1. Proceeding along the lines of the problem done in class -
Probability of success in one attempt is \( \frac{\sqrt{n}}{n} \cdot \frac{\sqrt{n} - 1}{n} \approx \frac{1}{n} \)
Hence, the probability of failure in one attempt is \( \approx \frac{1}{n} \)

Using the inequality \((1 - \frac{1}{n})^x \leq e^{-x/n} \) and \(n - \alpha\), we get: 

\[ k = \frac{\alpha \log n}{\log e} \]

Thus the number of attempts needed is no more than \( \alpha \log n \) with high probability.

In other words, the run time is \( \tilde{O}(n \log n) \). If we only make \( O(\log n) \) attempts, the probability of success can be verified to be not as high as what we want.

2. Algorithm \( A \) is called \( d \alpha \log n \) (\( d \) being a constant) times in the new algorithm. A count of the number of YESs and the number of NOs output by \( A \) is kept. Let \( u \) be the total number of YESs and \( v \) the total number of NOs. If \( u > v \), the new algorithm (call it \( B \)) outputs YES; otherwise it will output NO.

\( B \)'s output will be correct if there are more correct answers than incorrect answers in the \( d \alpha \log n \) answers obtained from \( A \). Let \( X \) be the number of times \( A \) outputs the correct answer and let \( Y \) be the number of times \( A \) outputs the incorrect answer. We want to show that \( \text{Prob.}[X > Y] \geq (1 - n^{-\alpha}) \).

Note that \( X \) is a binomial random variable with parameters \((d \alpha \log e n, c)\). Similarly, \( Y \) is a random variable with parameters \((d \alpha \log e n, (1 - c))\). Since \( c \) is greater than \( \frac{1}{2} \), let \( c = \frac{1}{2} + \delta \) for some constant \( \delta > 0 \). The expected value of \( X \) is \( d \alpha \log e n \left( \frac{1}{2} + \delta \right) \) and the expected value of \( Y \) is \( d \alpha \log e n \left( \frac{1}{2} - \delta \right) \).

Chernoff’s bounds are typically used to show that if the mean of a binomial random variable is \( \mu \), then, with high probability, the actual value of the random variable can not be significantly greater (or less) than \( \mu \). This is what we are going to use to prove our result.

Using Chernoff’s inequality,

\[
\text{Prob.} \left[ X \leq \left( 1 - \frac{\delta}{2} \right) d \alpha \log e n \right] \leq e^{-\frac{\delta^2}{2} d \alpha \log e n}
\]

The RHS of the above inequality will be \( \leq n^{-\alpha} \) if \( d \geq \frac{8}{\delta^2} \).

Thus we have

\[
\text{Prob.} \left[ X \leq \left( \frac{1}{2} + \frac{3}{4} \delta - \frac{\delta^2}{2} \right) d \alpha \log e n \right] \leq n^{-\alpha}
\]

if \( d \geq \frac{8}{\delta^2} \).

From equation 1 we see that \( X \) will be more than \( Y \) with high probability if we pick \( d \geq \frac{8}{\delta^2} \).

3. Here is an algorithm:

\[
\text{for } i = 1 \text{ to } \alpha \log e n \text{ do}
\]

Pick a random \( i \in [1, n] \); Pick a random \( j \in [1, n] \);

\text{if } A[i, j] \neq B[i, j] \text{ then output NO and quit;}

Output YES;

\textbf{Analysis:} If \( A = B \), the above algorithm will never output an incorrect answer.
Consider the case $A \neq B$. In this case the probability that $A$ and $B$ match on a randomly picked row $i$ and column $j$ is $\leq \left(1 - \frac{1}{n}\right)^{an \log_e n}$. Using the fact that $\left(1 - 1/x\right)^x \leq 1/e$ (for any $0 < x < 1$), this probability is $\leq e^{-an \log_e n} \leq n^{-a}$. This is indeed the probability that the algorithm will output an incorrect answer.

4. We assign a polynomial to each node in each of the two trees, by the following rules:

- Every leaf gets polynomial $P = x_0$
- An internal vertex $v$ at height $h$ having children $v_1, v_2, \ldots, v_k$ gets polynomial $P_v = (x_h - P_{v_1})(x_h - P_{v_2}) \ldots (x_h - P_{v_k})$

We claim that the two trees are isomorphic if and only if the polynomials at their roots are equal. The left to right implication is immediate: if the trees are isomorphic then the polynomials will be identical by virtue of multiplication being commutative.

If the polynomials are equal, then we can prove by induction that the trees are isomorphic. The base case is trivial: if two trees have polynomial $x_0$ then they are single node trees and are isomorphic. If the polynomial at the root of both trees is $P_v = (x_h - P_{v_1})(x_h - P_{v_2}) \ldots (x_h - P_{v_k})$, since $x_h$ does not appear in any of $P_{v_i}, i = 1, k$ it must be the case that both trees have $k$ children which can be paired based on their polynomials. By induction, since the polynomials of the children are equal, the children’s subtrees are isomorphic and thus the two trees are isomorphic.

So the problem of checking tree isomorphism has been reduced to checking equality of (at most) degree $n$ multivariate polynomials. This can be done in $\tilde{O}(n)$ as presented in class.

5. A trivial algorithm is algorithm 1. The worst case runtime is $O(mT_c)$ where $T_c$ is the time needed to check if a graph has a perfect matching, which is the same as the time needed to multiply two matrices.

**Algorithm 1**

1. pick edge $e = (u, v) \in E$
   remove edge $e$ and nodes $u$ and $v$ from the graph
   if there exists a perfect matching for the new graph then
     recursively compute perfect matching $M$ on the remaining graph
     return $M \cup (u, v)$
   else
     restore graph to initial state
     remove edge $(u, v)$ but not nodes $u$ and $v$
     go to 1
   end if

6. For each submatrix of size $m^2$, the probability of giving an incorrect answer is $\leq \frac{m^2}{t \log t}$ where $[1, t]$ is the range out of which we choose prime $p$. The probability of giving an incorrect answer for any of the $(n - m + 1)^2$ submatrices is $\leq \frac{(n-m+1)^2m^2}{t \log t}$. The max of $(n - m + 1)^2m^2$ is obtained for $m = n/2$ and is $O(n^4)$. So, if we choose $t = n^{a+4.1}$ then the previous probability is less than $n^{-a}$.

The runtime is $O(n^2)$ using the following observation. Let $B[i, j]$ be the fingerprint of the submatrix having the lower right corner at $(i, j)$ and let $C[i, j]$ be the fingerprint of $m$ contiguous values in row $i$, ending at position $j$. Then
\[
B[i, j] = (B[i - 1, j] - 2^{m^2 - m}C[i - m + 1, j])2^m + C[i, j](mod p)
\]

For row \(i\), we only need values of \(B\) from row \(i - 1\), so the extra memory for \(B\) is linear. The values of \(C\) for row \(i\) and \(i - m + 1\) at column \(j\) can be computed on the fly as we scan the current row from left to right, so for \(C\) we only add constant memory overhead.

Thus, we can compute the fingerprint of the submatrix ending at \((i, j)\) in constant time from the fingerprint of the submatrix ending at position \((i - 1, j)\). Since testing each fingerprint takes \(O(1)\) time, the total runtime is \(O(n^2)\) (including the cost of fingerprinting the initial submatrices ending at positions in row \(m - 1\)).