PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL 2015 EXAMINATIONS

CSC 263 H1 S

Duration—3 hours

PLEASE HAND IN

Examination Aids: One *double*-sided *hand*written *8.5"×11"* aid sheet.

Student Number: |_|_|_|_|_|_|_|_|_|_|

Last (Family) Name(s): _____

First (Given) Name(s): _____

---

*Do **not** turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully.*

This final examination consists of 8 questions on 15 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the examination is complete and fill in the identification section above.*

Answer each question directly on the examination paper, in the space provided. Use one of the "blank" pages (at the end) for rough work. If you need more space for one of your solutions, use one of the "blank" pages and *indicate clearly the part of your work that should be marked.*

In your answers, you may use without proof any theorem or result covered in lectures, tutorials, problem sets, assignments, or the textbook, as long as you give a clear statement of the result(s)/ theorem(s) you are using. You must justify all other facts required for your solutions. Write up your solutions carefully! In particular, use notation and terminology correctly and explain what you are trying to do.

If you are unable to answer a question (or part of a question), you will get 20% of the marks for any solution that you leave *entirely blank* (or where you cross off everything you wrote to make it clear that it should not be marked).

*Remember that, in order to pass the course, you must obtain a grade of **at least 40%** on this final examination.*

MARKING GUIDE

N° 1:	_____/ 10
N° 2:	_____/ 10
N° 3:	_____/ 15
N° 4:	_____/ 10
N° 5:	_____/ 15
N° 6:	_____/ 15
N° 7:	_____/ 10
N° 8:	_____/ 15
TOTAL:	_____/100

Good Luck!

Question 1. [10 MARKS]

Consider a **max**-heap of four items with distinct keys k_1, k_2, k_3, k_4 where $k_1 < k_2 < k_3 < k_4$.

Part (a) [4 MARKS]

Draw all the different max-heaps that are possible for the elements described above.

Part (b) [6 MARKS]

Suppose that all the arrangements from the previous part are equally likely, and that we insert a random item whose key is equally likely to be in any order with respect to k_1, k_2, k_3, k_4 (in between any two of them or beyond either endpoint).

Calculate the average **number of swaps** that would take place for this one INSERT operation.

Question 2. [10 MARKS]

Part (a) [2 MARKS]

In the decision tree model, which property of the tree represents the wort-case running time of a *sorting* algorithm?

Part (b) [2 MARKS]

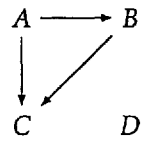
What does each leaf node represent in one of the decision trees from Part (a)?

Part (c) [6 MARKS]

Give a lower bound on the number of comparisons needed to determine the *two largest elements* (the order between the two does NOT matter) in an unsorted list with n elements? Briefly justify your answer.

Question 3. [15 MARKS]

The same graph can be represented by a number of different data structures. Consider the directed graph G pictured on the right. The same directed graph is represented in four different ways below (some of which we have studied and some others we have not).



Adjacency Matrix: Vertices $[A, B, C, D]$.

0	1	1	0
0	0	1	0
0	0	0	0
0	0	0	0

Regular Adjacency Lists: (Every vertex u has a list of the vertices v such that G contains an edge (u, v) *from* u *to* v.)

- $A : [B, C]$
- $B : [C]$
- $C : []$
- $D : []$

***Inverted* Adjacency Lists:** (Every vertex u has a list of the vertices v such that G contains an edge (v, u) *from* v *to* u.)

- $A : []$
- $B : [A]$
- $C : [A, B]$
- $D : []$

List of vertices and unsorted list of edges:

$V = [A, B, C, D]$

$E = [(A, B), (A, C), (B, C)]$

Below is a table listing five operations that could be performed on a graph. For each of the representations above, give the asymptotic worst-case complexity of the operation expressed in terms of $|V|$ and $|E|$. The first operation has been completed for you as an example.

For every representation, assume that it is possible to find the *size* of any list (or matrix) in $\mathcal{O}(1)$ time. You only need to put your final answer in the table—no justification is required.

