You have 120 minutes to answer all five questions.
Write your answers in the separate answer booklet.
Please turn in your question sheet and your cheat sheet with your answers.

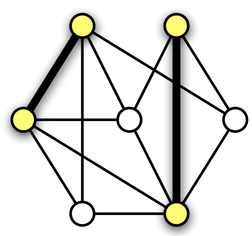
1. Consider the following modification of the 'dumb' 2-approximation algorithm for minimum vertex cover that we saw in class. The only change is that we output a set of edges instead of a set of vertices.

```
APPROXMINMAXMATCHING(G):
  M ← ∅
  while G has at least one edge
      let (u, v) be any edge in G
      remove u and v (and their incident edges) from G
      add (u, v) to M
  return M
```

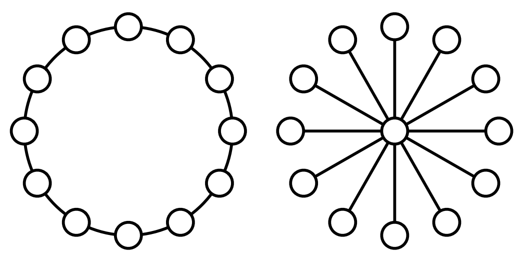
   (a) ***Prove*** that this algorithm computes a *matching*—no two edges in M share a common vertex.

   (b) ***Prove*** that M is a *maximal* matching—M is not a proper subgraph of another matching in G.

   (c) ***Prove*** that M contains at most twice as many edges as the *smallest* maximal matching in G.



The smallest maximal matching in a graph.



A cycle and a star.

2. Consider the following heuristic for computing a small vertex cover of a graph.

   - Assign a random *priority* to each vertex, chosen independently and uniformly from the real interval $[0, 1]$ (just like treaps).
   - Mark every vertex that does *not* have larger priority than *all* of its neighbors.

   For any graph G, let $OPT(G)$ denote the size of the smallest vertex cover of G, and let $M(G)$ denote the number of nodes marked by this algorithm.

   (a) ***Prove*** that the set of vertices marked by this heuristic is *always* a vertex cover.

   (b) Suppose the input graph G is a *cycle*, that is, a connected graph where every vertex has degree 2. What is the expected value of $M(G)/OPT(G)$? ***Prove*** your answer is correct.

   (c) Suppose the input graph G is a *star*, that is, a tree with one central vertex of degree $n-1$. What is the expected value of $M(G)/OPT(G)$? ***Prove*** your answer is correct.

3. Suppose we want to write an efficient function SHUFFLE($A[1..n]$) that randomly permutes the array A, so that each of the $n!$ permutations is equally likely.

   (a) ***Prove*** that the following SHUFFLE algorithm is ***not*** correct. *[Hint: There is a two-line proof.]*

   ```
   SHUFFLE(A[1..n]):
     for i = 1 to n
         swap A[i] ⟷ A[RANDOM(n)]
   ```

   (b) Describe and analyze a correct SHUFFLE algorithm whose expected running time is $O(n)$.
   Your algorithm may call the function RANDOM(k), which returns an integer uniformly distributed in the range $\{1, 2, \ldots, k\}$ in $O(1)$ time. For example, RANDOM(2) simulates a fair coin flip, and RANDOM(1) *always* returns 1.

4. Let Φ be a legal input for 3SAT—a boolean formula in conjunctive normal form, with exactly three literals in each clause. Recall that an assignment of boolean values to the variables in Φ ***satisfies*** a clause if at least one of its literals is TRUE. The ***maximum satisfiability problem***, sometimes called MAX3SAT, asks for the maximum number of clauses that can be simultaneously satisfied by a single assignment. Solving MAXSAT exactly is clearly also NP-hard; this problem asks about approximation algorithms.

   (a) Let $\mathit{MaxSat}(\Phi)$ denote the maximum number of clauses that can be simultaneously satisfied by one variable assignment. Suppose we randomly assign each variable in Φ to be TRUE or FALSE, each with equal probability. ***Prove*** that the expected number of satisfied clauses is at least $\frac{7}{8}\mathit{MaxSat}(\Phi)$.

   (b) Let $\mathit{MinUnsat}(\Phi)$ denote the *minimum* number of clauses that can be simultaneously *unsatisfied* by a single assignment. ***Prove*** that it is NP-hard to approximate $\mathit{MinUnsat}(\Phi)$ within a factor of $10^{10^{100}}$.

5. Consider the following randomized algorithm for generating biased random bits. The subroutine FAIRCOIN returns either 0 or 1 with equal probability; the random bits returned by FAIRCOIN are mutually independent.

   ```
   ONEINTHREE:
     if FAIRCOIN = 0
         return 0
     else
         return 1 − ONEINTHREE
   ```

   (a) ***Prove*** that ONEINTHREE returns 1 with probability 1/3.

   (b) What is the *exact* expected number of times that this algorithm calls FAIRCOIN? ***Prove*** your answer is correct.

   (c) Now suppose you are *given* a subroutine ONEINTHREE that generates a random bit that is equal to 1 with probability 1/3. Describe a FAIRCOIN algorithm that returns either 0 or 1 with equal probability, using ONEINTHREE as a subroutine. **Your *only* source of randomness is ONEINTHREE; in particular, you may not use the RANDOM function from problem 3.**

   (d) What is the *exact* expected number of times that your FAIRCOIN algorithm calls ONEINTHREE? ***Prove*** your answer is correct.