Question 1

1.1

- To get full marks for this sub-question, students are expected to state all and only the seven squares that form the *good* set of squares with minimal cost.
- The question explicitly asks for the "*good* set of squares with minimum total cost". So just stating the minimum cost would not attract any marks.
- There is more than one square with a given cost. Hence, referring to a square with its cost is not clear enough.

1.2

- The most common mistake observed in this question was that the students not specifying the process for recovering the *good* set of squares with the minimum total cost, after running a max-flow algorithm. Edmonds-Karp algorithm or Ford–Fulkerson algorithm returns the maximum flow (a single number) which is equal to the minimum cut. Therefore, after running a max-flow algorithm, the students are expected to traverse the residual flow network and recover the *good* set of squares with minimum total cost.
- Some students did not state all the features of the flow network that is being constructed. Such mistakes could have been avoided by following a systematic approach to presenting these features.

Question 2

Students generally did well on this problem.

2.1 Students generally did well on this section. To be awarded all the marks for the construction of the graph, the construction of all edges must've been made explicit and the response must explain how this reflects the constraints of the problem. Some common issues were:

- Failing to omit vertices which represent the jth town on day i if there are no people, or equivalent.
- Failing to explain how/when edges should be constructed from day to day.

To be awarded all the marks for the algorithm, students must explain why the max flow/result of the DFS/BFS represent the solution to the problem. Some common issues were:

- Running a max-flow algorithm and failing to justify why it terminates in the specified complexity.
- Failing to justify why the max-flow or BFS/DFS represents a solution.

To be awarded all the marks for the time complexity analysis, a correct complexity bound must've been stated for the algorithm chosen. And then bounds in terms of n and m must've been explicitly computed and substituted.

We were lenient on the justification of correctness, so responses which achieved higher marks expressed why their final evaluation of the result of the BFS/DFS or max flow algorithm yield a correct response. Some common issues were:

- Explain why a successful DFS/BFS implies the existence of a slow person but failing to explain why an unsuccessful DFS/BFS would imply the opposite.
- The inverse case of above.

2.2 Students general did well on this section. To be awarded all the marks for the construction of the graph, the construction of all edges must've been made explicit and the response must explain how this reflects the constraints of the problem. Some common issues were:

- Failing to consider vertex weights and splitting them, rather making incoming/outgoing edges have weight $p_{i,j}$ which did not solve the problem correctly. This is due to it failing to account for the case where the graph allows for a flow greater than $p_{i,j}$ through a particular node, which violates the constraints of the problem.
- Omitting the weight of non-trivial edges.
- Failing to explain how the max-flow/augmented flow represents the constraints of the problem.

To be awarded all the marks for the algorithm, students must explain why the max flow represent the solution to the problem. Some common issues were:

- Did not explicitly justify the relationship between max-flow and max(slow people) or between max(slow people) and min(fast people).

The marks awarded for time complexity are of similar standard to 2.1. The marks awarded for justification of correctness are of similar standard to 2.1.

Question 3

Overall this was a bit of a tricky question. Most students had good ideas towards solving it, but many struggled to properly encapsulate the k-dislikes condition within a flow network.

A lot of students chose to submit only 3.2. If you are going to do this, please double check that your solution actually does solve 3.1 as well. Markers will substitute $k = 0$ into your solution and mark it as if it were submitted for 3.1 as well - there were many cases where key edges had capacity k and hence just disappeared in 3.1 resulting in empty networks.

A lot of students also did not quite understand what the maximum flow of their network represented. For this question, the maximum flow does not represent the maximum number of songs, but rather the maximum number of matches that could occur. In fact, this does not even guarantee that all these matches could be allocated in a way that all songs can be played. Additional steps are required to actually extract the maximum number of songs possible.

The sample solution also does not prove that after finding a maximum number of matches such that flow is equal to nx, an allocation will exist for x many songs. A more rigorous solution would prove this, but it is not particularly easy and so we did not expect students to prove it either.

We were fairly lenient with time complexity calculations related to the graph construction.

In general, testing your algorithms against as many edge cases as possible is good practise.

3.1

- Some students attempted a greedy approach which simply looked for the minimum number of outgoing/incoming edges for any boy or girl. The counter example for this is: $n = 3$, where b_1 likes g_1 and g_2, and g_3 likes b_2 and b_3. In this case, the min edges is 1, but actually no songs can be played.
- The counter example above also applies for solutions that had capacity n from source/to sink, connected b_i to g_j if they liked each other, and then ran max flow and tried extracting a solution based on that.
- The edge between boys/girls must have capacity 1 to ensure they do not dance multiple times.

- Many solutions tried removing edges from the network once they had been paired, before running max flow again to test if another pairing was possible. The counter example to this is: $n = 3$, where b_1 likes g_1, g_2, b_2 likes everyone, and b_3 likes g_2, g_3. Here, you could have the first round of pairing where b_i pairs with g_i, but this makes it impossible to pair again. The correct solution would be 2 songs.

3.2

- Carry forward error was applied here if you made a mistake in 3.1 and reused that mistake in 3.2, provided that the mistake did not reduce the difficulty of the question.
- Solutions which tried adding k to the answer from 3.1 are incorrect. The first counter example in the previous section demonstrates this. When $k = 0$, max songs is 0, but when $k = 1$, max songs is 2.
- Similar for solutions which tried creating a "dislike" bipartite graph, getting the max songs from this, and then adding it to the 3.1 solution.
- A few solutions also tried to prioritise finding full songs that contained liked or disliked pairings. There could be optimal solutions which require you to spread the disliked pairings equally throughout the songs, so this approach is also flawed.
- Some solutions tried to add a maximum of k edges to the 3.1 network flow to account for the dislike condition. The issue here is which edges do you add? In most cases, this creates issues in the possible pairings and without an oracle of some sort this is likely impossible to do.
- Some students also seemed to confuse k for the number of likes/dislikes each boy/girl had, and hence limited the edges to the "like" subgraph to $(n - k)$ capacity which is incorrect.

Question 4

Question 4 tested a wide range of topics and skills ranging from graph search algorithms to the content of network flows. It was critical to understand the behaviour of all the functions utilised within the the algorithm to demonstrate that the time complexities as required were achieved. As the questions were similar in nature and progressively built off of each other, the following comments apply generally unless noted otherwise.

- Although it was quite clear, there were more than expected responses where the output was not given in constant time either due to ongoing computation or the use of hash tables/maps without explicitly defining how collisions were handled. Please note that in the worst case the access of an element in a hash map is $O(n)$ not $O(1)$ when called upon generically as a data structure.
- It was repeatedly seen that submissions tried to utilise the Ford-Fulkerson algorithm in place of the Edmonds-Karp algorithm but specified the time complexity of the EK algorithm. These algorithms are not the same thing and cannot be used like for like when aiming to achieve a specific time complexity.
- Quite a number of statements were generic in nature such as, '*let S be the set of vertexes reachable from the source.*' Reachable can mean many things and therefore in all instances does not actually communicate how the algorithm was undertaking specific consideration of vertexes within the flow network.
- The question specified that the edges were directed but again was something that was overlooked by many students. Examples given of adding edges between two sets meant nothing if it wasn't specific about start and end points e.g. '*added edge has one end in S and one end*

in T' does not provide sufficient information to determine whether the added edge is heading in the right direction.

- For question 4.2, many submissions did not justify why in several instances the maximum flow did not increase.
- For questions 4.3 and 4.4, many students tried to use BFS/DFS to find the max flow from the source to a vertex v in the residual network in place of using EK.
- For questions 4.3 and 4.4, there were some instances where submissions suggested that the maximum flow did not change at all.