Part (a)

We are given $f(n) = \text{poly}(n)$ n-bit ciphertexts $c_1, \ldots, c_{f(n)}$, each from a unique plaintext/pad pair. Our goal is to show that the upper bound on the longest run of matching bits between any two ciphertexts, at any offset, is $\log_2 n + \log_2 \ln n = \log_2 (n \ln n)$ with high probability [Piazza:18].

First, let's consider a particular pair of ciphertexts c_i and c_j ($i \neq j$) with no offset. We can model the longest run of matching bits within the n bits of c_i and c_j as the longest run of heads in n coin flips, where a coin flip represents matching bits at the same position. Since each ciphertext is encrypted from a one-time pad (chosen independently and uniformly at random), `0` and `1` bits are equally likely. Therefore, the probability of matching bits at the same position is $\frac{1}{2} = 0.5$. Using the fact from the problem, the longest run $R_{0.5}(n)$ is, with high probability, less than $\log_{\frac{1}{0.5}} n + \log_{\frac{1}{0.5}} \ln n = \log_2 (n \ln n)$.

Next, let's consider a particular pair of ciphertexts offset by k bits ($0 \leq k < n$). There are $2n - 1$ possible offsets. We can model this as above, finding the upper bound for the longest run of matching bits within the $n - k$ bits of c_i and c_j. Since $n - k \leq n$, $R_{0.5}(n - k) \leq R_{0.5}(n) = \log_2 (n \ln n)$, so the upper bound from above still holds.

Finally, let's put it all together for the $f(n)$ ciphertexts. The longest run may occur in any of $\binom{f(n)}{2}$ pairs of ciphertexts, each at any of $2n - 1$ offsets. The number of pairs/offsets is $f'(n) = (2n-1)\binom{f(n)}{2} = (2n-1) \cdot \frac{1}{2} f(n)(f(n) - 1)$ which, since $f(n)$ is $\text{poly}(n)$, is also $\text{poly}(n)$. We next show that the $\log_2 (n \ln n)$ upper bound holds for all these pairs/offsets with high probability.

For each possible pair c_i and c_j and offset, the longest run common to c_i and c_j is less than $\log_2 (n \ln n)$ with high probability $p(n)$. That means that for any asymptotically positive polynomial $q(n)$, $p(n) > 1 - \frac{1}{q(n)}$, for all sufficiently large n [Piazza:14]. Let a polynomial $\hat{q}(n) = \Omega(f'(n))$. Then $p(n) > 1 - \frac{1}{\hat{q}(n)}$. The probability that the upper bound does not hold for a particular pair/offset is $1 - p(n) < \frac{1}{\hat{q}(n)}$. The probability that it does not hold for at least one pair/offset among the $f'(n)$ is, by union bound, less than $f'(n) \cdot \frac{1}{\hat{q}(n)} = \frac{1}{\hat{q}'(n)}$, where $\hat{q}'(n) = \frac{\hat{q}(n)}{f'(n)}$. $\hat{q}'(n)$ is an asymptotically positive polynomial because we chose $\hat{q}(n)$ to asymptotically dominate $f'(n)$ and both are positive. The probability that the upper bound holds for all pairs/offsets is $p'(n) \geq 1 - \frac{1}{\hat{q}'(n)}$. In fact, for any asymptotically positive polynomial $q(n)$, $p'(n) > 1 - \frac{1}{q(n)}$ because we can repeat the argument above for a sufficiently large $\hat{q}(n) \geq \frac{q(n)}{f'(n)}$.

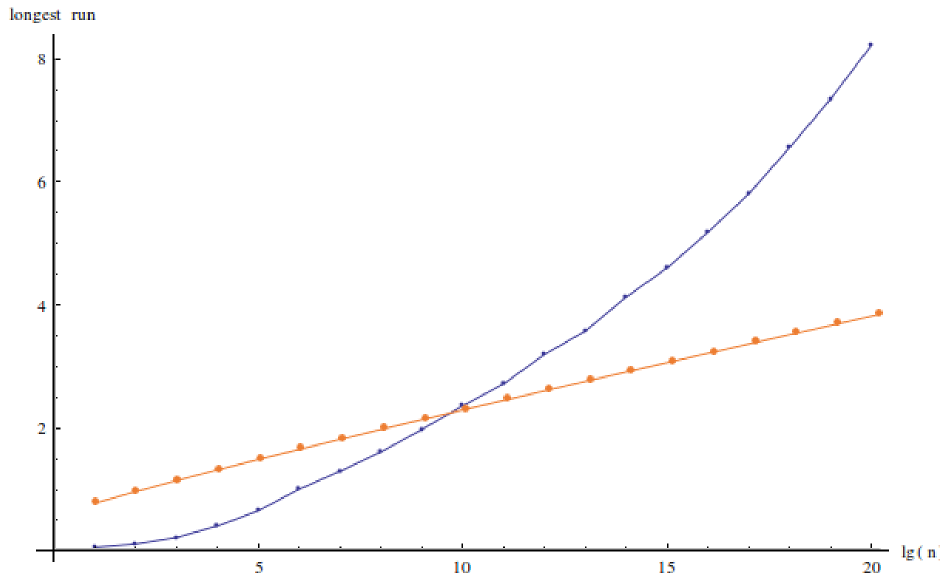
Therefore, the upper bound on the longest run of matching bits between any two ciphertexts, at any offset, is $\log_2 (n \ln n)$ with high probability $p'(n)$.

