FOR INTERNAL SCRUTINY (date of this version: 1/4/2021)

UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFORMATICS 2: INTRODUCTION TO ALGORITHMS AND DATA STRUCTURES

Inf2-IADS Sample Exam (2 hours)
(published 30th March 2021, for individual practice)

Thursday 1st April 2021

INSTRUCTIONS TO CANDIDATES

This is an Open Book exam.
You have 2 hours to complete all questions.

1. **Answer all five questions in Part A, and two out of three questions in Part B. Each question in Part A is worth 10% of the total exam mark; each question in Part B is worth 25%.**
2. **Use a single script book for all questions.**
3. **Calculators may be used in this exam.**

PART A:

1. (a) Illustrate the application of MergeSort to the list [3,9,8,2,4,1,6,5], by means of a diagram showing the recursive call structure. You should show both the argument passed to MergeSort on each call and the result returned by each call as the lists are recombined. [*5 marks*]

   (b) Suppose we restrict our use of MergeSort to lists of length 2^m for various natural numbers m. Give Θ-estimates *in terms of* m for the best-case, worst-case and average-case number of comparisons performed by MergeSort on such lists. You need not justify your answers. [*3 marks*]

   (c) Give a Θ-estimate, again in terms of m, for the amount of memory used by MergeSort on lists of length 2^m, assuming a version of MergeSort that is as space-efficient as possible. Briefly justify your answer. [*2 marks*]

2. Consider functions $f_1(n) = (\lg(n))^{\lg(n)}$, $f_2(n) = n^{\lg(n)}$, $f_3(n) = n^2$ and $f_4(n) = n$.

   (a) There is exactly one $i \in \{1, 2, 3, 4\}$ such that

   $$f_i(n) \in O(f_j(n)) \ \text{ for all } j = \{1, 2, 3, 4\}.$$

   (ie, f_i is "big-O" of all the other f_j).
   State which function is f_i, giving informal justification of $f_i(n) \in O(f_j(n))$ for each $j \neq i$. [*5 marks*]

   (b) There is also exactly one $k \in \{1, 2, 3, 4\}$ such that

   $$f_k(n) \in \Omega(f_j(n)) \ \text{ for all } j = \{1, 2, 3, 4\}.$$

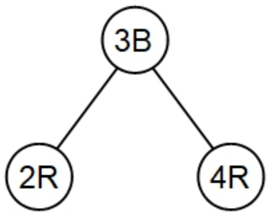
   (ie, f_k is "big-Ω" of all the other f_j).
   State which function is f_k, and rigorously justify $f_k = \Omega(f_j)$ for each $j \neq k$. [*5 marks*]

3. (a) Suppose you require a data structure for storing sets of up to 10,000 integers, supporting membership testing and insertion. You are trying to decide between two implementation schemes:

   - a bucket-array hash table of size 2,000, using chaining to resolve collisions,
   - a red-black tree implementation, storing integers at the internal nodes.

   Discuss the relative advantages and disadvantages of each scheme. Your discussion should be as specific as possible, including rough numerical estimates of relevant quantities based on your understanding of these data structures. [*6 marks*]

(b) The following red-black tree is a possible representation of the set $\{2, 3, 4\}$. The letters R and B show whether nodes are red or black. We have omitted the trivial leaf nodes.



With the help of diagrams, explain the steps that occur when the new element 1 is inserted into this set. [*4 marks*]

4. A *decimal counter of length* n consists of an array D indexed by $0, \ldots, n-1$, in which each cell contains one of the decimal digits $0, \ldots, 9$. The *value* $v(D)$ of such a counter is the sum $\sum_{i=0}^{n-1} D[i] * 10^i$.

