Question 1

Question 2

This question was straightforward and therefore we expected rigorous solutions from students. Vague and convoluted solutions were not awarded full marks.

2.1

- For full marks students had to explicitly state that the rations should be bought from the town where they are the cheapest among town 1 to town $n-1$ (in other words, rations bought from town i, where $c_i = \min(c_1, c_2, \ldots, c_{n-1})$) and include a convincing reason.

2.2

- Similar to Question 2.1, for full marks students had to explicitly state (along with a convincing reason) that the last change (if any) happens at the town where the rations are the cheapest among town 1 to town $n-1$.
- Students should note the difference between "changing the rations" vs "changing the rations for the last time".

2.3

- The expected output of this algorithm is the minimum cost to travel from town 1 to town n. Hence, students are expected to specify this output clearly.
- Variables and their initial values are expected to be clearly defined.
- Some students proposed solutions that resemble a set of instructions to a human rather than an algorithm. Students are expected to explicitly state the steps followed to accomplish each task.
- Students are expected to present their algorithms in plain English. Pseudo code should only aid an otherwise complete answer.

Question 3

The question was particularly difficult, leading to a wide range of marks across the cohort.

3.1

- Justification of Correctness:
  Many students fail to sufficiently justify the correctness of their algorithm. The marking was generally lenient in this regard. Justifications which were awarded all the marks clearly conveyed how the construction of the graph, combined with the graph algorithm applied, yields a correct output.
  Explicitly, this looked like: "By creating the super-source/sink and applying weights s_v and f_v respectively, any path between the source and the sink p' represents some path in the original graph p. The first vertex traveled to from the source in p' would represent the starting vertex in p, while the last vertex before travelling to the sink in p' would represent the last vertex in p. Hence the starting/finishing costs are accounted for in our constructed graph, and by definition, the minimum cost path is found using Dijkstra's algorithm."

Implicitly, this looked like: "We make the edges between the source and all vertices v as S_v to account for the cost of starting at that node ... and now we run Dijkstra's algorithm, as the shortest path between the source and the sink would contain the shortest path accounting for the starting and finishing costs.".
Justifications which were awarded some of the marks vaguely conveyed how the construction of the graph, combined with the graph algorithm applied, yields a correct output, or appropriately explained the rationale behind the construction. Some justifications simply chose to say "this is optimal as Dijkstra's works" or "clearly then, Dijkstra's finds the shortest path". These responses were awarded no marks as they do not make any progress towards justifying why the given solution is correct.
Note that an explanation of Dijkstra's was at no point necessary to be awarded all the marks.

- Justification of Time Complexity:
Some students fail to sufficiently justify the time complexity of their algorithm. Justifications that were awarded all the marks explicitly provided the time complexity of the graph algorithm used, and computed the bounds on the number of vertices and edges. An omission of any element led to partial marks deducted, e.g. failing to specify the data structure used to implement Dijkstra's algorithm or failing to explain how the number of nodes remains linearly bounded by n after constructing a new graph.

- Alternative solutions:
Many alternate solutions were given. Few were correct. There were generally many ways to formulate an algorithm which terminated correctly within $O((n+m)^2 \times \log(n+m))$. All of these effectively ran Dijkstra with every node as the starting node. After running each iteration of Dijkstra, the appropriate starting/finishing costs are added to their respective path weights, and the minimum cost path in this iteration is found. Then finding the minimum cost path across all iterations as the final output.
Alternative solutions which attempted to include the finishing costs into the Dijkstra search did not work as it allowed for path costs to decrease upon extending the path, making Dijkstra possibly omit the shortest path.
Almost all other alternatively solutions clearly did not work.

3.2

- Justification of Correctness:
Justifications which were awarded all the marks expanded upon the response from 3.1 by clearly conveying how the construction of the graph and graph algorithm applied reflects the added condition and therefore yields the correct output. This expansion upon the justification in 3.1 often looked like, "By adding all edges in G between u and v as edges between u and v' in G', any path between the source and the sink in G' must now go through at least 1 edge. By keeping the edges between all u and v from G in G', paths can still be more than 1 edge". Note how these responses had to contain how the construction reflected the 1 edge lower bound.
Justifications which vaguely justified this or appropriately explained the rationale were awarded some marks.

- Justification of Time Complexity:
Same issues as 3.1.

- Alternative solutions:
Many alternate solutions were given. Basically none were correct. Either they extended an incorrect formulation from 3.1, or incorrectly attempted to somehow exclude 0 edge paths within the Dijkstra implementation.
A common incorrect attempt to exclude 0 edge paths would be to not consider the super-sink when en-queuing vertices if the direct predecessor of the current vertex is the super-source.

