Module Title: Inf2-IADS
Exam Diet (Dec/April/Aug): Aug 2021
Brief notes on answers:
PART A

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

   (b) $f(n)$ is bounded below by $(n^2 - n + 1)\lg n$, and above by $(n^2 - n + 1)\lg(n^2) = 2(n^2 - n + 1)\lg n$. Hence clear that $f(n) = \Theta(n^2 \lg n)$.

   [2 marks for the answer, 3 for the justification. The above would suffice: explicit values for c, C, N not required.]

2. (a) In the first step, a new red node bearing 4 (with two trivial leaves) is added as the right child of '3'. In the second step, the red uncle rule is applied, so '3' and '7' turn black and '5' turns red. In the third step, the red root is turned black.

   [2 marks for first step, 3 marks for second step, 2 marks for third step.]

   (b) 4 times. Each application of red-uncle pushes a non-trivial black downwards in order to eliminate a double-red; this may introduce another double-red at the next level up. Since there are 4 non-trivial blacks along each path, this can happen at most 4 times, and it's clear that this max can be attained.

   [1 mark for answer, 2 for explanation.]

3. The course of computation is:

Operation	Input left	Stack state
	(n)	Exp
Lookup (, Exp	(n)	(Exp) Ops
Match (	n)	Exp) Ops
Lookup n, Exp	n)	n Ops) Ops
Match n	)	Ops) Ops
Lookup Ops,)	)	) Ops
Match)		Ops
Lookup Ops, $		Stack empties

[2 marks for attempting something of the right form, and 1 mark per correct line.]

4. **(worked example and problem solving, 10 marks)**

   We have been given the initial distance matrix $D^{<0}$ in the question.

   To compute $D^{<1}$ we allow ourselves to consider intermediate visits through V_1, which is $\{0\}$. We can do this by computing the "matrix sum" of column 0 and row 0, which is

$$\begin{bmatrix} 0 & -1 & 3 & 3 \end{bmatrix} \quad \text{“}+\text{”} \quad \begin{bmatrix} 0 \\ 2 \\ \infty \\ 2 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} - & - & - & \\ - & 1 & 5 & \mathbf{5} \\ - & \infty & \infty & \infty \\ - & \mathbf{1} & \mathbf{5} & 5 \end{bmatrix}$$

with the highlighted values being lower than what we had in $D^{<0}$. Hence

$$D^{<1} = \begin{bmatrix} 0 & -1 & 3 & 3 \\ 2 & 0 & 2 & \mathbf{5} \\ \infty & \infty & 0 & 4 \\ 2 & \mathbf{1} & \mathbf{5} & 0 \end{bmatrix}.$$

Next for building $D^{<2}$ we consider intermediate vertex 1 and this may be helpful, given the -1 valued edge ($0 \to 1$). The alternate options are given by the following matrix:

$$\begin{bmatrix} 2 & 0 & 2 & 5 \end{bmatrix} \quad \text{“}+\text{”} \quad \begin{bmatrix} -1 \\ 0 \\ \infty \\ 1 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} 1 & - & \mathbf{1} & 4 \\ - & - & - & - \\ \infty & - & \infty & \infty \\ 3 & - & \mathbf{3} & 6 \end{bmatrix},$$

with the values which improve on $D^{<1}$ in bold. We have

$$D^{<2} = \begin{bmatrix} 0 & -1 & \mathbf{1} & 3 \\ 2 & 0 & 2 & 5 \\ \infty & \infty & 0 & 4 \\ 2 & 1 & \mathbf{3} & 0 \end{bmatrix}.$$

The matrix $D^{<3}$ will be identical to that for $D^{<2}$ - to see this, note that any simple path through simple path through vertex 2 incurs a cost of at least 5 (assuming the -1 edge is used), which is $\geq$ every non-∞ value already there. Also, the remaining ∞ values are in the row for 2, so cannot be improved in this iteration.

Finally, we can allow routes with 3 as an intermediate point. Every simple path through vertex 3 uses the subpath $2 \to 3 \to 0$, and (allowing the use of the -1 edge) has cost at least 5, and only is of value in improving ∞ cells.

For cell $(2, 0)$, we have $D^{<3}[2, 3] + D^{<3}[3, 0] = 4 + 2 = 6$.