The following operation increments the current value of the counter modulo 10^n:

```
Inc():
    i = 0
    while i<n and D[i]==9
        D[i] = 0
        i = i+1
    if i<n
        D[i] = D[i]+1
```

(a) Give a Θ-estimate for *best case* runtime of Inc. Give an example, parametric in n, illustrating best-case behaviour (you need not prove that it does so). [*2 marks*]

(b) Give a Θ-estimate for *worst case* runtime of Inc. Give an example, parametric in n, illustrating worst-case behaviour (again without proof). [*2 marks*]

(c) Suppose now that we apply the Inc operation 10^n times (so that the counter returns to its starting state). Give a Θ-estimate for the *total runtime* for this sequence of operations, and one for the *amortized average cost* per Inc operation. Justify your answer by means of a mathematical analysis of the total runtime. [*6 marks*]

5. In class we gave details of a *polynomial-time reduction from* the 3-SAT problem to the INDEPENDENT SET problem. In this question we consider strategies for attempting a reduction in the opposite direction (from INDEPENDENT SET to 3-SAT).

   (a) Consider a 2-literal clause $C = (\ell_1 \vee \ell_2)$ and show how we can design a pair of 3-literal clauses C' and C'' such that C is true under the identical conditions as $C' \wedge C''$. *[2 marks]*

   (you may introduce "dummy variables" if this helps)

   (b) Consider a k-literal clause $C = (\ell_1 \vee \ldots \vee \ell_k)$ for $k \geq 4$ and show how we can design a sequence of 3-literal clauses $C(1), \ldots, C(k-2)$ such that C is true under the identical conditions as $C(1) \wedge \ldots \wedge C(k-2)$. *[2 marks]*

   (you may introduce "dummy variables" if this helps)

   (c) Suppose now that we are given a graph $G = (V, E)$ and we are interested in whether G has an Independent Set of size ≥ 4. Show how to construct a 3-CNF formula Φ with a polynomial number of clauses, such that G has an independent set of size ≥ 4 if and only if Φ is satisfiable. *[4 marks]*

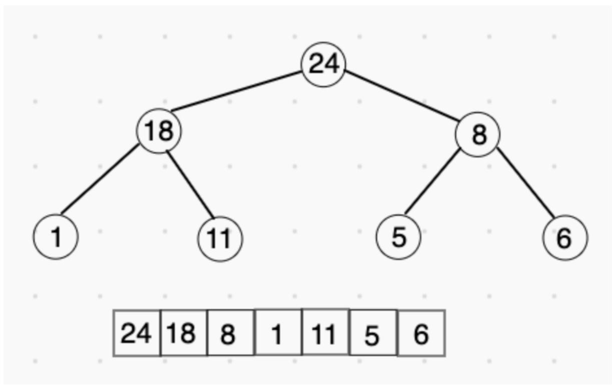
   [**Hint:** You will need to add clauses to capture the "no two adjacent vertices" constraints, and other clauses to encode the "at least 4 vertices" constraint (for this, you will need to start with a clause for each set of vertices of size $|V| - 3$). You may make use of (a) and (b).]

   (d) Suppose you were asked to generalise the reduction in (c) to the case where we were interested in an Independent Set of size $\geq h$, for an arbitrary h. Would the approach of (c) still give a polynomial-time reduction? *[2 marks]*

PART B:

1. (a) Consider the Heap below, and suppose that we carry out the following operations in sequence: Heap-Extract-Max(), Max-Heap-Insert(17), Heap-Extract-Max(), Max-Heap-Insert(7), Max-Heap-Insert(19).

   What is the order of the (remaining) elements in the Heap array after this sequence of operations has been performed? [*5 marks*]



   (b) We will now consider *Priority Queues* and their implementation. Recall that Priority Queues support the operations maxElement, removeMax and insertItem(k, e). These can be directly implemented by a Heap, via the methods Heap-Maximum, Heap-Extract-Max, and Max-Heap-Insert (under the assumption that Heap cells now contain an element e as well as a key k).

   Suppose that we would like to implement an data structure *Extended Queue* which offers the extra method findElement(k) (for a given key value k), as well as the standard operations MaxElement, removeMax, insertItem(k, e).

   i. Describe an algorithm for implementing findElement(k) on a Heap data structure. State the worst-case running time of your algorithm as a $\Theta(\cdot)$ expression, giving justification. [*5 marks*]

   ii. Explain briefly how you could implement maxElement and removeMax on a Red-Black tree. State the worst-case running time you expect to get for these methods in $\Theta(\cdot)$, justifying your answer. [*5 marks*]

   iii. Compare and contrast the advantages of using a Heap for the *Extended Queue* implementation against using a Red-Black tree. [*5 marks*]

   (c) Finally, reconsider the Heap realisation of *Extended Queue*, and a new method called UpdateKey(o, k'), where the required action is to update the key value of object o to become k', maintaining the Heap property. Note o is an object reference to some existing Heap cell (k, e).

