Question 1

Students generally did well on the first half of this question, as it is effectively identical to a tutorial question. The second half was reasonably challenging, some students had a correct idea, few were entirely correct.

1.1 Most coherent attempts to this section had the correct idea and many were awarded almost all the marks. To achieve all the marks,

- sub-problems must've been defined precisely, such that it is entirely sufficient to justify the recursion using the definition of the sub-problem.
- the recursion must contain all 4 cases, with their respective reasoning's provided either explicitly, or justified implicitly.
- the solution to the original problem must be explicitly produced by some computation or explicit reference to a particular sub-problem. Furthermore, reasoning must be provided.
- all the steps above must be justified, and some explanation about why this computation is both possible, and produces the correct solution.
- explicitly state the order of computing the sub-problems, AND justify why this is possible.
- the time complexity must've been justified. This involves reasoning about the complexity of solving each sub-problem, the bound on the number of sub-problems, and the complexity of any over-head.

Note that we did recognise that the issue of stepping out of the grid can be addressed multiple ways, including extra base cases to cover this. We were generally lenient and gave all the marks where possible to explanations in prose for what should occur when we recurse to i = 1 or j = 1.

1.2 Most coherent attempts to this section had the correct idea and a portion were awarded almost all the marks. The same standards as 1.1 were required to achieve all the marks. There were generally five different approaches to the problem.

[A] Many students used the sub-problem of determining $f(i, j)$, be the probability of passing square (i, j) on a random path. Then the expected value can be rearranged as

$$\sum_{1 \leq i \leq m, 1 \leq j \leq n} f(i, j) \times a_{i,j},$$

using linearity to decompose each path into the sequence of squares comprising it, and regrouping the terms corresponding to each square of the grid. If the recurrence was formulated correctly with respect to this sub-problem, the solution is valid. Of responses which obtained a valid solution using this approach, many lost marks for failing to clearly explain why the expected value equals this sum.

[B] Some students approached this problem going forward like in 1.1. They used the sub-problem of determining $e(i, j)$, the expected number of coins acquired travelling from square $(1, 1)$ to square (i, j). The most natural attempt at a recurrence here is

$$e(i, j) = a_{i,j} + \begin{cases} 0, & \text{if } i = 1, j = 1; \\ e(i, j-1), & \text{if } i = 1, j \neq 1; \\ e(i-1, j), & \text{if } j = 1, i \neq 1; \\ p \times e(i, j-1) + (1-p) \times e(i-1, j) & \text{otherwise.} \end{cases}$$

Unfortunately, this approach is incorrect. In the fourth case, we assign probabilities of p and $1-p$ to moves that are in fact forced, since we are in the bottom row or rightmost column respectively. This issue is exacerbated when moving to larger grids, as the same moves would no longer be forced when viewed in the context of a larger grid. Responses which took this approach were eligible for up to 6 marks.

For a specific counterexample, consider a grid with two rows and three columns. The path $(1,1) \to (1,2) \to (1,3) \to (2,3)$ is taken with probability p^2, so it should contribute $p^2(a_{1,2} + a_{1,3})$ to the answer. However when $i = 2$ and $j = 3$ in the above recurrence, where the fourth case applies, the corresponding term $e(1,3)$ is included with weight $1-p$ towards $e(2,3)$. In a grid with more columns though, the probability of a path beginning with these three steps is indeed $p^2(1-p)$. Thus, this choice of subproblems fails to exhibit the (enumerative analogue to the) *optimal substructure* property.

[C] Some students addressed the case of forced moves by adding 2 extra cases.

$$e(i,j) = a_{i,j} + \begin{cases} 0, & \text{if } i = 1, j = 1; \\ e(i, j-1), & \text{if } i = 1, j \neq 1; \\ e(i-1, j), & \text{if } j = 1, i \neq 1; \\ p \times e(i, j-1) + (1) \times e(i-1, j) & \text{if } j = n, i \neq m; \\ (1) \times e(i, j-1) + (1-p) \times e(i-1, j) & \text{if } i = m, j \neq n; \\ p \times e(i, j-1) + (1-p) \times e(i-1, j) & \text{otherwise.} \end{cases}$$

Unfortunately, this is still incorrect. The specific counter-example given above is still valid. The general reasoning for why this is still incorrect is that it makes the invalid supposition that the probability of coming from above is always p and the probability of coming from the right is always $(1-p)$. Responses that took this approach were eligible for up to 8 marks.

