Module Title: Inf2-IADS
Exam Diet (April): 2023
Brief notes on answers: ***Fill in***
PART A

1. **JL**

   (a) Bookwork, from lectures:

   $$f = o(g) \text{ iff } \forall c > 0.\ \exists N.\ \forall n \geq N.\ f(n) < cg(n)$$

   [Up to 3 marks. $\forall n > N$ and/or $f(n) \leq cg(n)$ also accepted.]

   (b) False, false, true. [1 mark each.]

   (c) No [1 mark.] Suppose $f = o(g)$ and $g = o(f)$, and consider $c = 1$. Then we obtain N such that $\forall n \geq N.\ f(n) < g(n)$, and also N' such that $\forall n \geq N'.\ g(n) < f(n)$. Taking any $n \geq \max(N, N')$ gives $f(n) < g(n) < f(n)$, a contradiction. [Up to 3 marks for the proof.]

2. **JL** The pseudocode here may take some time to assimilate, but the solutions require very little writing.

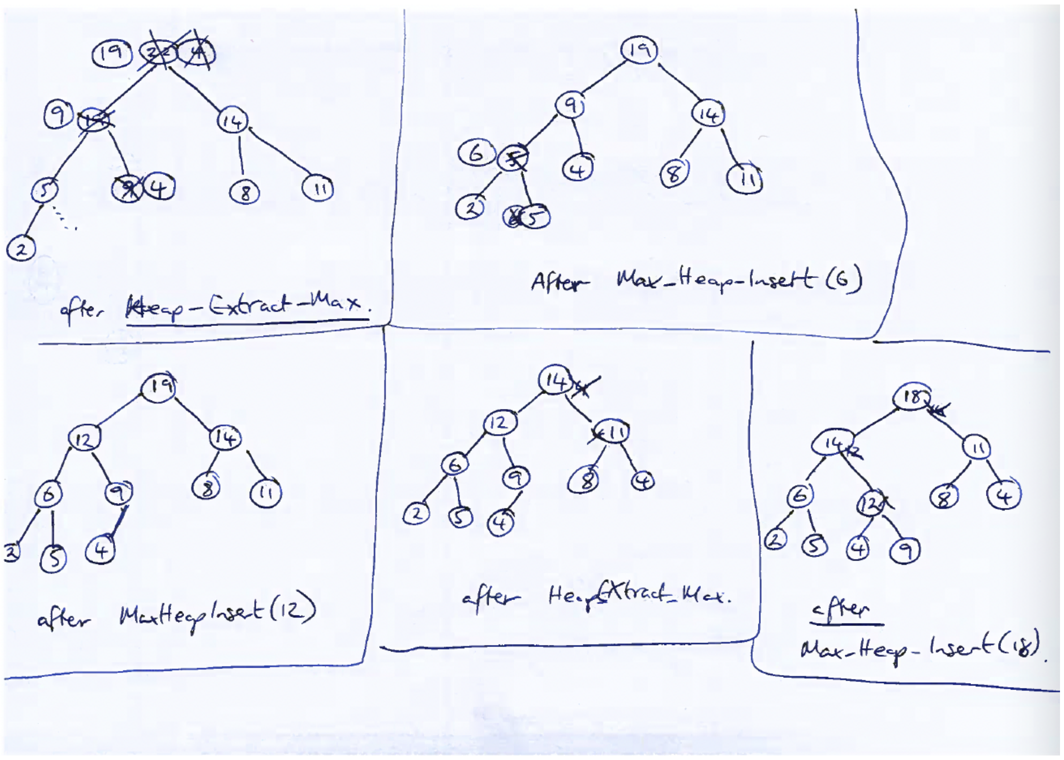
   (a) Best case is with the reverse-sorted array $A = [n-1, n-2, \ldots, 0]$: exactly $n-1$ comparisons. There is just one comparison for each $i = 1, \ldots, n-1$, as the A[j] ¡ x test yields false even when j=0. [1 mark for the example, 1 marks for exact number of comparisons (anything else $\Theta(n)$ would get 1 mark), 1 mark for justification.]

