CS 161 HW1

Vincent Quan Thai

April 14, 2023

1.

2.

This problem can be solved using the Pigeonhole Principle. The Pigeonhole Principle states that if there are more people than possible outcomes, then it is not possible for every person to have a unique outcome not shared by others. In other words, at least 2 guests will share the same outcome.

In the specific problem, we have 81 total guests, including ourselves. As party organizer, we shook hands with everybody; thus the range of possible handshake count is [1, 80] as everybody is guaranteed at least 1 handshake with ourselves. This gives us a total of 80 different outcomes for handshake count.

However, there are a total of 81 guests. This number is greater than the amount of possible outcomes. We would run out of values if we tried to map everyone to a different handshake count. Thus, we are forced to reuse at least one value in the range of possible handshakes, which guarantees at least 2 guests had the same number of handshakes at the function.

The result can also be described as the function that maps guests to handshake count not being an injective function due to the size of the set of guests being larger than the size of the set of handshake counts. As such, at least 2 domain values will map to the same target value.

3.

Program P1: The phrase "hi" is printed $\lfloor \frac{n}{2} \rfloor$ times as the print statement only executes when i is even, which happens half of the time for values in range [1, n]. We take the floor because it's not possible to execute the statement 0.5 times.

Program P2: The phrase "hi" is printed $\sum_{i=1}^{n}(n-i)$ times. i is in the range [1, n] which means the outer loop executes n times. This means that inside the inner loop, starting at $i = 2$ we print "hi" 1 less time as we increment i, as the difference between n and i gets smaller. Thus, for every value of i, we print "hi" $n - i + 1$ times. In addition, the

inner loop has j starting at $j = i + 1$, which makes "hi" print 1 less time in addition as j is closer to n than i by 1. This means "hi" gets printed $n - i + 1 - 1$ times for every value i, which equals $n - i$ times. To get the total sum of times "hi" is printed, we simply sum the results from the formula n times.

Program P3: Do the matrix approach where the rows get progressively longer

Program P4: The phrase "hi" is printed $n(n - 1)$ times. We can visualize the nested loop in the program as a 2D array, where i represents the current row and j represents the current column, and i, j are in the range [1,n]. Let every row in the matrix be a sequence from [1, n] where the value at a column is equal to j, or which column the value is at. We notice that the main diagonal of the matrix is where the value is equal to row number, or where i == j. This means that for every row, or every iteration of the inner loop, we print "hi" exactly n - 1 times as the main diagonal represents the 1 instance that does not pass the if statement. Multiply that by the number of rows, n, and we get our equation $n(n - 1)$