This does not work as it potentially omits the shortest path that finishes on the current vertex. All attempts to make further modifications to address this case were either; incomplete, or raised more issues.

3.3

- Justification of Correctness:
  Justifications which were awarded all the marks expanded upon the response from 3.2 by clearly conveying how the construction of the graph and graph algorithm applied reflects the added condition and therefore yields the correct output.
- Justification of Time Complexity:
  Same issues as 3.1.
- Alternative solutions:
  Same issues as 3.2.

Question 4

This was a difficult question that stumped a lot of students. Some overall notes:

- Proof by example is not a valid proof. Just because an idea works for a specific array does not guarantee that it will work for all arrays
- Many proofs were either poorly explained, not thorough, or far too long. 1-2 pages is a decent guide for how long we'd expect most answers to be. 2 sentences is likely insufficient, whereas 5+ pages likely means you aren't explaining things as clearly as you could.
- The cost to swap an two values is based on the indices when you make the swap, not where they are at input. Lots of submissions tried to always swap an value directly into its correct position without considering where it will send the other value which is incorrect.
- Following from the above point, attempts that tried to always place at least 1 value in the correct spot at every swap is inherently flawed. A counter example is $A = [2, 3, 5, 1, 4]$ where no value can immediately go to its correct position without negatively affecting other values.

4.1 Mostly well done, and we applied some leniency here.

- Proof by induction here is difficult. You cannot use an inductive hypothesis that bases itself on the length of the array.
- Many people stated that even/odd parity mismatch situations always came in pairs without too much explanation. Some leniency was given if the rest of your proof was valid, but statements like this should be backed up.
- You cannot choose to ignore the absolute value signs without reasoning.

4.2 There seemed to be some confusion with this question. You had to prove that S is a lower bound for the cost of swapping, and there are 2 mains ways to do this:

- Show that any sequence of swaps cannot cost less than S dollars
- Prove that a solution to 4.3 is the optimal sequence of swaps

Some commonly encountered submissions were:

- State that the simplest array had only 1 swap out of place, which costs S to swap, and hence any other array must cost at least S. This isn't a great proof because it still leaves a lot of questions. What if more complex arrays have more complex series of swaps that could

reduce the time? What about arrays that don't have this scenario - is S still a lower bound? 1 mark was given in most cases.

- Some students claimed that the minimum distance required for each value to travel is $A[i]-i$. This is a decent start, but you also need to explain clearly where the 1/2 comes from and why the cost cannot be below this distance.
- A large number of students attempted a cycle/ring based proof here. It runs into issues in that sorting within a ring is essentially still the same problem, and you can't just swap adjacent elements expecting it to be the minimum. These submissions commonly mentioned forwards and backwards distances, but again, following these swaps directly does not guarantee a minimum cost swap.

4.3 This subquestion probably had a lot of variance between attempts, and unfortunately many were poor. Of the attempts that understood the overall idea, many still missed key details in what should be a formal proof. The key points were:

- Always move values closer towards their correct positions
- Ensure that you do not 'overshoot' when swapping
- Explain that this type of swap is always possible
- Explain that this type of swap is always optimal, and will always produce a sorting that costs exactly S dollars

You had to hit all of these points to get full marks here. Quite a few attempts mentioned the first two points, but not the last two. A proof by induction is non-trivial, and I'm not sure there was a successful attempt on this.

4.4 Again, lots of variance in the attempts for this question. Some common incorrect attempts/mistakes:

- Simply sorting the array with BubbleSort/MergeSort. This makes no attempt to minimise the cost, and would likely be awarded 0.
- Implementing the ideas in 4.3 that were flawed to begin with
- Somehow weaving in S into the time complexity because it is the cost of the swapping. The dollar cost does not represent any time spent in your algorithm. Rather, its the number of swaps that you have to perform is upper bounded by S.
- Not including the time taken to find a swap target at each step.
- Assuming that cycles/rings of values can be swapped immediately.
- Not ensuring conditions laid out in 4.3 were always met
- Incorrectly forcing the algorithm to begin sorting from one end of the array. This is done correctly in the alternative solution below, but many students did not account for cases where it is impossible to include the first value in the first swap.

Also, an alternative solution to the suggested solution: An in-place BubbleSort/SelectionSort style solution where you find each value from 1..n and swap its way down the array into place, ensuring that all swaps meet the criteria in 4.3.