   Give details of how UpdateKey(o, k') can be implemented in $O(\lg(n))$ time on a Heap, with reference to known Heap methods if this is helpful. [*5 marks*]

2. In English, there are two common ways of forming a list of items. One is to insert *and* between all adjacent items. The other is to insert a comma between adjacent items, except for the last two where *and* is inserted:

   cows and pigs and goats and sheep
   cows , pigs , goats and sheep

   The following context-free grammar generates lists of both these kinds. The start symbol is L. The non-terminals AL, CAL, CL respectively stand for 'and-list', 'comma-and-list', 'comma-list'.

$$\begin{array}{rcl}
\mathsf{L} & \rightarrow & \mathsf{AL} \mid \mathsf{CAL} \\
\mathsf{AL} & \rightarrow & \mathsf{I} \mid \mathsf{I}\ \textit{and}\ \mathsf{AL} \\
\mathsf{AL} & \rightarrow & \mathsf{CL}\ \textit{and}\ \mathsf{I} \\
\mathsf{CL} & \rightarrow & \mathsf{I} \mid \mathsf{I}\ ,\ \mathsf{CL} \\
\mathsf{I} & \rightarrow & \textit{cows} \mid \textit{pigs} \mid \textit{goats} \mid \textit{sheep} \mid \cdots
\end{array}$$

   (a) Even though our grammar is not in Chomsky Normal Form, it is still possible to construct CYK-style parse charts for it. Draw up and fill out a CYK-style chart for the following phrase as a 5×5 matrix:

   cows , goats and sheep

   For all possible substrings, your chart should record all non-terminals that can generate the substring — not only the ones that contribute to a parse of the complete phrase. (On this occasion, please do *not* include pointers or other information showing the origin of table entries.) [*7 marks*]

   (b) Is the above context-free grammar *ambiguous*? Justify your answer. [*2 marks*]

   For the remainder of this question, we shall discard the lexical rules with left-hand-side I, and treat I simply as a terminal symbol along with , and *and*.

   (c) Explain why our grammar is *not* LL(1). Your explanation should present a specific example of an input string and identify the point at which the LL(1) parsing algorithm is bound to fail. [*4 marks*]

   (d) The following is a parse table for an LL(1) grammar equivalent to the one above. The start symbol is again L, and the terminals, non-terminals and productions can be read from the table. (For example, the top left entry tells us that there is a production $\mathsf{L} \rightarrow \mathsf{I}\ \mathsf{Rest}$.)

	I	*and*	,	$
L	I Rest			
Rest		*and* I ATail	, I CTail *and* I	ϵ
ATail		*and* I ATail		ϵ
CTail		ϵ	, I ATail	

Show the execution of the LL(1) parsing algorithm on the input string:

$$\mathsf{I}\ ,\ \ \mathsf{I}\ \textit{and}\ \mathsf{I}$$

Present your solution as a table with a row for each step of the computation, as in lectures. The three columns should be 'Operation applied', 'Remaining input' and 'Stack state'. [*10 marks*]

(e) Briefly discuss whether LL(1) grammars are in general an appropriate technology for natural language processing. [*2 marks*]

3. (a) Hamiltonian Cycle: Given an undirected input graph $G = (V, E)$, determine whether there is some cycle of length n in G (a cycle which visits each vertex *exactly* once).

Consider the brute-force approach to finding a Hamiltonian Cycle, where we choose some specific starting vertex s, and iteratively build a path $p = s \dots v$ from that point: at each step, we consider the most recent vertex v and add any $(v, v') \in E$ such that v' is not yet an element of the path, updating $p' \leftarrow p, v'$. If at any point the path includes all vertices of V, we check that $(v, s) \in E$, and if this succeeds, then we have found a Hamiltonian cycle; otherwise we need to backtrack.