   (b) For the usual algorithm, a reverse-sorted array gives the worst-case number of comparisons: $1+2+\cdots+(n-1) = n(n-1)/2 = \Theta(n^2)$. For each $i = 1, \ldots, n-1$, the new element x is compared with the i existing elements at positions from $i-1$ down to 0. [2 marks for exact formula, 1 for $\Theta(n^2)$, 1 for justification.]

   (c) The number of line executions will be $\Theta(n^2)$. This is because for each i we are doing $\Theta(i)$ work: any indices $< i$ not featuring in the first loop will feature in the second. [1 mark for $\Theta(n^2)$, 2 for justification.]

3. **MC.**

   (a) **worked example, 7 marks**


after ~~Heap~~-Extract-Max.
After Max-Heap-Insert (6)
after MaxHeapInsert (12)
after HeapExtract-Max.
after Max-Heap-Insert(18).

   marking: 1 mark each for showing the result of each operation, 2 marks going for some explanations along the way.

   (b) **worked details, 3 marks**

   For $Left(x)$, the formula is $2x + 1$, for $Right(x)$ the formula is $2x + 2$. For $parent(x)$, the formula is $\lfloor \frac{x-1}{2} \rfloor$.

   marking: 1 mark each for the correct answer.

4. **MC. This was part of a tutorial question this year. Values in (b) have been changed.**

   (a) The algorithm is driven by two nested for-statements, the outer iterating n times, the inner one iterating C times. The statements within the inner loop just carry out $\Theta(1)$ operations (comparison, addition, subtraction) on each iteration, so overall $\Theta(nC)$ time.

   Marking: Up to 4 marks for this answer, with some justification of why.

   (b) The following is the main dynamic programming table, where the cell value for (i, j) is the value of the "max-knapsack which uses items 1 to i to achieve value at most j".

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2
2	0	0	2	3	3	5	5
3	0	0	2	3	4	5	6

marking: Up to 6 marks depending on correctness/completeness of the table. Will give at most 4 marks if done in ad-hoc memoized way.

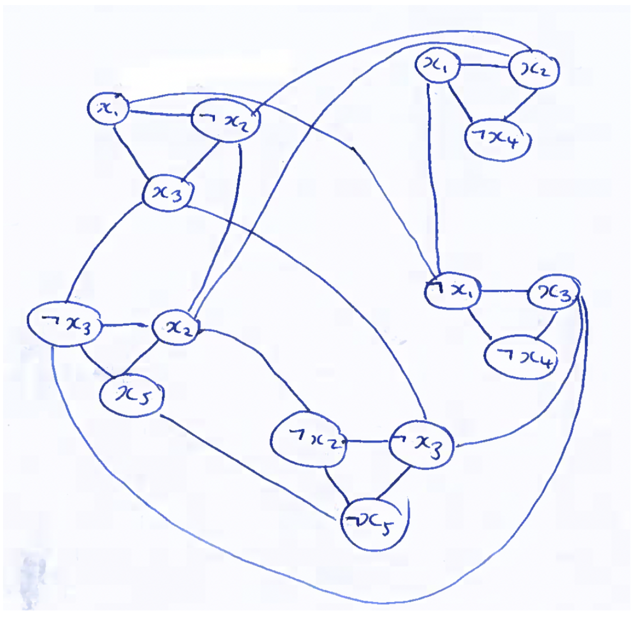
5. **MC. Note: a different Part (b) appeared as a Question in the 2021 resit (not published online), the clauses are changed here**

   (a) **fleshing-out details, 5 marks** The students are asked to give a specific $\Theta(n+m)$ algorithm and they should take care to achieve that time. First thing they should do is read the input into memory, as adjacency list structure. They also should read the sequence of I nodes into memory, and the sensible way to do this is to initialise an array I of length n to 0, and then set $I(i) \leftarrow 1$ if i is read while scanning the input sequence. This array representation can then be exploited to check "Independence" in a simple iteration as follows:
   "for $i = 0$ to $n-1$, check first whether $I(i) = o?$. If so, skip.
   If $I(i) = 1$, then scan the adjacency list $A[i]$ for i ... if any u in that list has $I[j] = 1$, **return** No.
   If the loop completes (without No being returned), we return Yes.

