Problem 1

There is not always a stable pair of schedules. Consider the following example:

Network A has two shows, a_1 and a_2, with ratings 20 and 40, respectively. Network D has two shows, d_1 and d_2, with ratings 10 and 30, respectively.

Each network can reveal one of two possible schedules. We denote the schedules as follows:

- Network A's possible schedules:
  - Schedule 1: $\{a_1, a_2\}$
  - Schedule 2: $\{a_2, a_1\}$
- Network D's possible schedules:
  - Schedule 1: $\{d_1, d_2\}$
  - Schedule 2: $\{d_2, d_1\}$

Now consider the following scenarios:

1. If Network A reveals Schedule 1 ($\{a_1, a_2\}$) and Network D reveals Schedule 1 ($\{d_1, d_2\}$):

- Slot 1: $a_1(20)$ vs $d_1(10) \Rightarrow$ Network A wins
- Slot 2: $a_2(40)$ vs $d_2(30) \Rightarrow$ Network A wins

In this case, Network A wins both slots. To avoid losing all slots, Network D will want to switch the order of its shows to:

$$\{d_2, d_1\}$$

2. If Network A reveals Schedule 2 ($\{a_2, a_1\}$) and Network D reveals Schedule 2 ($\{d_2, d_1\}$):

- Slot 1: $a_2(40)$ vs $d_2(30) \Rightarrow$ Network A wins
- Slot 2: $a_1(20)$ vs $d_1(10) \Rightarrow$ Network A wins

In this case, Network A wins both slots again. To avoid this, Network D will want to switch back to its original order:

$$\{d_1, d_2\}$$

3. If Network A reveals Schedule 1 ($\{a_1, a_2\}$) and Network D reveals Schedule 2 ($\{d_2, d_1\}$):

- Slot 1: $a_1(20)$ vs $d_2(30) \Rightarrow$ Network D wins
- Slot 2: $a_2(40)$ vs $d_1(10) \Rightarrow$ Network A wins

In this case, each network wins one slot. However, Network A will want to switch its schedule to:

$$\{a_2, a_1\}$$

to win both slots.

4. If Network A reveals Schedule 2 ($\{a_2, a_1\}$) and Network D reveals Schedule 1 ($\{d_1, d_2\}$):

- Slot 1: $a_2(40)$ vs $d_1(10) \Rightarrow$ Network A wins
- Slot 2: $a_1(20)$ vs $d_2(30) \Rightarrow$ Network D wins

Again, each network wins one slot. Network D will want to switch its schedule to:

$$\{d_2, d_1\}$$

to avoid losing both slots.

This example shows that there is no stable pair of schedules, as each network will always want to switch its schedule to improve its outcome.

1 Problem 2

Part 1

Let A and B be sets. We can prove that $A \times B = B \times A$ if and only if $A = \emptyset$ or $B = \emptyset$ or $A = B$.

(a) Proof by Contraposition

If $A \times B = B \times A$, then $A = \emptyset$ or $B = \emptyset$ or $A = B$.

We will prove the contrapositive: If $A \neq \emptyset$, $B \neq \emptyset$, and $A \neq B$, then $A \times B \neq B \times A$.

Assume $A \neq \emptyset$, $B \neq \emptyset$, and $A \neq B$. Let $a \in A \setminus B$ and $b \in B \setminus A$. Then, $(a, b) \in A \times B$ but $(a, b) \notin B \times A$. Hence, $A \times B \neq B \times A$.

(b) Direct Proof

If $A = \emptyset$ or $B = \emptyset$ or $A = B$, then $A \times B = B \times A$.

If $A = \emptyset$ or $B = \emptyset$, then $A \times B = \emptyset = B \times A$. If $A = B$, then $A \times B = A \times A = B \times A$.

Part 2

Let x and y be positive real numbers. Prove by contradiction: If $x^2 - y^2 = 1$, then x or y (or both) are not integers.