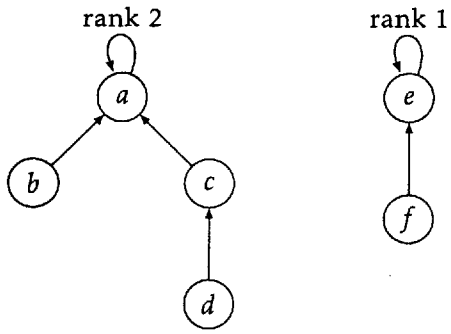
Operation	Adjacency Matrix	Regular Adjacency Lists	Inverted Adjacency Lists	List of Vertices and Edges
edge query	$\mathcal{O}(1)$	$\mathcal{O}(\min(\|V\|, \|E\|))$	$\mathcal{O}(\min(\|V\|, \|E\|))$	$\mathcal{O}(\|E\|)$
Determine if a path of length 2 exists from u to v				
Return a list of in-neighbours of vertex v				
Return a list of the self-loops				
Return the number of edges				

Question 4. [10 MARKS]

Suppose that Kruskal's algorithm is run on an undirected graph G, and that the disjoint-set ADT is implemented with a data structure that uses trees with union-by-rank and path-compression heuristics.

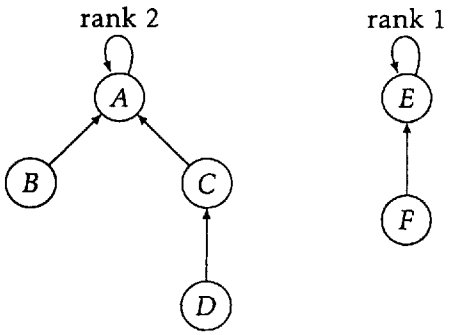
Each question below shows the state of the disjoint sets just *before* Kruskal's algorithm examines a certain edge $\{u,v\}$ to decide whether or not u and v are in the same connected component. Draw the disjoint sets that result *after* Kruskal's algorithm has examined the edge and added it to the partial tree (or not, if u and v were in the same connected component). *Remember to indicate the rank of each tree.*

Part (a) [2 MARKS]



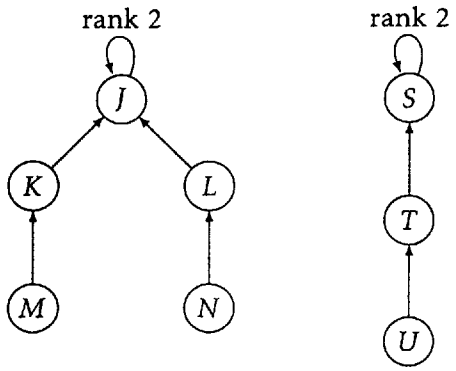
Examine edge $\{b,d\}$.

Part (b) [4 MARKS]



Examine edge $\{C,E\}$.

Part (c) [4 MARKS]



Examine edge $\{M,U\}$.

Question 5. [15 MARKS]

A vertex v in an undirected graph G is called a *celebrity* if v has an edge to *every* other vertex in G. Consider this algorithm for finding a celebrity vertex, if one exists. The input for the algorithm is an $n \times n$ adjacency matrix representing G. (Remember that there are **no** self-loops in an undirected graph.)

```
FindCelebrity(M):
    n ← M.size
    for i ← 1, 2, ..., n:
        edges ← 0
        for j ← 1, 2, ..., n:
            if M[i][j] = 1 :   # count only this operation
                edges ← edges + 1
        if edges = n − 1:
                return i
    return NIL
```

In this question, you will analyse the time complexity of algorithm FindCelebrity by counting *only* the number of accesses to the adjacency matrix (ignoring all other operations).

Part (a) [3 MARKS]

Assume *for only this subquestion* that $M.size = 4$. Give an adjacency matrix (for an undirected graph) that results in the **best**-case number of accesses.

Part (b) [1 MARK]

In the general case (where $M.size = n$), how many accesses are performed in the best-case?

Part (c) [2 MARKS]

What property is satisfied by an adjacency matrix M (of size $n \times n$) on which FindCelebrity performs the **worst**-case number of accesses?

Part (d) [2 MARKS]

In the general case (where $M.size = n$), how many accesses are performed in the worst-case?

Question 5. (CONTINUED)

For the remaining parts of this question, assume that FINDCELEBRITY is executed on a random graph with n vertices, where the probability of G containing any given edge $\{u,v\}$ is 0.5, for $u \neq v$, $1 \leqslant u,v \leqslant n$.

Part (e) [1 MARK]

What is the probability that the input to our algorithm is a best-case?

Part (f) [1 MARK]

What is the probability that an arbitrary vertex u is **not** a celebrity?

Part (g) [5 MARKS]

Compute the average-case number of accesses performed by FINDCELEBRITY on the random input graph specified above. *Do **not** simplify your expression.*

Question 6. [15 MARKS]

Consider the following INVENTORY ADT used to model the products sold on a shopping website.