   It is clear that the work done is $O(n)$ to read in the input set into I, and that the for loop runs in time proportional to $m = |E|$. hence the running time.

   marking: Up to 5 marks, depending on quality of details. They can get 3 marks for just doing polynomial checking but not the $\Theta(n+m)$.

   (b) b́f worked example, 5 marks
   Here is the Independent set graph for the example formula.



   The number of edges can be counted as 3×5 (for the 5 triangles, then for blocking edges between the positive/negative literals we have 2 (x_1) and 4 (x_2) and 4 (x_3) and 0 (x_4 always negative) and 1 (x_5). This is 26 in total.

   marking: up to 3 marks for the details of the diagram, 2 marks for the 26.

PART B

1. **JL question**

   (a) Anything equivalent to:

```
Lookup(A,a,i,j):
    if j<i then raise KeyNotFound
    else
        k = floor((i+j)/2)
        case compare (a, A[k].key):
            Lt => return Lookup(A,a,i,k−1)
          | Eq => return A[k].value
          | Gt => return Lookup(A,a,k+1,j)
```

   Essentially bookwork, but the three-way compare construct makes it a bit different from the lecture version.

   [4 marks for broadly the right structure including recursive calls, 4 marks for the details. Any correct way of handling the base case(s) is acceptable.]

   (b) $T(n) = \Theta(1)$ if $n = 0$, otherwise $T(n) = T(n/2) + \Theta(1)$. So the Master Theorem applies with $a = 1$, $b = 2$, $k = 0$. Here $a = b^k$ so the solution is $T(n) = \Theta(n^k \lg n) = \Theta(\lg n)$. [2 marks for recurrence, 2 marks for solution.]

   The best case is when the first item examined has the desired key: runtime in such cases is $\Theta(1)$. [1 mark]

   (c) Anything equivalent to:

```
HashLookup(H,a):
    c = #(a)
    if H[c] = null then raise KeyNotFound
    else return Lookup(H[c],a,0,size(H[c])−1)
```

   (d) Since $15 = 2^4 - 1$, it is clear that Lookup on an array of length ≤ 15 will require at most 4 comparisons. Thus, HashLookup will involve 1 hashing and at most 4 comparisons: $\Theta(1)$ time overall. [1 mark for 4 comparisons, 1 mark for 1 hashing, 1 mark for $\Theta(1)$.]

   (e) Suppose I has n items. Taking the time of a key comparison as our time unit, and ignoring everything except hashings and comparisons, the worst-case time for Lookup will be $\lfloor \lg n \rfloor + 1$, and that for HashLookup will be at most 7+4=11. So the worst case of HashLookup will be faster if $\lfloor \lg n \rfloor \geq 11$, i.e. if $n \geq 2^{11} = 2048$. [2 marks for a reasonable approach, 1 mark for a plausible right answer (not being too fussy), 2 marks for quality of explanation.]

2. **MC.**

(a) **bookwork, 5 marks**

```
Algorithm bfs(G)
1. Initialise Boolean array visited, setting all entries to FALSE.
2. Initialise Queue Q
3. for all $v \in V$ do
4.     if visited[v] = FALSE then
5.         bfsFromVertex(G, v)
```

```
Algorithm bfsFromVertex(G, v)
1. visited[v] = TRUE
2. Q.enqueue(v)
3. while not Q.isEmpty() do
4.     $v \leftarrow$ Q.dequeue()
5.     for all w adjacent to v do
6.         if visited[w] = FALSE then
7.             visited[w] = TRUE
8.             Q.enqueue(w)
```

Marking: Up to 5 marks depending on good details. At least 2 marks depend on use of the Queue data structure.

