1. **Minimum k-cut**

   To lower bound the number of edges in the graph, one possible way is to sum over all possible "trivial" k-cuts (i.e. $k-1$ singletons and the complement) and count how many times an edge is (over-)counted. You may also need to pay attention when the graph has become small enough (say only $2k$ vertices left), but then you can finish the proof by some simple argument.

2. **Quicksort**

   The idea is to consider the recursion tree, and analyze the probability of having a long execution path. A path is long only if we split the numbers unevenly many times.

3. **k-wise Independence**

   This is simple once you figure out what is the precise statement to prove.

4. **Almost k-wise Independence**

   Use probabilistic method (i.e. choose $y_1, \ldots, y_m$ be random strings) and apply Chernoff bound to bound the probability that they are not almost k-wise independent.

5. **Online Hiring**

   The first part is not too difficult. In the second part, it is okay to show that the probability tends to $1/e$ when n tends to infinity.

6. **Graph Drawing**

   Sample each vertex with an appropriate probability p, and then apply the simple bound.

7. **Algebraic Matching**

   Use the idea in parallel matching to encode the weight in the degree of a variable (and this is the reason why we need to consider matrices where each entry is a single variate polynomial). Do similar thing to encode the color information. You can assume the weights are integers.

8. **Network Coding**

   The proof is similar to that in notes. Just need to understand and write them out. The fast algorithm that I expect is to use some "gadget" that we've seen.

9. **Graph Coloring**

   Once you figure out the right bad events to analyze, the proof is straightforward. In the algorithm, be faithful to the bad events in the analysis (i.e. don't be distracted by the graph structure of the problem).

10. **Fractional Flow**

    Modify the random walk algorithm for regular bipartite matching. You need to be careful in defining the residual graphs after you sent some flow. Try to follow the traditional augmenting path algorithm as closely as possible (e.g. no need to contract vertices as in the notes, don't delete the paths found, etc). May need to use some data structure to implement it efficiently.