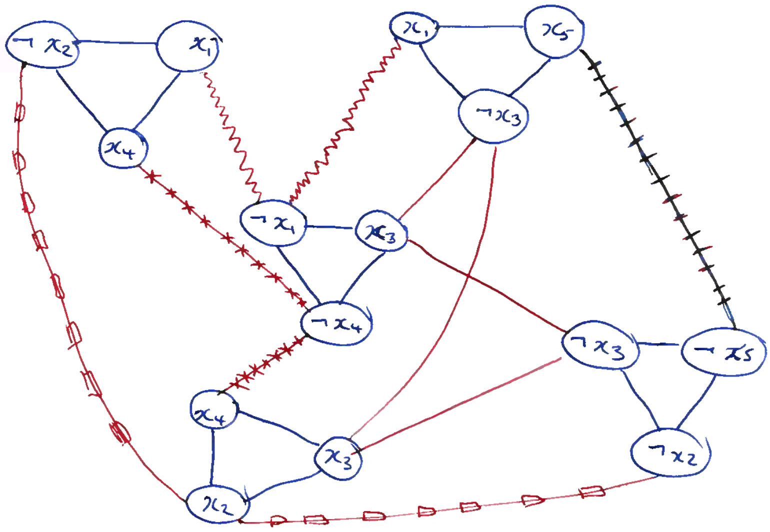
For cell $(2, 1)$, we have $D^{<3}[2, 3] + D^{<3}[3, 1] = 4 + 1 = 5$. Hence

$$D^{<4} = \begin{bmatrix} 0 & -1 & 1 & 3 \\ 2 & 0 & 2 & 5 \\ 6 & 5 & 0 & 4 \\ 2 & 1 & 3 & 0 \end{bmatrix}.$$

marking: 3 marks for computing/explaining $D^{<1}$ (depending on detail), 3 marks for computing/explaining $D^{<2}$, 2 marks for explaining why $D^{<3}$ is $D^{<2}$, 2 marks for computing/explaining $D^{<4}$.

5. **(worked example for 6, some reasoning for 4)**

(a) The graph.



The reduction aims to find one vertex from each triangle (to satisfy each clause) hence we require $k = 5$ for the I.S. The edge-count is 26, coming from 5 triangles (15 edges total) and the clashing literals for x_1 contributing 2 edges, for x_2 contributing 2, for x_3 contributing 4, for x_4 contributing 2 and for x_5 contributing 1.

marking: 3 marks for the diagram, 1 mark for k, 2 marks for the number of edges.

(b) VERTEX COVER and INDEPENDENT SET have a very close relationship which leads to an almost-trivial (polynomial-time) "reduction": we know that for any graph G, G has an Independent set of size $k \Leftrightarrow G$ has a Vertex Cover of size $n-k$. This is for the exact same graph, and means that the reduction in either direction just flips between k and $n-k$.

Therefore to reduce 3-Sat to VERTEX COVER, we just carry out exactly the same graph construction, but then change the value k to $n-k$ (in our example we would use $15-5=10$ for the Vertex Cover reduction.

marking: 2 marks for a good explanation/construction, just 1 mark for using transitivity of reductions.

(c) The reason we can tell this formula must be satisfiable is because we known that (by analysis of the expected value of a random assignment) every 3-CNF formula has an assignment to satisfy $\geq \frac{7}{8}m$ clauses. Since we have $m = 5$ clauses, this value is $\frac{35}{8} = 4.375$; hence we know some assignment satisfies all 5 clauses.

marking: 2 marks for the right answer with a decent explanation.

PART B:

1. (a) Something like the following (minor variations allowed):

```
MergeSort3(A,p,q):
   if q-p == 1
      return [A[p]]
   else if q-p == 2
      if A[p] <= A[p+1] then return [A[p],A[p+1]]
      else return [A[p+1],A[p]]
   else
      r = floor((2p+q)/3),  s = floor((p+2q)/3)
      B = MergeSort3(A,p,r)
      C = MergeSort3(A,r,s)
      D = MergeSort3(A,s,q)
      return Merge3(B,C,D)
```

[3 marks for something of broadly the right form; 2 marks for correct base cases; 2 marks for details of split.]

(b) The recurrence relation is:

$$T(n) \;=\; \begin{cases} \Theta(1) & \text{if } n \leq 2 \\ 3T(n/3) + \Theta(n) & \text{otherwise} \end{cases}$$

The Master Theorem therefore applies with $a = b = 3, k = 1$. Since $\log_3 3 = 1$, we are in the 'balanced case' and the solution is $T(n) = \Theta(n \lg n)$ (as for binary mergesort).

[3 marks for recurrence relation, 1 for solution, 2 for explanation.]

(c) For a subarray of size 9, there are 1+3+9 = 13 calls to MergeSort3. In general, for $n = 3^d$, there are $(3n-1)/2$ calls (obtained as the geometric sum $\sum_{i=0}^{d} 3^i$). For ordinary mergesort, the corresponding formula is $(2n-1)$ where $n = 2^d$.

[2 marks for the size 9 case, 2 for the general formula, 2 for the binary analogue.]

(d) We reason informally. If we first ignore the $O(1)$ 'error terms', then at each level of the recursion, the sizes of all calls to Merge3 sum to n, so the number of comparisons for this level is estimated by $5n/3$. And there are about $\log_3 n$ recursion levels. So total number of comparisons is about $(5n/3)\log_3 n = (5/(3 \lg 3))n \lg n \simeq 1.05 n \lg n$.