Assume x and y are integers. Then $x^2 - y^2 = (x - y)(x + y) = 1$. Since x and y are positive, $x - y = 1$ and $x + y = 1$, which implies $2x = 2$ and $x = 1$. This leads to $y = 0$, contradicting the assumption that y is positive. Thus, x or y (or both) are not integers.

2 Problem 3

Using induction:

Part 1

Prove that $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$.

Base Case

For $n = 1$,

$$1 = \frac{1(1+1)}{2} = 1.$$

Inductive Step

Assume $1 + 2 + \cdots + k = \frac{k(k+1)}{2}$ for some k. Then,

$$1+2+\cdots+k+(k+1) = \frac{k(k+1)}{2}+(k+1) = \frac{k(k+1)+2(k+1)}{2} = \frac{(k+1)(k+2)}{2}.$$

Thus, $1 + 2 + \cdots + (k+1) = \frac{(k+1)((k+1)+1)}{2}$, completing the induction.

Part 2

Find the sum of $1^2 + 2^2 + 3^2 + \cdots + n^2$.

Using the result of the previous part, we write:

$$(1 + 2 + \cdots + n)^2 = \left(\frac{n(n+1)}{2}\right)^2.$$

Thus, the sum of squares is:

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}.$$

Problem 4

Proof of Correctness

To prove the correctness of the given algorithm, we will use a loop invariant and induction approach for both loops.

Algorithm 1 smallest_with_min_freq

```
Input: An array A
Output: The smallest element with the minimum frequency
frequency ← {}
for each num in A do
   if num in frequency then
      frequency[num] ← frequency[num] + 1
   else
      frequency[num] ← 1
   end if
end for
min_freq ← min(frequency.values())
smallest_element ← inf
for each num, freq in frequency.items() do
   if freq = min_freq and num ¡ smallest_element then
      smallest_element ← num
   end if
end for
return smallest_element
```

First Loop: Frequency Calculation

Loop Invariant: At the start of each iteration i (where $0 \leq i \leq n$), the `frequency` dictionary contains the correct count of all elements in the subarray $A[0 : i]$.

Base Case: Before the first iteration ($i = 0$), `frequency` is an empty dictionary. This correctly represents the frequency counts for the empty subarray $A[0 : 0]$, so the invariant holds.

Inductive Hypothesis: Assume that at the start of iteration i, the loop invariant holds. That is, `frequency` correctly represents the counts of all elements in $A[0 : i]$.

Inductive Step: In iteration i, the loop processes the element $A[i]$:

- If $A[i]$ is already in `frequency`, the algorithm increments the count by 1.
- If $A[i]$ is not in `frequency`, the algorithm adds $A[i]$ with a count of 1.

After this iteration, `frequency` correctly represents the counts of all elements in $A[0 : i + 1]$. Thus, the loop invariant is maintained.

Conclusion: By the loop invariant and induction, after n iterations, `frequency` correctly contains the counts of all elements in A.

Second Loop: Finding the Smallest Element with Minimum Frequency

Loop Invariant: At the start of each iteration j (where $0 \leq j \leq m$), `smallest_element` is the smallest element with the minimum frequency among the first j elements of `frequency`.

Base Case: Before the first iteration ($j = 0$), `smallest_element` is initialized to ∞. This trivially satisfies the invariant because no elements have been processed yet, and any actual number will be less than ∞.

Inductive Hypothesis: Assume that at the start of iteration j, the loop invariant holds. That is, `smallest_element` is the smallest element with the minimum frequency among the first j elements of `frequency`.

Inductive Step: In iteration j, the loop processes the j-th element $(\text{num}, \text{freq})$ from `frequency`:

- If freq equals `min_freq` and num is smaller than `smallest_element`, then `smallest_element` is updated to num.

After this iteration, `smallest_element` is still the smallest element with the minimum frequency among the first $j + 1$ elements. Thus, the loop invariant is maintained.

Conclusion: By the loop invariant and induction, after m iterations, `smallest_element` correctly contains the smallest element with the minimum frequency in A.