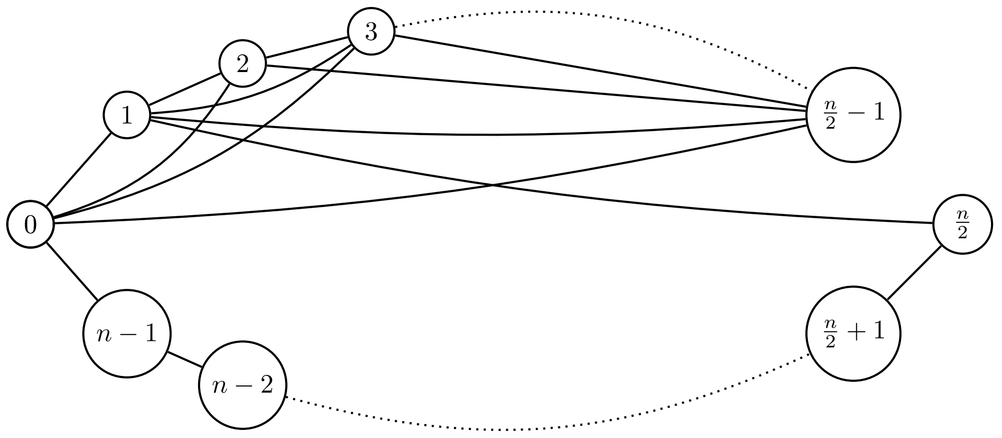
When following this naïve process, it is likely that we arrive at a situation where we cannot find any $(v, v') \in E$ such that $v' \notin p$; then we need to backtrack and undo the prior edge added to the current path.

Assume we explore the feasible next vertices in order of increasing index.

Adapt the *depth-first search* algorithm to develop pseudocode for this brute-force search for a Hamiltonian cycle. [*7 marks*]

(b) Consider an unweighted input graph $G = (V, E)$ where $V = \{0, \dots, n-1\}$ with edge set E defined by a *complete* graph on the nodes $\{0, 1, \dots, \frac{n}{2} - 1\}$, plus a linear path connecting the nodes $\frac{n}{2}, \frac{n}{2} + 1, \dots, n-1, 0$, and the extra edge $(1, \frac{n}{2})$.

This graph CLEARLY *does* contain a Hamiltonian cycle. Show that the brute-force search described in (a), started from node 0, and with the feasible next vertices explored in order of increasing index, will take at least $\Omega(2^{n/2})$ time before finding a Hamiltonian cycle. [*8 marks*]



(c) Now we consider a related decision problem on graphs, the Travelling Salesman problem (TSP), this problem defined on *weighted complete* graphs.

TRAVELLING SALESMAN: Given a natural number k, and an undirected input graph $G = (V, E)$ such that $(u, v) \in E$ for every $u, v \neq u \in V$ (*complete* graph) with a weight function $w : E \to \mathbb{R}^+$ on the edges of G, determine whether there is a simple cycle containing every vertex (exactly once), with total weight *at most* k.

We will consider the undirected graph $G = (V, E)$ from part (b), and expand this to a *complete graph* $G' = (V, E')$ with weight function w by setting

$$w(i,j) \;=\; \begin{cases} 1 & \text{for every edge } (i,j) \in E, \\ n & \text{for every } (i,j) \text{ such that } i \neq j, (i,j) \notin E \end{cases}$$

i. Specify a value of k such that a tour of G' of value $\leq k$ corresponds to a Hamiltonian cycle of G. You do not need to justify your answer. [*2 marks*]

ii. Suppose we run the Greedy algorithm for TSP on the weighted graph G' starting from 0, "breaking ties" between adjacent edges of equal weight in favour of lower-indexed vertices. Will the tour constructed correspond to a Hamiltonian cycle of G or not? Justify your answer. [*4 marks*]

iii. Suppose that we start with the *identity permutation* ($\pi(i) = i$) as our initial tour of G', and then we run the `SwapHeuristic` method. What will be the consequence of running this method? Justify your answer. [*4 marks*]