Please read the instructions carefully:

1. There are two stages to each assignment:

   (a) For the first stage, you need to submit your homework solution as a single pdf file to NYU Classes. It can be either a scanned copy of your written solution or photos joined into a pdf. You are also allowed to typeset the solutions in MS Word/LaTeX. In case of scanned copies it is your responsibility that the solution is readable. There are no late submissions accepted, as sample solution outlines (to be used for the second stage of the homework) will be given out on the day of the submission of the first stage of the homework. You should leave around 5 to 6 cm of space between each part of the answers for the second stage of the homework.

   (b) For the second stage of the assignment, you are required to critique your own solutions and analyze the mistakes that you made. Based on the outlines provided, you need to write the self-criticism in the empty space that you left in your homework. Do not rewrite your solution, you need to understand and point out the mistakes that you made when writing the original solutions.

   A good analysis of your solution should summarize the key steps, ideas or techniques that you previously missed. It should prompt you to reorganize your thoughts and think about what you can do to improve the solutions. Even in the case where you completely missed the solution it is insightful to point out why the approach adopted by you does not work and identify the step where it fails. This will allow you to get full credit on the self-evaluation part of the homeworks and of course help you in understanding the question.

   Mark correct solutions in green, incorrect solutions in red and if you are unsure about any solutions write it in blue.

2. For each question where you are required to design an algorithm, you need to clearly explain the algorithm and prove the running time complexity as asked in the question.

3. You can assume $O(1)$ complexity of arithmetic of numbers (e.g. multiplication/division of numbers and calculations of $n!$) unless otherwise stated in the question.
Question 1. (15 points) Picking a stream
We are given an input stream of integers, \( a_1, a_2, \ldots \), out of which we want to pick a number at random, with uniform probability. But the catch here is that we do not know how many numbers are going to be there in the stream.
Give an algorithm which returns a random number from the stream, with uniform probability and uses constant amount of memory.

Note 1.1. Assume that we can generate a random number in any range, in constant time.

Question 2. (15 points) Valleys in a permutation
Given a permutation, \( \pi \), of the numbers \( \{1, 2, \ldots, n\} \), we will say that index \( i \) is a valley, if \( \pi_i \) is smaller than its neighbours. E.g. in the permutation
\[
5, 3, 4, 2, 1
\]
the positions 2 and 5 (with the numbers 3, 1, resp.) are valleys.
Find the expected number of valleys for a random permutation, where the permutations have a uniform distribution.

Hint 2.1. Use an indicator variable, \( 1_i \), which recognizes the event that index \( i \) is a valley.

Hint 2.2. Use linearity of expectation.

Question 3. (15 points) Double hashing
Let \( \{h_k\}_{k \in K} \) be an \( \epsilon \)-universal family of hash functions from \( U \) to \( V \) (that is, \( h_k : U \to V \) for each \( k \in K \)). Let \( \{\phi_\lambda\}_{\lambda \in \Lambda} \) be a \( \delta \)-universal family of hash functions from \( V \) to \( W \). Show that
\[
\{\phi_\lambda \circ h_k\}_{(k, \lambda) \in K \times \Lambda}
\]
is an \( \epsilon + \delta \) universal hash family from \( U \) to \( W \).
Here, the function \( \phi_\lambda \circ h_k \) maps \( a \in U \) to \( \phi_\lambda(h_k(a)) \in W \).

Question 4. (15 points) Every minimum cut
Prove that in any graph the number of minimum cuts is \( \leq n(n-1)/2 \).

Hint 4.1. Use the analysis of Karger’s original algorithm, without the branching amplification.

Question 5. (15 points) The unbearable isolation of being
Let us construct a random undirected graph \( G \) on \( n \) vertices, \( \{1, \ldots, n\} \), such that each edge \( (i, j) \) is present in the edge with probability \( p \).
Let \( 1_i \) be the indicator variable for the event: vertex \( i \) is an isolated vertex, i.e. there is no edge from \( i \). Define \( I = \sum_{1 \leq i \leq n} 1_i \), which denotes the number of isolated vertices in this graph.
Find \( E[I] \), \( \text{Var}(I) \).

