2]

3. **NP** [2]

B. Short'ish Answers [12]

1. Let G(V, E) be a graph. Show that G has a vertex cover of size k if and only if G has an independent set of size $n - k$ where $n = |V|$. [3]

2. Point out and discuss the fallacy in the following "proof" that P $\neq$ NP: "To see if a 3-SAT formula is satisfiable, we need to look at 2^n possible truth assignments. This takes exponential time, so 3-SAT is not in P. But it is in NP, so P $\neq$ NP." [3]

3. Consider an undirected graph $G = (V, E)$ with nonnegative edge weights $w_e \geq 0$. Suppose that you have computed a minimum spanning tree of G, and that you have also computed shortest paths to all nodes from a particular node $s \in V$.
   Now suppose each edge weight is increased by 1: the new weights are $w'_e = w_e + 1$.
   (a) Does the minimum spanning tree change? Give an example where it changes or prove it cannot change. [3]

   (b) Do the shortest paths change? Give an example where they change or prove they cannot change. [3]

C. Student Unions are always cheap.. [20]

You are the manager of UTSU and are in charge of a group of n students, each of whom is scheduled to work one *shift* during the week. There are different jobs associated with these shifts (tending the main desk, helping with package delivery, rebooting cranky information kiosks, etc.), but we can view each shift as a single contiguous interval of time. There can be multiple shifts going on at once.

You are trying to choose a subset of these n students to form a *supervising committee* that you can meet with once a week. You consider such a committee to be *complete* if, for every student not on the committee, that student's shift overlaps (at least partially) the shift of some student who is on the committee. In this way, each student's performance can be observed by at least one person who's serving on the committee.

Given the schedule of n shifts, you want to produce a complete supervising committee containing as *few* students as possible.

Example: Suppose $n = 3$, and the shifts are:
Monday: 4 PM - 8PM
Monday: 6 PM - 10 PM
Monday 9 PM - 11 PM

Then the smallest complete supervising committee would consist of just the second student, since the second shift overlaps both the first and the third.

(a) Give an efficient algorithm to solve this problem. Be sure to give a brief high-level description of your algorithm, a pseudocode implementation, and an argument about the running time. [10]
Hint: How can an EFT sorting help?

(b) Prove that your algorithm is optimal [10].

D. It was the best of times, it was the worst of times. [20]

You are given a string of n characters $s[1...n]$, which you believe to be a corrupted text document in which all punctuation has vanished (so that it looks something like "itwasthebestoftimes..."). You wish to reconstruct the document using a dictionary, which is available in the form of a Boolean function $dict(\cdot)$: For any string w:

$$dict(w) = \begin{cases} true & \text{if } w \text{ is a valid word} \\ false & \text{otherwise} \end{cases}$$

Give a dynamic programming algorithm that determines whether the string $s[\cdot]$ can be reconstituted as a sequence of valid words. The running time should be at most $\mathcal{O}(n^2)$, assuming calls to $dict(\cdot)$ take unit time.

(a) Give a high-level description of each cell in the recurrence used by your algorithm. [3]

(b) Formally define the recurrence relation for the recurrence you gave in Part (a). [4]

(c) Give a dynamic programming algorithm implementing the recurrence relation. Be sure to give a brief high-level description of your algorithm, a pseudocode implementation, and an argument about the running time. [6]

(d) Prove that the recurrence relation you defined is correct. [7]

E. Monotone Satisfiability with Few True Variables [20]

Consider an instance of the Satisfiability problem, specified by clauses $C_1, ...C_k$ over a set of Boolean variables $x_1...x_n$. We say that the instance is *monotone* if each term in each clause consists of a nonnegated variable; that is, each term is equal to x_i, for some i, rather than $\overline{x_i}$. Monotone instances of Satisfiability are very each to solve: They are always satisfiable, by setting each variable equal to 1. For example, suppose we have the three clauses

$$(x_1 \vee x_2), (x_1 \vee x_3), (x_2 \vee x_3)$$

This is monotone, and indeed the assignment that sets all three variables to 1 satisfies all the clauses. But we can observe that this is not the only satisfying assignment; we could also have set x_1 and x_2 to 1, and x_3 to 0. Indeed, for any monotone instance, it is natural to ask how few variables we need to set 1 in order to satisfy it.

Given a monotone instance of Satisfiability, together with a number k, the problem of *Monotone Satisfiability with Few True Variables* asks: Is there a satisfying assignment for the instance in which at most k variables are set to 1?

(a) Show that this problem is in NP. [6]

(b) Show that the problem is NP-hard by a reduction from one of the NP-Complete problems discussed in the lectures or tutorials. [14]

F. Job Scheduling [20]

Suppose we have n jobs each of which take time t_i to process and m identical machines on which to schedule their completion. Jobs cannot be split between machines. For a given scheduling, let A_j be the set of jobs assigned to machine j. Let $L_j = \sum_{i \in A_j} t_i$ be the load of machine j. The **Minimum Makespan Scheduling Problem** is to find an assignment of jobs to machines that minimizes the makespan, defined as the maximum load over all machines (i.e. $\max_j L_j$).

Consider the following greedy algorithm for this problem which sorts the jobs so that $t_1 \geq t_2 \geq \ldots \geq t_n$, and iteratively allocates the next job to the machine with the least load.

Algorithm 1 Greedy Approximation Algorithm for Job Scheduling

```
1: $A_j \leftarrow \emptyset, T_j \leftarrow 0, \forall j$
2: for i = 1 ... n do
3:     $j \leftarrow \min_k T_k$
4:     $A_j = A_j \cup \{i\}$
5:     $T_j = T_j + t_i$
6: end for
```

(a) Show that the algorithm is a $\frac{1}{2}$-approximation algorithm. [10]

(b) Give a more careful analysis that shows that the approximation factor is at most $\frac{2}{3}$. [10]
Hint: Which of the bounds in part (a) can you improve?

Blank.

Blank.

Blank.