UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR08026 INFORMATICS 2: INTRODUCTION TO ALGORITHMS AND DATA STRUCTURES

Wednesday 19th May 2021

13:00 to 15:00

INSTRUCTIONS TO CANDIDATES

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. **Calculators may be used in this exam.**

Convener: D.K.Arvind
External Examiner: J.Gibbons

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

PART A

1. Consider the following sequence of commands in Python:

```
A = [[2,3,5],[2,3,5]]
B = [A,A]
C = B
D = C[0][1][2]
```

   (a) Draw a schematic representation of the contents of program memory after executing these four commands, showing the data living on the stack and the heap and how these are related. A memory location containing a reference to another memory location should be represented by an arrow, as in the examples in lectures. [*6 marks*]

   (b) What will each of the following expressions evaluate to? In cases where the value is a reference to some heap object, indicate the relevant object instance via a label added to your diagram (e.g. '`B[0][0]` yields a reference to the object labelled x').

```
B[0][0]
B[0][1]
B[1][0]
B[0][1][2]
```

[*4 marks*]

2. Suppose we are using a probe-based hash table to store a set of natural numbers. Our table has size 10, and the hash-probe function is $h(n, i) = (n + 3i) \bmod 10$. Each entry in the table is initialized to -1.

   (a) Explain in steps what happens when the following sequence of operations is performed:

   insert(16), insert(36), insert(29), lookup(46)

   Show the state of the table at the end of this sequences of operations. [*7 marks*]

   (b) A programmer proposes to implement a delete operation for this hash table as follows: to perform delete(n), simply locate the table entry containing n if there is one, and replace the n by -1.

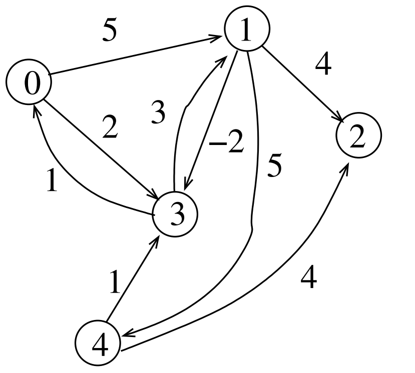
   Give an example to illustrate the problem with this idea, starting with the table state reached at the end of part (a). Your example should consist of a delete operation followed by one further table operation. Explain what happens when these operations are performed. [*3 marks*]

3. (a) Suppose that $f_1(n) = n/4$ and $f_2(n) = \sqrt{n}$ for all natural numbers n. For which of the four pairs (i, j) (where $i, j \in \{1, 2\}$) is it the case that $f_i \in O(f_j)$? For which pairs is it the case that $f_i = o(f_j)$? You need not give justifications here. [*4 marks*]

(b) Give a rigorous justification for your answer to whether or not $f_2 \in O(f_1)$, appealing to the definition of O. [*3 marks*]

(c) Give a rigorous justification for your answer to whether or not $f_1 \in o(f_1)$, appealing to the definition of o. [*3 marks*]

4. Consider the *all pairs shortest paths (APSP)* problem on (directed) weighted graphs, and the dynamic programming algorithm for solving APSP. Run the dynamic programming algorithm on the directed graph below, considering the sets of vertices V_k in increasing order, and constructing each of the matrices $D^{<k}$ for $k = 0, \ldots, 5$. Please justify your updates (or lack of updates) with a sentence of two about each $D^{<k}$. [*10 marks*]



5. In this question we consider QuickSort and its auxiliary method Partition:

```
Algorithm QuickSort($A, p, r$)

1. if $p < r$ then
2.     $split \leftarrow$ Partition($A, p, r$)
3.     QuickSort($A, p, split - 1$)
4.     QuickSort($A, split + 1, r$)
```

```
Algorithm Partition($A, p, r$)

1. $pivot \leftarrow A[r].key$
2. $i \leftarrow p - 1$
3. for $j \leftarrow p$ to $r - 1$ do
4.     if $A[j] \leq pivot$
5.         $i \leftarrow i + 1$
6.         exchange $A[i]$ and $A[j]$
7. exchange $A[i + 1]$ and $A[r]$
8. return $i + 1$
```

The elements of A are composite, with at least a field "*key*".

(a) Demonstrate the operation of QuickSort on the input array $5, -4, 2, -1, -3$. [*5 marks*]

(b) The "best case" running-time for QuickSort is known to be $\Theta(n \lg(n))$, and the "worst case" is $\Theta(n^2)$.

Suppose that we consider the restricted case where the all values come from the restricted set $\{1, 2, \ldots, 10\}$ (the input array including many duplicates, of course). [*5 marks*]

Give a $\Theta(\cdot)$ for QuickSort's worst case running-time on these restricted input instances, justifying your reasons.

Give a $\Theta(\cdot)$ for QuickSort's best case running-time on these restricted input instances, justifying your reasons.