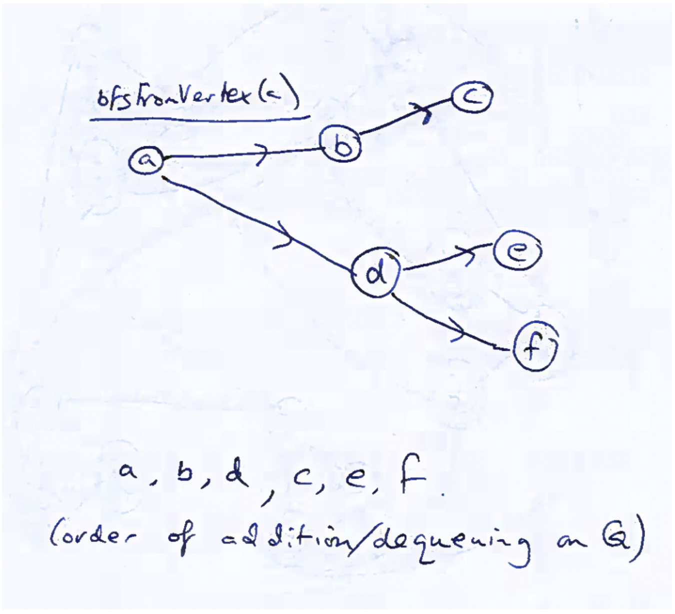
(b) **bookwork/thinking (DFS (similar) was done in detail in lects), 5 marks**

By examination, they should note that the top-level method consists of $O(n)$ initalisation work, as well as a $O(n)$-bounded loop where the work is done by the collective bfsFromVertex calls.

To analyse the work done by (all the) bfsFromVertex calls, we note that we will mark each vertex as "visited" when it is pushed onto the Queue, and that the push only is done for "not yet visited" vertices. So "all the pushes" take $O(n)$ overall. The loop in bfsFromVertex will explore all adjacent nodes from v when that vertex gets deQueued (which can only happen 1 time, as it is only Queued once). With an Adj.List structure that is $O(out(v)$ time ton explore all the edges for v, and overall $O(m)$ time.

marking: Up to 5 marks depending on quality of analysis.

(c) **worked example, 7 marks**



marking: Up to 7 marks depending on quality of answer (linear order is at least 2 marks)

(d) **(problem solving, was tutorial Qn, 8 marks total)**

(i). The algorithm is based on bfs and bfsFromVertex, with a minor change: we label the starting vertex v as "blue", and as we explore, we colour the nodes on each level alternating between "red" and "blue" at successive levels (so the neighbours of v will get colour "red", and so on). As we do this, we may notice that a neighbour w of our current explored node u has already been generated at the same level - if this is the case, we return "failure" immediately; alternatively, if we generate the entire tree without this happening, then this component is bipartite. If all "bfsFromVertex" calls pass, then the entire graph is bipartite.
marking: Up to 4 marks depending on correctness/level-of-detail.

(ii). Note that an undirected graph is bipartite if and only if each connected component is bipartite. Therefore we just need to show that the method is correct for any maximal connected component $C \subset V$ of G.
In the case where we explore the entire component without ever seeing an edge between two vertices on the same level, we have constructed the bipartition (red/blue), having checked all edges. In the case we find one of these level edges during exploration, we note that taking the least common ancestor $\ell = lcs(u, w)$ and the paths from ℓ to u, from ℓ to w, and (u, w) shows an odd-length cycle.
marking: 1 mark for the connected components detail, up to 3 marks for arguing correctness.