[D] Some students approached this by constructing a second dynamic programming algorithm to solve in parallel. This approach effectively combines approaches 1 and 2. First, we calculate $f(i,j)$, the probability of passing square (i,j) on a random path, as in the first method. Then we can do a second pass, with subproblems of determining $h(i,j)$, the expected number of coins acquired travelling from square $(1,1)$ to square (i,j). The most intuitive recurrence is

$$h(i,j) = a_{i,j} + \begin{cases} 0, & \text{if } i = 1, j = 1; \\ h(i, j-1), & \text{if } i = 1, j \neq 1; \\ h(i-1, j), & \text{if } j = 1, i \neq 1; \\ \lambda \times h(i, j-1) + \mu \times h(i-1, j) & \text{otherwise.} \end{cases},$$

where

$$\lambda = \frac{f(i, j-1)}{f(i, j-1) + f(i-1, j)}$$

and

$$\mu = \frac{f(i-1, j)}{f(i, j-1) + f(i-1, j)}.$$

Note that $h(i,j)$ depends not only on prior values $h(\cdot,\cdot)$, but also the multipliers λ and μ, which are calculated from the solutions to the earlier problem. The correct recursion to calculate f(i, j) is given in the solutions.

Some students also had two extra cases to account for forced move on $i = m$ or $j = n$.

Unfortunately this approach is not correct. The general reasoning for why is that the definition of $h(i,j)$ is the expected value of coins collected upon reaching $h(i,j)$ conditional on

reaching (i, j). However λ and μ are calculated conditional on reaching (m, n). Thus, this choice of subproblems fails to exhibit the (enumerative analogue to the) *optimal substructure* property.

[E] Some students correctly deduced that to a similar approach to the second approach but backwards would not suffer the same issue. This was the alternative solution given.

Question 2

Most students performed well in the first part of this question, but many struggled with the second part which was much trickier. Quite a few of the comments for part 2 will be applicable to DP more broadly as well.

2.1 There's not much to comment on here. Most students did well in spotting at least one of the issues with the greedy approach. The most common mistake was not reading the question properly, and not explicitly providing all the requested values in the counter example. To reiterate the question, we wanted to see values for all of:

- n, s_i's, e_i's, a_i's, b_i's, c_i's
- the calculated net earnings from the greedy approach
- a higher calculated net earnings that used a valid allocation of work

Most counter examples we saw used n values of 3 or 4, which is great. $n > 4$ is fine, but it's worth noting for future improvement that counter examples demonstrate a weakness in the algorithm. It's good practice to dig into what this weakness is, and see if it can be exploited more concisely. The smallest possible counter example for this question uses $n = 2$ where customer 1 has $(s, e, a, b, c) = (1, 3, 1, 0, 0)$ and customer 2 has $(s, e, a, b, c) = (2, 3, 2, 0, 0)$. Here Greedy earns \$1 whereas flipping the greedy allocation will earn \$2.

2.2 This question was tricky and lots of students struggled with correctly encapsulating all the different cases that were possible within a subproblem/recursion formula. General comments:

- When defining subproblems, make sure to clearly state what each of the variables are, and their ranges.
- You should always provide an explanation for what your recursive formula is trying to do. This involves explaining the conditions for each branch, and the behaviour the formula represents.
- Your subproblem must be related to your recursion; it's not just some random formula, high level idea, or summary of your DP solution. Any variables that your recurrence formula needs to track will most likely need to be included in your subproblem as well. That is, you can't say something like: "let $p(i)$ be the problem of solving $opt(i)$ which is the maximum profit for customers up to i" and then have a recurrence like:
  - $opt(i) = \max opt(i-1) + v_i$ where v_i is the optimal value for customer i by considering s_i, e_i, a_i and b_i
  - $opt(i, j, k) = ...$
  - $opt(i) = \max opt(i, j)...$
  - etc.

  Similarly, your base cases and final solutions should be represented in terms of the subproblem as well. If you're struggling to make all these pieces work together, it's very likely that your subproblem is flawed.

