You have 120 minutes to answer four questions. There is no question 3.

Write your answers in the separate answer booklet.

Please return this question sheet and your cheat sheet with your answers.

1. A line of n students and teachers is waiting to board the Hogwarts Express, which has exactly n seats, each one assigned to a different passenger. The first passenger in line is Professor Dumbledore, who has forgotten his assigned seat. When the professor boards the train, he chooses a seat uniformly at random and sits there.

   The remaining $n-1$ passengers then board the train one at a time. When each passenger boards, if Dumbledore is sitting in their assigned seat, he apologetically chooses a different *unoccupied* seat, again uniformly at random, and moves to sit there. The next passenger boards only after the previous passenger and Dumbledore are seated. After all n passengers have boarded, everyone is in their assigned seat, including Dumbledore.

   (a) What is the exact probability that Dumbledore never moves after choosing his first seat?

   (b) What is the exact probability that the kth passenger has to ask Dumbledore to move? *[Hint: Consider the special cases $k = 2$ and $k = n$, in particular when $n = 3$.]*

   (c) What is the exact expected number of times Dumbledore changes seats?

2. **Part (c) of this problem was removed from the exam, because the intended solution was incorrect. Everybody received full credit for that part.**

   Two strings are *almost equal* if one can be transformed into the other by inserting, deleting, or changing *at most one* character. More concisely, two strings are almost equal if their edit distance is at most 1. For example, HEEDLESS is almost equal to HEADLESS, and HUNTED is almost equal to HAUNTED.

   A string P is an *almost-substring* of another string T if P is almost equal to a substring of T. For example, KMP is an almost-substring of STICKPREFIXESTOSUFFIXES, and POOKA and SCALY and SCARS and TOONS are all almost-substrings of SPOOKYSCARYSKELETONS.

   Describe and efficient analyze algorithms for each of the following problems. The input to each problem is a pair of strings $P[1..m]$ and $T[1..n]$.

   (a) For each index i, find the longest prefix of P ending at $T[i]$.

   (b) For each index i, find the longest suffix of P starting at $T[i]$.

   (c) Determine whether P is an almost-substring of T.

[Questions 4 and 5 are on the back of this page.]

4. Every Halloween, hundreds of ghosts rise from their graves and haunt the houses of Sham-Poobanana. This is not as straightforward as it sounds, for two reasons. First, each ghost can only haunt houses where they spent significant time when they were alive. Second, at most one ghost can haunt each house. There are n ghosts and m houses.

   (a) Describe and analyze an efficient algorithm that either assigns each ghost a distinct house that they can haunt, or correctly reports that such an assignment is impossible. Your input is a two-dimensional boolean array $CanHaunt[1..n, 1..m]$, where $CanHaunt[i, j] = \text{TRUE}$ if and only if ghost i can haunt house j.

   (b) Oh, no! Beetlejuice broke into the main office and assigned each ghost to a house they can't haunt! Halloween is ruined! Stay-Puft proposes the following strategy to fix everyone's assignments. At exactly midnight, each ghost will give their assigned house to another ghost that actually wants it. For example, suppose

   - Agnes was assigned house A, but she can only haunt houses C and D.
   - Banquo was assigned house B, but he can only haunt houses A and C.
   - Casper was assigned house C, but he can only haunt houses A and D.
   - Daayan was assigned house D, but she can only haunt houses B and C.

   The ghosts can fix their assignment as follows: Agnes gives house A to Banquo; Banquo gives house B to Daayan; Casper gives house C to Agnes, and Daayan gives house D to Casper.

   Describe and analyze an efficient algorithm to compute an exchange that results in a valid assignment of ghosts to houses. The input to your algorithm is the Boolean array $CanHaunt$ from part (a) and a second array $Haunt[1..n]$, where $Haunt[i]$ is the index of the house originally assigned to ghost i. The correct output is an array $GiveTo[1..n]$, where $GiveTo[i] = j$ means ghost i should give their house to ghost j.

   You can assume that $CanHaunt[i, Haunt[i]] = \text{FALSE}$ for every index i (that is, no ghost can haunt their assigned house) and that a valid exchange exists. **Assume also that $m = n$.**

5. Your friends are organizing a board game party, and because they admire your personal dice collection, they ask you to bring dice. You choose several beautiful, perfectly-balanced, six-sided dice to bring to the party. Unfortunately, by the time you arrive, the other guests have already chosen a game ("Let's Summon Demons") that requires 20-sided dice! So now you get to improvise.

   (a) Describe an algorithm to simulate one roll of a fair 20-sided die using independent rolls of a fair 6-sided die *and no other source of randomness*. Equivalently, describe an implementation of RANDOM(20), whose only source of randomness is an implementation of RANDOM(6).

   (b) What is the *exact* expected number of 6-sided-die rolls (or calls to RANDOM(6)) executed by your algorithm?

   (c) Derive an upper bound on the probability that your algorithm requires more than N rolls. Express your answer as a function of N.

   (d) Estimate the smallest number N such that the probability that your algorithm requires more than N rolls is less than δ. Express your answer as a function of δ.