1. Implement the Miller-Rabin primality test. Use your primality test function to implement a \texttt{next prime}(v) function that finds a minimal prime \( p \) with \( p \geq v \).

2. Implement the algorithm \texttt{strong prime}(n) for generating a strong prime (see Figure 1). Think of a strategy in selecting the proper sizes of primes \( s, t \) and parameters \( i_0, j_0 \).

3. Implement RSA key generation for a given parameter \( n \), with \( n \) being the bit length of the key. Allow for the use of the \texttt{strong prime} (default) and \texttt{next prime} functions. Implement RSA encryption and decryption for a set of public and private keys. Test your implementation for different key length \( |k| \in \{256, 512, 1024, 2048\} \). Write a test program, that first generates a random pair of public and private key. Then read an input value (message) \( m \) and encrypt it using the public key. Afterwards, decrypt the encrypted message using the private key. The decrypted value should match the input value.

4. Implement a factorization attack on RSA for key length \( |k| \in \{60, 80, 100, 120\} \) using your factorization strategy from Project 2. 

   \textbf{Extra credit:} Adjust your strategy to attack RSA with key length \( |k| \in \{140, 160\} \).

5. Compare results of factorization when random primes (function \texttt{next prime}(v)) used instead of strong primes. Describe your experiments and give conclusions.

Please use the method described in Project 1 to measure performance and the factorization routines of Project 2.

---

\textbf{INPUT:} \hspace{5pt} n - bit length of the prime  
\textbf{OUTPUT:} A strong prime of bit length \( n \)  
1: Generate two primes \( s \) and \( t \) of size roughly equal to \( n/2 \)  
2: Select integer \( i_0 \). Find the first prime \( r = 2it + 1, i = i_0, i_0 + 1, i_0 + 2, \ldots \)  
3: Compute \( p_0 = (2s^{-2} \mod r)s - 1 \)  
4: Select integer \( j_0 \). Find the first prime \( p = p_0 + 2jrs, j = i_0, j_0 + 1, j_0 + 2, \ldots \)  
5: \hspace{10pt} RETURN \( p \)  

---

Figure 1: Gordon’s algorithm for generating a strong prime.

This project is due by December 15.