Some of the common mistakes we saw also included:

- Solutions that tried to compute $E(i)$, the maximum earnings for customers $1...i$, by taking the allocation that gives the maximum earnings $E(j)$ for some $j < i$ such that $e_j \leq s_i$, and extending it by adding customer i to the allocation.
- Solutions that did the above, but also possibly replacing the last customer (or more) in the old allocation if their time interval overlaps with customer i
- Solutions that used separate subproblems for Alice and Bob, but did NOT call each other in their recurrence
- Solutions that did not track the availability of Alice and Bob at all
- Solutions that recursed on specific conditions on the availabilities of Alice and Bob, but did not consider all possible cases (e.g. both available, but still do not want to take the customer), or did not take the most optimal of all branches.
- Solutions that were essentially minor improvements to the greedy algorithm provided in 2.1
- Solutions that did not consider the c_i's at all

The first two are flawed simply due to the number of possible combinations of allocations to consider - either a counter example will exist, or you've essentially brute forced the question in exponential time. The third considers Alice and Bob separately which may not be optimal - there are easily cases where both Alice and Bob need to sacrifice on individual maximum profits, for the benefit of the overall garage. The fourth is usually close to a n^3 solution, but has missed some key points in the recursion and can end up in the n^4 realms when corrected. The fifth will likely have a counter example that's a slight modification of the 2.1 solution, but is fundamentally flawed because Greedy cannot look far enough into the future. The sixth is obviously an issue.

Question 3

This problem statement and the time complexities specified positioned the use of the Floyd-Warshall algorithm as a good starting point for how these problems might be solved. Many students were able to make that identification however there were a number of students that didn't actually answer the question which appeared most commonly as returning of all edges identified in all shortest paths between any two vertices for both parts 3.1 and 3.2. Other observations for the specific questions are noted below.

3.1

- A number of students incorrectly used the Bellman-Ford algorithm in lieu of the Floyd-Warshall algorithm thinking they achieved the same result. These two algorithms are not the same and while they both are represent graph searches, the BF algorithm starts from one nodes and finds shortest paths to all other nodes while the FW algorithm finds all shortest paths in simple graphs.
- Some submissions submitted the make-up of the Floyd-Warshall algorithm i.e. sub-problems, recurrence, base cases, final answer derivation and associated time complexity either alone or with an additional statement noting that all found edges would be the returned in a set as being *useful*. This is effectively an incomplete solution.
- Following on from the previous item, utilising the FW algorithm in isolation was not sufficient. This was not considered wholly correct as the applicability of an edge being useful or not required further consideration rather than extracting the output of the FW algorithm.

- Submissions that did attempt to justify what it meant for an edge to be *useful* with regards to its weight and distances between any two nodes (hint: shortest paths) more often than not got it correct. Reiterating the question did not gain any marks in this regard.

3.2

- The bulk of the points made above in **3.1** apply here also with the main caveat being that submissions that were able to identify the additional conditions around what it meant for an edge to be *very useful* usually did it well.
- We were pleased to see that some students took inspiration from problem **3** of Assignment 2.
- Again, running the Floyd-Warshall algorithm and extracting all edges found was considered an incomplete solution.
- Submissions which identified the issue arising from zero-weight cycles received a bonus mark.

Question 4

This question is slightly tricky. However, Question 4.1 is pretty straightforward and most students scored full marks.

4.1

- This part is an algorithm design question. So, as usual, scoring full marks require justification of the correctness and the time complexity of the proposed algorithm.

4.2

- The most common mistake observed in this part of the question was that the students assumed this problem to be identical/similar to the knapsack problem. However, unlike in the case of the knapsack problem, this problem requires each winner to have a loser among the rest of the players. Hence, when picking a player to be in the set of winners, there should be a way to ensure that there is a player with a lesser skill level. This fact significantly changes the solution.
- One other common mistake was that the students assumes m to be the maximum imbalance that can be achieved. m is necessarily not the maximum imbalance (therefore $\frac{S+m}{2}$ is necessarily not the maximum skill level) that can be achieved with n players selected from $2n$ players.
- Some students did not follow the typical skeleton/structure that is used for stating a dynamic programming solution and missed essential details. Such mistakes could have been avoided by following a systematic approach taught in this course.
- Some students did not mention that they sort the skill levels (in descending order) before proceeding with the rest of the algorithm.