PART B

1. In lectures we have mostly considered various sorting algorithms (e.g. InsertSort, MergeSort) as applying to array representations of lists. In this question, however, we shall consider variants of these algorithms that work on *linked lists.*

   For our purposes, a (singly) linked list cell will be an object `C` with two fields:

   - `C.value`, which will always contain an integer,
   - `C.next`, which will contain a reference to the next cell in the list, or else the special value `nil` if `C` is the last cell in the list.

   The expression `Cell(a,C)` will create and return a new cell `D` with `D.value = a` and `D.next = C`. Both the `value` and the `next` fields may be re-assigned after a cell has been created. For instance, the commands

   ```
   A = Cell(4,nil)
   A.next = Cell(8,nil)
   B = Cell(2,A)
   ```

   result in a linked list `B` representing [2,4,8]. Note that a linked list is given simply by a reference to its head cell, or else by `nil` in the case of the empty list.

   (a) Give pseudocode for a function `Insert(A,a)` which takes a sorted linked list `A` and inserts a new cell bearing the value `a` in the correct position. Your function should return the updated linked list. [*4 marks*]

   (b) Give pseudocode for a function `InsertSort(A)` which takes an arbitrary linked list `A` and returns a *sorted* linked list with the same contents as `A`. The linked list referenced by `A` should be unchanged by this operation. [*5 marks*]

   (c) Assuming that all the basic operations on list cells take constant time, what is the asymptotic worst-case runtime for `InsertSort` on linked lists of length `n`? Briefly justify your answer (your explanation need not be too detailed or formal). [*4 marks*]

   (d) In the array-based version of InsertSort given in lectures, we go through the elements of a *source array* from left to right, inserting them in turn into a *target array* which we populate from left to right, shifting existing elements rightward as necessary. Suggest a family of lists, one for each length `n`, for which the performance of our linked list based InsertSort will be asymptotically better than the array-based one. What is the asymptotic runtime of each version of InsertSort on this family of lists? [*3 marks*]

(e) We now consider how to adapt MergeSort to linked lists. This could be done by splitting our linked list at each level into two sublists, but here we consider an approach that avoids the need for a separate pass through the list to perform this splitting. You may suppose you are already given a function `Merge(A,B)` that takes two sorted linked lists `A`, `B` and merges them to create a new sorted linked list, which it returns. (You need not write pseudocode for `Merge`.)

Write pseudocode for a recursive function `MergeSort(A,n)` which takes a linked list `A` and a positive integer `n`, known to be at most the length of `A`, and returns a pair `(B,C)` where

- `B` is a sorted linked list with the same contents as the first `n` cells of `A`,
- `C` is the remaining portion of `A` (after the first `n` cells).

Your pseudocode should be an adaptation of the standard MergeSort algorithm, treating `n=1` as the base case. [*8 marks*]

(f) Assuming that `Merge` works in the expected way, what is the asymptotic worst-case runtime for your `MergeSort` as a function of `n`? You need not justify your answer. [*1 mark*]

2. As explained in lectures, the CYK parsing algorithm processes an input of length n in time $O(n^3)$, and in some sense detects all possible parses for the input. However, this does not mean that it explicitly lists all possible syntax trees, of which there might be very many. In this question we shall investigate this phenomenon in more detail.

   Consider the following grammar in Chomsky normal form. The start symbol is NP, and the lowercase English words are terminals.

$$\begin{array}{rcl} \text{NP} & \rightarrow & \text{NP NP} \\ \text{NP} & \rightarrow & \text{A NP} \\ \text{NP} & \rightarrow & \text{AP NP} \\ \text{AP} & \rightarrow & \text{NP A} \\ \text{NP} & \rightarrow & \text{duck} \mid \text{egg} \mid \text{blue} \mid \text{border} \\ \text{A} & \rightarrow & \text{blue} \end{array}$$

   (a) First apply the standard algorithm to construct a complete CYK chart for the phrase

   duck egg blue

   Include pointers to indicate how entries in the table were obtained from other entries. You may include multiple entries in each table cell, and should even include multiple occurrences of the same non-terminal if there is more than one way of obtaining it. *[6 marks]*

   (b) Explain how you would adapt the CYK algorithm to compute the total *number of syntax trees* for the given input with respect to a Chomsky normal form grammar G. Your answer should make clear exactly what kind of information is stored in each cell, and how this information is computed. You need not give complete pseudocode, though you may use mathematical formulae and/or small pseudocode fragments to make your method clear.

