```
recFib:
        addi    $sp, $sp, -12       # save $ra, $s0, and $s1
        sw      $ra, 0($sp)
        sw      $s0, 4($sp)
        sw      $s1, 8($sp)
        li      $t0, 1
        bgt     $a0, $t0, Recurse   # if (n > 1) recurse
        move    $v0, $a0            # return n
        j       Done
Recurse:
        addi    $s0, $a0, -1        # $s0 <-- n-1
        addi    $a0, $a0, -2        # $10 <-- n-2
        jal     recFib              #
        move    $s1, $v0            # $s1 <-- Fib(n-2)
        move    $a0, $s0            # $a0 <-- n-1
        jal     recFib              #
        add     $v0, $v0, $s1       # $v0 <-- Fib(n-1) + Fib(n-2)
Done:
        lw      $ra, 0($sp)         # restore $ra, $s0, and $s1
        lw      $s0, 4($sp)
        lw      $s1, 8($sp)
        addi    $sp, $sp, 12
        jr      $ra
```

```
iterFib:
        li      $t0, 1              # i <-- 1
        bgt     $a0, $t0, Compute   # if(n > 1) goto Compute
        move    $v0, $a0
        jr      $ra
Compute:
        move    $v0, $zero          # j <-- 0
        li      $t2, 1              # k <-- 1
Loop:
        add     $t3, $t0, $v0       # t <-- i + j
        move    $t0, $v0            # i <-- j
        move    $v0, $t3            # j <-- t
        addi    $t2, $t2, 1         # k <-- k + 1
        ble     $t2, $a0, Loop      # if(k <= n) goto Loop
        jr      $ra
```

Figure 1: (a) Recursive Fibonacci; (b) Iterative Fibonacci

Question 4 (20 points): The Fibonacci sequence is defined as follows:

$$\text{Fib}(n) \quad = \quad \begin{cases} \text{Fib}(n-1) + \text{Fib}(n-2) & \text{if} n > 1, \\ n & \text{if} n \leq 1. \end{cases}$$

Figure 1 shows two versions of MIPS assembly program that compute the Fibonacci value of a number n, which is the input parameter in `$a0`.

a. (**10 points**) Which version, `recFib` or `iterFib`, is more efficient for a large value of n? Explain your answer.

b. (**10 points**) Give an expression, in terms of n, for the amount of storage, given in bytes, that must be available to grow the stack for each of the subroutines to execute correctly.