1. CLR Problem 27-1 (page 625).

2. Recall from Lecture 13, Karger’s min-cut algorithm:

   While number of vertices > 2:
   
   Select edge \( e = (u, v) \) uniformly at random,
   
   Merge \( u \) and \( v \) into a single vertex.
   
   Return remaining edges.

   We proved that this returns a minimum cut in the original graph with probability at least \( \frac{1}{n^2} \), where \( n = |V| \).

   (a) Describe a data structure that allows each iteration of the while loop to be performed in \( O(n) \) time, and justify this bound. Describe how this allows us to find a minimum cut with probability at least \( 1 - \epsilon \) in time \( O(n^4 \log \epsilon^{-1}) \).

We next consider how to improve on this run time. The idea behind this improvement is that the probability of the min-cut algorithm hitting an edge in the minimum cut is fairly small for the first few edges that are chosen, but increases with each merge that is performed. Thus, we should be repeating the merges at the later stages of the algorithm more times than the merges at the early stages.

Let us consider how many merges we can perform before the total probability of hitting a particular minimum cut \( C \) is \( \frac{1}{2} \). From the argument in Lecture 13, the probability that \( C \) survives down to \( \ell \) vertices is at least \( \frac{6(\ell - 1)}{n(n-1)} \). You should verify this (but you don’t need to turn this in.) If we take \( \ell = \lceil n/\sqrt{2} \rceil \), this probability is \( \frac{1}{2} - O(1/n) \). Therefore, we need only to repeat these early stages \( \log \epsilon^{-1} \) times (rather than \( n^2 \log \epsilon^{-1} \) times, as in part (a)) for \( C \) to survive down to \( \lceil n/\sqrt{2} \rceil \) vertices at least once with probability \( \epsilon \). To finish off the process, we can then apply the same procedure recursively to the remaining \( \lceil n/\sqrt{2} \rceil \) vertex graph. The resulting algorithm, Rec-MinCut, is as follows:

\[
\text{Rec-MinCut}(G)
\]

If number of vertices = 2 return edges of \( G \).

Else repeat twice

   By repeated merging, reduce \( G \) to an \( \lceil n/\sqrt{2} \rceil \) vertex graph \( G' \).

   Rec-MinCut(\( G' \))

   Return the smaller of the two cuts returned by the recursive calls.

(b) Write down a recurrence relation for the running time, \( T(n) \) of Rec-MinCut on a graph with \( n \) vertices. Use this recurrence relation to show that \( T(n) = O(n^2 \log n) \). To do so, you can ignore floors and ceilings.

(c) In an \( n \)-vertex graph \( G \), let \( P(n) \) be the probability that a particular minimum cut survives to the end of Rec-MinCut(\( G \)). Show that \( P(n) \) satisfies the recurrence relation

\[
P(n) = P \left( \lceil \frac{n}{\sqrt{2}} \rceil \right) - \frac{1}{4} P \left( \lceil \frac{n}{\sqrt{2}} \rceil \right)^2 - O(\frac{1}{n}), \text{ for } n > 2,
\]

and \( P(2) = \Theta(1) \).
3. Consider the following problem. Alice has an n-bit string $x$, and Bob has an n-bit string $y$. They wish to know if $x = y$, but can only communicate over a slow communication channel. If $n$ is small, then Alice could send her string over the channel, and then Bob could simply compare $x$ and $y$. However, for large $n$, they wish to be more efficient. Thus, they shall use the following protocol, where Alice and Bob interpret their strings as binary integers:

1) Bob selects a prime $p$ in the range $[2 \ldots r]$, for $r$ to be determined.
2) Bob computes $y' = y \mod p$, and sends Alice $y'$ and $p$.
3) Alice computes $x' = x \mod p$.
4) If $x' = y'$ Alice says $x = y$. Otherwise, Alice says $x \neq y$.

(a) Describe a value of $r$ such that the probability of Alice giving an incorrect answer is at most $\frac{1}{2}$ and the total number of bits transmitted is $O(\log n)$. You can use the fact that there are at least $M$ primes in the range $[2 \ldots 2M \log M]$.

Hint: note that $x' = y'$ only if either $x = y$ or $p$ divides $|x - y|$.

(b) Describe how Bob can perform Step 1 in time $O(\sqrt{n}(\log n)^c)$. Also, describe a slightly revised algorithm that allows Bob to perform Step 1 in time $O((\log n)^d)$, for a constant $d$. Is the probability of an error in the revised algorithm smaller, larger or the same as the probability of an error in the original algorithm?

(c) Alice and Bob decide that an error probability of $1/2$ is too large. How should this protocol be modified so that the error probability is at most $2^{-100}$?