Objects: Sets of *products*. Each product P consists of the following attributes: $P.code \in \mathbb{Z}^+$ (a **unique** positive integer *product code*) and $P.rating \in [0, 10]$ (a **real number** *rating* in the interval $[0, 10]$).

Operations:

- GETPRODUCT(S, C): return a product $P \in S$ such that $P.code = C$; return NIL if there is **no** product in S with product code C.
  Requirement: runs in average-case time $O(1)$.
- ADDPRODUCT(S, P): add product P to set S; precondition: S contains **no** other product with product code $P.code$—*do NOT check this condition, just assume it.*
  Requirement: runs in worst-case time $O(\log n)$.
- TOPRATED(S, r): return the **number** of products in set S with a rating $\geqslant r$.
  Requirement: runs in worst-case time $O(\log n)$—*part marks for* $O(n)$.

Part (a) [1 MARK]

What data structure would you use to achieve $O(1)$ average-case runtime for operation GETPRODUCT?

Part (b) [1 MARK]

Write an implementation for operation GETPRODUCT. State your assumptions clearly.

Part (c) [4 MARKS]

What data structure would you use to achieve $O(\log n)$ worst-case runtime for operation TOPRATED? Describe clearly what information is stored in each element of your data structure.

Question 6. (CONTINUED)

Part (d) [4 MARKS]

Write an implementation for operation ADDPRODUCT (in clear English). State your assumptions clearly.

Part (e) [5 MARKS]

Write a *detailed* implementation for operation TOPRATED, **in pseudo-code**. State your assumptions clearly.

Question 7. [10 MARKS]

Consider the following strategies to implement a Dynamic Array, on which we perform a sequence of m APPEND operations. For each strategy, give the amortized time **per APPEND operation** and indicate how much is *charged* for the i-th APPEND operation, for $1 \leqslant i \leqslant m$, to pay for all the costs (using the accounting method of amortized analysis). Assume that each array assignment takes 1 unit of time.

Part (a) [2 MARKS]

When the array is full, allocate a new array with *triple* the size and migrate all elements from the original array. **No detailed analysis required.**

- Amortized time per APPEND (asymptotic notation): _____
- Charge for i-th APPEND (exact value): _____

(Use the space below for rough work.)

Part (b) [2 MARKS]

When the array is full, allocate a new array whose size is 263 plus the size of the original array, then migrate all elements from the original array. **No detailed analysis required.**

- Amortized time per APPEND (asymptotic notation): _____
- Charge for i-th APPEND (exact value): _____

(Use the space below for rough work.)

Part (c) [6 MARKS]

On the next page, carry out a *detailed* amortized analysis of the following strategy: when the array is full, allocate a new array whose size is 1.5 times the size of the original array (rounded up), then migrate all elements from the original array before assigning the new element.
State clearly which analysis method you are using (aggregate or accounting) and *show your work.*

Question 7. (CONTINUED)

Part (c) (CONTINUED)

Question 8. [15 MARKS]

A *word ladder* is a sequence of English words, where two adjacent words differ by only one letter. For example:

COLD → CORD → CARD → WARD → WARM

You are given the two words "PHYSICS" and "GEOLOGY", and you must design an algorithm that computes a word ladder between the two words, with the **shortest possible** sequence of words.

Input: The two words "PHYSICS" and "GEOLOGY", and an **unsorted** list of 7-letter English words. Assume that there are n such words in the list, and that all words consist of **only** the 26 English alphabet letters.

Output: A word ladder sequence from "PHYSICS" to "GEOLOGY" that contains the smallest possible number of words. If such a word ladder is impossible, return NIL.

Requirement: The average-case runtime of the algorithm must be in $\mathcal{O}(n)$.

Part (a) [4 MARKS]

How do you model this problem using a graph? Describe what each vertex represents and what each edge represents. Indicate whether it is a directed or undirected graph.

Part (b) [2 MARKS]

In order to be able to construct your graph from Part (a) in $\mathcal{O}(n)$ average-case runtime, you need to use a data structure to store the list of 7-letter English words. What data structure should you use?

Question 8. (CONTINUED)

Part (c) [4 MARKS]

Give a clear description of the construction process of your graph from Part (a) and justify that it takes average-case runtime $\mathcal{O}(n)$.

Part (d) [5 MARKS]

Using the graph you constructed in Part (c), describe in clear English how to find a word ladder with the shortest sequence of words. Justify briefly that your algorithm is correct and has the desired runtime.

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.