Question 6. (20 points) Longest palindromic substring
You are to design an algorithm for the following problem. The input is an array \( A[1 \ldots n] \) of 8-bit ASCII characters. A palindromic substring of \( A \) is a substring \( A[i \ldots j] \) of \( A \) that is a palindrome, i.e. it reads the same forwards or backwards. Of course, any substring of length 1 is a palindromic
substring. Your algorithm is to find a longest palindromic substring. The output of your algorithm is just the indices \( i \) and \( j \) that define the substring \( A[i \ldots j] \).

Your algorithm should be a Las Vegas algorithm and should run in expected time \( O(n \log n) \). Your algorithm should also use just constant space - not including the input array \( A \) (which is read only). Your algorithm should use universal hashing. Here are the rules of the game, in terms of counting time and space. Calculations on array indices and reading an element of \( A \) cost 1 time unit. Storage for an array index costs 1 space unit. To implement a hash function, your algorithm may choose a prime \( p \) of any bit length, as long as that bit length is \( O(\log n) \). You are charged 1 time unit for selecting \( p \). Given such a prime \( p \), each of the following operations cost 1 time unit:

- generating a random element in \( \mathbb{Z}_p \)
- addition, subtraction, multiplication, or division in \( \mathbb{Z}_p \)

Also, you are charged 1 space unit for each element of \( \mathbb{Z}_p \) that your algorithm stores in memory.

**Hint 6.1.**

- Use a variation of the Rabin-Karp pattern matching algorithm.
- Try solving the following problem first: given \( A[1 \ldots n] \) as above and \( l \in \{1, \ldots, n\} \), determine if \( A \) has a palindromic substring of length \( l \) in expected time \( O(n) \) and constant space.

**Note 6.1.** There is actually a deterministic \( O(n) \) time algorithm for this, but it uses additional space \( O(n) \).

**Question 7. (20 points) Fraud detection**

On auction websites, a common scam is the making of duplicate accounts to raise bids on sales. One way to detect duplicate accounts is to find the friends of each account and see which of them have the same friends. We will ban two accounts if they have the exact same set of friends (excluding each other). E.g. in the graph in fig. 1, \((4,5)\) and \((6,7)\) are accounts that we will ban.

Given a graph \( G = (V, E) \), the nodes denote accounts and an edge \((u, v)\) means that \( u \) and \( v \) are friends.

Given a Monte Carlo algorithm which finds the number of unordered pairs \((u, v)\) that we have to ban in \( O(n \log n) \), which succeeds with a probability of at least \( 1 - \frac{1}{n} \).

**Note 7.1.** This problem has the same constraints on calculations in \( \mathbb{Z}_p \) and other unitary operations as in the previous problem, for calculating the time complexity.

**Hint 7.1.**

- Try to modify Rabin-Karp pattern matching algorithm to this problem.
- When would you get an incorrect answer? Try to use Union Bound to get an upper bound on the error probability.
- What is the size of the prime that you need for the probability of success to be large?
Question 8. (25 points) Querying strings

For two binary strings, $A, B$, of equal length, $n$, we define the similarity of the strings as

$$\text{sim}(A, B) = \sum_{i=1}^{n} 1_{A[i] = B[i]}$$

We know that there is a hidden binary string $S$ of length $n$, which we want to find. We also know that $n$ is even. The only thing we can do, to guess the value of $S$, is to call the function $\text{reducedSim}(T)$ for our choice of a binary string $T$, where the $\text{reducedSim}$ function is defined as

$$\text{reducedSim}(T) = \begin{cases} \text{sim}(T, S) & \text{if } \text{sim}(T, S) = n \text{ or } \text{sim}(T, S) = n/2 \\ 0 & \text{otherwise} \end{cases}$$

Which means that the only values of $\text{sim}$ we can see with $\text{reducedSim}$ are $n$ (which would tell us that we have found $S$) and $n/2$.