3. **JL question**

(a) The three stages are:

1 Eliminate all rules with ≥ 3 symbols on the right-hand side as follows: given a rule $X \to x_1 x_2 \ldots x_n$ where $n \geq 3$, introduce fresh non-terminals $X_2, \ldots, X_{n-1}$ and replace the rule with $X \to x_1 X_2$, $X_2 \to x_2 X_3$, $\ldots X_{n-1} \to x_{n-1} x_n$.

2 Eliminate unit rules $X \to Y$ where Y is non-terminal. For each such rule, and each existing rule $Y \to \alpha$, add the rule $X \to \alpha$. Repeat until no unit rules left.

3 For every terminal a appearing in a RHS of length 2, add a fresh non-terminal Z_a with rule $Z_a \to a$, and replace a by Z_a within such RHSs.

[Bookwork. 2 marks each. A less formal answer than the above if fine if the idea is clear.]

(b) After stage 1:

$$\mathsf{S} \longrightarrow \mathsf{X} \mid \mathsf{XY} \qquad \mathsf{Y} \longrightarrow \ \mathsf{:S} \qquad \mathsf{X} \longrightarrow \mathsf{a} \mid \mathsf{(Z} \qquad \mathsf{Z} \longrightarrow \mathsf{X)}$$

After stage 2:

$$\mathsf{S} \longrightarrow \mathsf{a} \mid \mathsf{(Z} \mid \mathsf{XY} \qquad \mathsf{Y} \longrightarrow \ \mathsf{:S} \qquad \mathsf{X} \longrightarrow \mathsf{a} \mid \mathsf{(Z} \qquad \mathsf{Z} \longrightarrow \mathsf{X)}$$

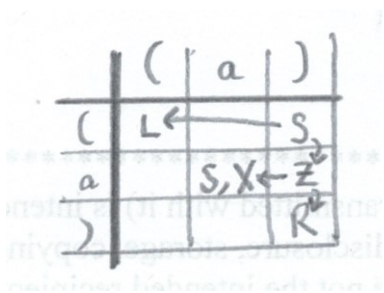
After stage 3:

$$\mathsf{S} \longrightarrow \mathsf{a} \mid \mathsf{LZ} \mid \mathsf{XY} \qquad \mathsf{Y} \longrightarrow \mathsf{CS} \qquad \mathsf{X} \longrightarrow \mathsf{a} \mid \mathsf{LZ} \qquad \mathsf{Z} \longrightarrow \mathsf{XR}$$

$$\mathsf{L} \longrightarrow \mathsf{(} \qquad \mathsf{R} \longrightarrow \mathsf{)} \qquad \mathsf{C} \longrightarrow \mathsf{:}$$

[Roughly 2 marks each.]

(c) The CYK chart is as follows:



	(	a	)
(	L		S
a		S, X	Z
)			R

[1 mark for a table of the right form, 2 marks for the entries, 1 mark for the pointers.]

(d) Stage 1: For a rule of size $k \geq 3$, we add $k-2$ non-terminals and $k-1$ productions: $\Theta(k)$ work. So this stage takes $\Theta(n)$ work overall in a bad case (which arises e.g. when there are $\geq n/4$ rules of size 3).

Stage 2: There are clearly $O(n)$ unit rules $X \to Y$, and for each of these, $O(n)$ corresponding rules $Y \to \alpha$. So $O(n^2)$ work altogether. Moreover, this can be attained e.g. if there is a specific non-terminal Y with say $n/3$ unit rules $X \to Y$ and $n/3$ rules $Y \to \alpha$. So $\Theta(n^2)$.

Stage 3: Applying just this stage to a grammar of size m is clearly $\Theta(m)$ work. But if the grammar is the output from stage 2, we have $m = \Theta(n^2)$ in the worst case, so stage 3 will take $\Theta(n^2)$ work.

[This part is a slightly unusual kind of question and may be challenging (though Stage 1 is easy). 3 marks per stage, being quite generous — e.g. only 0.5 per stage for noting how the worst case arises.]