Part (b)

We plotted the upper bound of ciphertext from part (a) and superimposed it on the provided graph. We assume that the English plaintext is encoded as US-ASCII characters, each 7 bits long. Since n here is the text length (in characters), the upper bound of ciphertext (in characters) is $\frac{1}{7}\log_2(7n \ln 7n)$.

$\log_2 n$ vs. longest run.

Blue curve: English plaintext (average). Orange curve: ciphertext (upper bound).



The "random" ciphertext dominates at the beginning because there is more flexibility in matching bits than matching 7-bit US-ASCII characters: (1) the bits can match across 7-bit boundaries; (2) there are more characters to match in the plaintext: 26 English letters, plus punctuation; and (3) the ciphertext is an upper bound, while the plaintext is just an average. But, interestingly, the plaintext average dominates the ciphertext upper bound when $\log_2 n > 9$. We reason that matching characters occur more frequently there because the English language has a lot of redundancy: high frequency of `e`, the strong tendency for `h` to follow `t` or for `u` to follow `q`, etc. (formalized by Claude Shannon in "Prediction and Entropy of Printed English" in 1951).

Therefore, for sufficiently large n, the average longest runs within English plaintext are larger than even the upper bound on longest runs within ciphertext. On the graph above, it appears as exponential vs. linear.

Part (c)

We are given $f(n) = \text{poly}(n)$ n-bit ciphertexts $c_1, \ldots, c_{f(n)}$, and two of them reuse the same pad. Our goal is to find those those ciphertexts in quasilinear time in $N = |c_1 \cdots c_{f(n)}| = n \cdot f(n)$.

Idea We assume that each pair of ciphertexts, at any offset, with independently chosen one-time pads do not share a long common substring [Piazza:11]. From part (a), the length of a common substring is less than $\log_2(n \ln n)$ bits with high probability. We assume that each pair of plaintexts share a long common substring [Piazza:11]. Therefore, if ciphertexts corresponding to the plaintexts are encrypted with the same pad, then they also share a long common substring. The graph from part (b) shows that, for sufficiently large n, its length is greater than $\log_2(n \ln n)$. Therefore, we can take $\log_2(n \ln n)$ as the separator between pad reuse and no pad reuse.

Algorithm In the following algorithm, we find the longest common substring between any two ciphertexts and detect pad reuse if its length is $> \log_2(n \ln n)$.

1. Let the alphabet $\Sigma = \{\texttt{0}, \texttt{1}, \$_1, \ldots, \$_{f(n)}\}$, which represents the bits and $f(n)$ special terminators. Append the terminators to the ciphertexts, and concatenate them into one string $S = c_1 \$_1 \cdots c_{f(n)} \$_{f(n)}$.
2. Construct a suffix tree for S.
3. Traverse the suffix tree to find the internal node with the highest depth d.
4. If $d \leq \log_2(n \ln n)$, then output *no pad reuse*. Otherwise, follow a path of edges from the node down (i.e., toward a leaf) until an edge label contains a special terminator $\$_i$. This path corresponds to a suffix of ciphertext c_i. Since an internal node must have ≥ 2 children, repeat this process along a different path—until an edge label contains $\$_j$. Output c_i and c_j.

Correctness Using the procedure stated in the problem, Step 3 finds the longest repeated substring of S. In a suffix tree, the depth d corresponds to its length. There are three cases of where it might occur in S:

1. *In different ciphertexts*, a substring of c_i and a substring of c_j:
   This case is desired, because it corresponds to the longest common substring of a pair of ciphertexts. Correctness follows from the idea paragraph. If there is no pad reuse, then $d \leq \log_2(n \ln n)$. Otherwise there is, and the special terminators $\$_i$ and $\$_j$ identify the ciphertexts c_i and c_j that reuse the same pad.
2. *In the same ciphertext*, a substring of c_i and a substring of c_i:
   This case is not desired, because it corresponds to longest common substring between a ciphertext and itself at an offset of k bits. However, we can model this using part (a), since the successive bits of the pad are independent. The longest run of matching bits within the $n-k$ bits of c_i and the offset c_i is less than $R_{0.5}(n-k) \leq R_{0.5}(n) = \log_2(n \ln n)$. This correctly corresponds to no pad reuse.
3. *Across ciphertexts*, a substring of $c_i \$_i c_j$ and a substring of S:
   This case never occurs because the special terminator $\$_i$ only appears once in S.

Running Time Step 1 (concatenating the ciphertexts) takes $O(N)$ time. Step 2 (building the suffix tree) takes $O(N \log |\Sigma|) = O(N \log f(n)) = O(N \log N)$ time [Piazza:25]. Step 3 (traversing for longest repeated substring) takes $O(N)$ time, as stated in the problem. Step 4 (traversing to identify c_i and c_j) takes $2 \cdot O(n)$ time, since the internal node is $\leq n$ edges away from an edge label containing the special terminator. This dominates computing $\log_2(n \ln n)$, which involves operations on $\log_2 n$-bit numbers. Therefore, the total time is $O(N \log N)$, quasilinear in N.