Let $a, m$ be integers, $m \neq 0$. Recall that $(a \mod m)$ denotes the smallest non-negative remainder of the division of $a$ by $m$. In other words, let $a = mq + r$ where $0 \leq r \leq |m| - 1$. This $r$ is unique and is denoted $r = (a \mod m)$.

**Problem (modular exponentiation):** Calculate $(a^b \mod m)$ where $a, b, m$ are integers, $a, m \geq 1, b \geq 0$.

**Solution:** the method of repeated squaring.

**Pseudocode A.**

0 Initialize: $X := 1$, $B := b$, $A := (a \mod m)$  
\[ [X \text{ is the “accumulator” that collects the partial results}] \]
1 while $B \geq 1$ do  
2 \hspace{1em} if $B$ odd then $B := B - 1$, $X := (AX \mod m)$  
3 \hspace{1em} else $B := B/2$, $A := (A^2 \mod m)$  
4 end(while)  
5 return $X$

The **correctness** of the algorithm follows from the following **loop invariant** (verify!)

$$XA^B \equiv a^b \mod m.$$  

The **efficiency** of the algorithm follows from the observation that after every two rounds, the value of $B$ is reduced to less than half. (Prove!) This implies that the number of rounds is $\leq 2n$ where $n$ is the number of bits (binary digits) of $b$. Moreover, we never deal with integers greater than $m^2$. Therefore, if $a, b, m$ each have $n$ bits (initial zeros permitted) then every number involved has $\leq 2n$ bits and the total number of bit-operations is $O(n^3)$ (using the schoolbook multiplication/division method) so this is a **polynomial-time algorithm.** (Recall that the comparison is made with the bit-length of the input, which in this case is $3n$.)

We now describe an alternative, recursive implementation. The non-recursive code is preferred.

**Pseudocode B: recursive.**

0 procedure $f(a, b, m) = (a^b \mod m)$  
\[ (b \geq 0, a, m \geq 1) \]
1 \hspace{1em} if $b = 0$ then return 1  
2 \hspace{1em} elseif $b$ odd then return $(a \cdot f(a, b - 1, m) \mod m)$  
3 \hspace{1em} elseif $b$ even then return $f((a^2 \mod m), b/2, m)$