   (You may find it helpful to develop your method in connection with the example in (c) below.) *[7 marks]*

   (c) Apply your method to compute the number of syntax trees for the phrase

   duck egg blue border

   with respect to the grammar above. You should ensure that each cell in your chart contains all necessary information, and explain any notation conventions you adopt. *[6 marks]*

   (d) Now consider the following simple grammar (with start symbol S):

$$\text{S} \rightarrow a \mid \text{SS}$$

   Show that for $n \geq 2$, the string a^n has at least 2^{n-2} syntax trees under this grammar. Reason by induction, with $n = 2$ as the base case. *[4 marks]*

(e) Could there be a general parsing algorithm that explicitly lists all syntax trees for the input string, and which (for any fixed grammar) runs in polynomial time with respect to the input length? Justify your answer. [*2 marks*]

3. The VERTEX COVER optimization problem asks for a Vertex Cover of minimum size for the input graph $G = (V, E)$. Recall that a subset $\mathcal{K} \subseteq V$ is said to be a *Vertex Cover* for G if for every $e \in E$, at least one of e's endpoints lies in $\mathcal{K}$.

   The decision version of VERTEX COVER is NP-complete and we do not expect to be able to solve it exactly in polynomial-time. In this question we consider a simple Greedy algorithm.

   Algorithm Greedy-VC($G = (V, E)$)

```
1. $E' \leftarrow E$
2. Initialise array $K$, $K[v] \leftarrow 0$ for all $v \in V$
3. Initialise array $D$, $D[v] \leftarrow deg(v)$ for each $v \in V$
4. while $(E' \neq \emptyset)$
5.       $w \leftarrow \text{argmax}_v\{D[v] : D[v] \neq 0\}$
6.       $K[w] \leftarrow 1$
7.       Update $E', D$ to reflect deletion of all edges adjacent to $w$
8. return $K$
```

   Note that $\text{argmax}_v\{D[v]\}$ denotes the v which achieves the max value for $D[v]$.

   (a) Assume that the graph $G = (V, E)$, and the auxiliary "graph" E', are both represented with an $n \times n$ *Adjacency matrix* data structure.

   Give details of how we can implement Greedy-VC to to achieve a running-time bound of $\Theta(n^2)$ for Greedy-VC, explicitly specifying the details of how we initialise D in line 3., how we compute w in 5., how we update D and E' in line 7., and how we perform the test of the while-loop. [*10 marks*]

   (b) We consider specially structured bipartite graphs called *Spindle graphs*, and analyse the operation of Greedy-VC on these graphs.

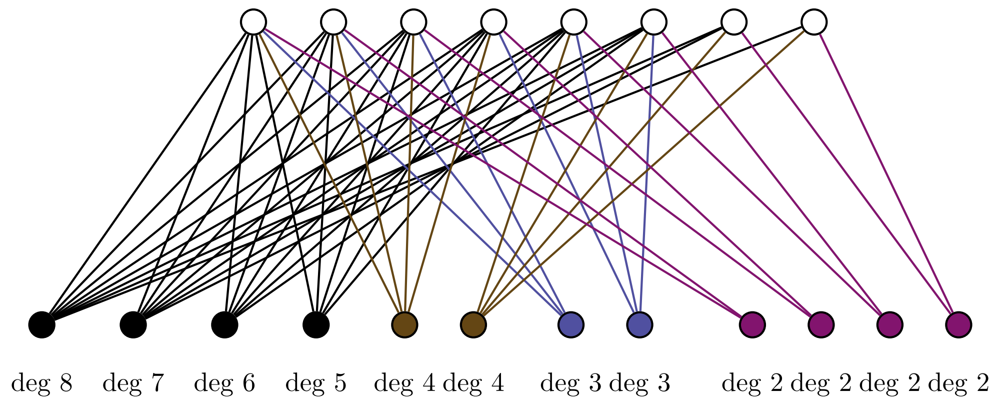
   In what follows, note that $\dot\cup$ is the "disjoint union" symbol.

   As with all bipartite graphs, the set of nodes is separated as $V = L\dot\cup R$, every edge having one endpoint in L and one in R. In our construction, we will set L to have n nodes (for some n which is a power of 2). We define $R = \dot\bigcup_{k=2}^{n} R_k$, where each R_k consists of $\lfloor \frac{n}{k} \rfloor$ vertices, each of which will have degree k. The total number of nodes in R will be greater than in L.

   We define the edge set $E = \dot\bigcup_{k=2}^{n} E_k$ of the Spindle graph in terms of the edges E_k adjacent to the R_k vertices. For every $k = 2, \ldots, n$, E_k must be a set of $k \cdot \lfloor \frac{n}{k} \rfloor$ edges from the vertices R_k to L such that

   - each $v \in R_k$ has degree k
   - every $u \in L$ is adjacent to *at most one* of the R_k vertices.

The diagram below gives an example of a Spindle graph for $|L| = 8$.



i. Show, for every $v \in L$, and every $m, 2 \leq m \leq n$, that the number of adjacent vertices to v from $R_m \cup \ldots \cup R_2$ is at most $m - 1$. *[4 marks]*

ii. Give an inductive argument to show that if we run Greedy-VC on a Spindle graph, that the Vertex Cover computed will contain all nodes in R. *[8 marks]*

iii. Briefly describe the optimal Vertex Cover for a Spindle graph. *[3 marks]*