We will design a randomized algorithm to find the hidden string in $O(n^2)$. The algorithm is broken down into two parts.

1. In the first part of the algorithm, we want to find a string, $T$, with $\text{reducedSim}(T) = n/2$.

   (a) (5 points) What is the probability that a randomly binary string, $T$, of length $n$ will have $\text{sim}(T, S) = n/2$?

   Hint 8.1. Use Stirling’s approximation

   $$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

   (b) (10 points) Give a Monte-Carlo randomized algorithm which finds a binary string, $T$, with reduced similarity $n/2$, in $O(n^3)$, with probability $\geq 1 - \frac{1}{e}$. 
2. (10 points) Once we have found a random string, \( T \), of reduced similarity \( n/2 \), we can now start modifying it to get the hidden string. Design an algorithm (deterministic) which find \( S \) from \( T \) in \( O(n^2) \).

**Question 9. (30 points - Extra credit) Encrypting parts**

For two strings \( X, Y \) (in lower case english alphabet), we will say that they are equivalent if there is a permutation of characters \( \pi \), which can let us transform \( X \) to \( Y \) by substituting all occurrences of character \( \alpha \) by \( \pi_\alpha \).

Some examples of equivalent strings are: \( aabb \) and \( ccdd \), \( aabbb \) and \( ccbbb \); while the following are not equivalent: \( abab \) and \( abcd \), \( acca \) and \( abca \).

Given an encrypted message, one of the most important things is to find patterns in the encryption so that we can get information about how to decrypt it. We want to know what sub-strings of \( m \) are equivalent to each other. More specifically, we are given \( Q \) queries of the form \((i, j, k, l)\) and we have to tell if the sub-strings \( m[i...j] \) and \( m[k...l] \) are equivalent or not. I.e. our algorithm should take in as input \( Q \) queries \([(i_1, j_1, k_1, l_1), \ldots, (i_Q, j_Q, k_Q, l_Q)] \) and produce a list \([a_1, \ldots, a_Q]\) where \( a_i = YES/NO \).

1. (10 points) We want to know when two sub-strings \( m[i...j], m[k...l] \) are equivalent. More specifically, let \( a \rightarrow \pi_a, b \rightarrow \pi_b, \ldots, z \rightarrow \pi_z \) be an assignment which proves equivalence of the two sub-strings. One way to find \( \pi_\alpha \) is to find the position, \( \zeta_\alpha \), of the first occurrence of \( \alpha \) in the sub-string \( m[i...j] \) and get \( \pi_\alpha = m[k + \zeta_\alpha - 1] \).

E.g. Let \( m = xbbaxggbx \), the query is \((1, 5, 5, 9)\). Here the sub-strings are \( xbbax, xggbx \), which are equivalent. Suppose we want to find \( \pi_a \), we find \( \zeta_a = 4 \), then we find \( \pi_a = m[5 + 4 - 1] = m[8] = b \), which is the correct substitution for \( a \).

Show how you can calculate \( \zeta_\alpha \) in \( O(1) \), given \( i, j \).

**Hint 9.1.** Keep \( \alpha \) fixed. Now use dynamic programming to calculate \( \zeta_\alpha \) for each index \( i \) (notice that it is independent of \( j \)).

How do you need to do this for each \( \alpha \in \{a, b, \ldots, z\} \)?

2. (20 points) Given that we know what character \( \alpha \) needs to be replaced by, we need to know if each position of \( \alpha \) corresponds to the positions of \( \pi_\alpha \) in the second sub-string (currently we are only sure about the first occurrence).

For this, create an array \( I_\alpha[1...n] \), such that

\[
I_\alpha[\xi] = \begin{cases} 
1 & \text{if } m[\xi] = \alpha \\
0 & \text{otherwise}
\end{cases}
\]

Now, equipped with the arrays \( I_\alpha \forall \alpha \in \{a, b, \ldots, z\} \), give a Monte Carlo algorithm which answers each query, with error \( \leq \frac{1}{n} \).

**Hint 9.2.** Use a method similar to the one that you developed for the previous two problems.