For the error terms, there is an $O(1)$ contribution for each call, and by part (c) there are $\Theta(n)$ calls. So we have an upper bound of $(5/(3\lg 3))n \lg n - \Theta(n)$ for the number of comparisons. For binary mergesort, we know from lectures, the corresponding formula is just $n \lg n - \Theta(n)$ (we know from lectures that a binary merge of size n requires at most $n-1$ comparisons).

[This part is more challenging, though I'm not expecting anything more rigorous than the above. 3 marks for the coefficient $5/(3\lg 3)$; 1 mark for the error term; 2 marks for the binary version.]

2. (a) (bookwork/simple workings, 4 marks) 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: 4 marks for the right answer

   (b) **(worked example, 8 marks)**

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	3	3	3	3	3
2	0	0	0	3	4	4	4	7
3	0	0	0	3	4	5	5	7

   marking: 2 marks for dimensions/labelling right, 1 mark for 0-row and 0-column, the other 5 marks for the column details.

   (c) **(worked example, 4 marks)**

   The empty set will satisfy the constraint, as will each of the singleton sets. This is 4 sets altogether. For sets with two distinct weights, the only set that will fit is $\{3, 4\}$. Therefore 5 of the 8 subsets are feasible.

   marking: 4 marks for the right answer.

   (d) **(problem solving, 6 marks)**

   The new recurrence for $feas$ basically exchanges the "maximisation" for summation:

$$feas(\mathbf{w[k]}, C') = \begin{cases} 1 & C' = 0 \\ 1 & k = 0 \\ feas(\mathbf{w[k-1]}, C') & k > 0, C' > 0, w_k > C' \\ (feas(\mathbf{w[k-1]}, C') + & \text{otherwise} \\ feas(\mathbf{w[k-1]}, C' - w_{k+1})) & \end{cases}$$

   Note the extra difference that at the base cases we write 1 (as our measure is the existence of a solution, not the value of that solution).

   marking: 4 marks for the details of the recursive branches, and 2 marks for the base cases being correct.

   (e) **(reflection, 3 marks)**

   Whether this naïve method will be a fast way to return a uniform random sample depends on how dense the feasible solutions in the set $\{0, 1\}^n$. If $feas(\mathbf{w[n]}, C) \geq d \cdot 2^n$ for some constant $d > 0$ which is bounded away from 0, then we would expect to be able to return a solution after a reasonable number of trials.

   marking: 3 marks for a reasonable consideration of relevant issues.

3. The Firefighter problem.

(a) **worked example, 6 marks**

If we choose the defence strategy (s, r), then at time step 0, we have threatened nodes q and s ... and the first defence is node s, so q burns and s is safe. However, q now only has one not-burnt-yet neighbour, r, so T_1 is then just $\{r\}$.

Then by defending r at time step 2, that means that q has no undefended threatened neighbours, and therefore the fire's spread is contained ... after 2 defence steps.

marking: 3 marks for the shorter strategy, and 3 marks for the explanation.

(b) **problem solving, 13 marks**

Algorithm fireFighter($G = (V, E)$, f, $v_1, \ldots, v_k$)

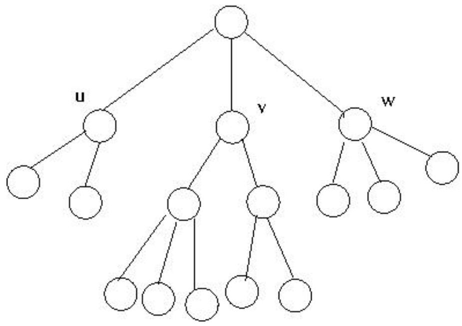
```
initialise arrays B, D to "all FALSE"
B[f] ← TRUE
Q.enqueue(f, 0)
count ← 1
t ← 0
while (t ≤ k + 1 and Q.isEmpty() = FALSE) do
    (u, t') ← Q.dequeue()
    if t' = t and t ≤ k + 1 then
        if count = 0 return "TRUE"
        else
            t ← t + 1
            count ← 0
            if t ≤ k then D[v_t] ← TRUE
    for all w ∈ Nbd(u) do
        if B[w] = FALSE and D[w] = FALSE then
            B[w] ← TRUE
            count ← count + 1
            Q.enqueue(w, t)
return "FALSE"
```

marking: Up to 10 marks for the pseudocode, depending on how good. Up to 3 marks for explaining why it's still $O(m + n)$.

(c) **problem solving, 6 marks**

Students are asked to give a graph where the "effective degree" heuristic fails to minimize steps to containment. Here is one example, a tree: the optimum (for time steps to containment) strategy is to defend v at step 1, thus containing/burning the tree in 3 steps; however, the "effective degree" strategy will defend w first, thus forcing 4 steps to contain the burning.

marking: 4 marks for the example, 2 for explaining.


u
v
w