Procedure body

```
swap: sll     $t1, $a1, 2          # reg $t1 = k * 4
      add     $t1, $a0, $t1        # reg $t1 = v + (k * 4)
                                   # reg $t1 has the address of v[k]
      lw      $t0, 0($t1)          # reg $t0 (temp) = v[k]
      lw      $t2, 4($t1)          # reg $t2 = v[k + 1]
                                   # refers to next element of v
      sw      $t2, 0($t1)          # v[k] = reg $t2
      sw      $t0, 4($t1)          # v[k+1] = reg $t0 (temp)
```

Procedure return

```
      jr      $ra                  # return to calling routine
```

FIGURE 2.34 MIPS assembly code of the procedure `swap` **in Figure 2.33.**

```
void sort (int v[], int n)
{
   int i, j;
   for (i = 0; i < n; i += 1) {
      for (j = i – 1; j >= 0 && v[j] > v[j + 1]; j -= 1) {
            swap(v,j);
      }
   }
}
```

FIGURE 2.35 A C procedure that performs a sort on the array `v`**.**

Saving registers

	sort:	addi	$sp,$sp, -20	# make room on stack for 5 registers
		sw	$ra, 16($sp)	# save $ra on stack
		sw	$s3,12($sp)	# save $s3 on stack
		sw	$s2, 8($sp)	# save $s2 on stack
		sw	$s1, 4($sp)	# save $s1 on stack
		sw	$s0, 0($sp)	# save $s0 on stack

Procedure body

Move parameters		move	$s2, $a0	# copy parameter $a0 into $s2 (save $a0)
		move	$s3, $a1	# copy parameter $a1 into $s3 (save $a1)
Outer loop		move	$s0, $zero	# i = 0
	for1tst:	slt	$t0, $s0, $s3	# reg $t0 = 0 if $s0 ≥ $s3 (i ≥ n)
		beq	$t0, $zero, exit1	# go to exit1 if $s0 ≥ $s3 (i ≥ n)
Inner loop		addi	$s1, $s0, -1	# j = i - 1
	for2tst:	slti	$t0, $s1, 0	# reg $t0 = 1 if $s1 < 0 (j < 0)
		bne	$t0, $zero, exit2	# go to exit2 if $s1 < 0 (j < 0)
		sll	$t1, $s1, 2	# reg $t1 = j * 4
		add	$t2, $s2, $t1	# reg $t2 = v + (j * 4)
		lw	$t3, 0($t2)	# reg $t3 = v[j]
		lw	$t4, 4($t2)	# reg $t4 = v[j + 1]
		slt	$t0, $t4, $t3	# reg $t0 = 0 if $t4 ≥ $t3
		beq	$t0, $zero, exit2	# go to exit2 if $t4 ≥ $t3
Pass parameters and call		move	$a0, $s2	# 1st parameter of swap is v (old $a0)
		move	$a1, $s1	# 2nd parameter of swap is j
		jal	swap	# swap code shown in Figure 2.34
Inner loop		addi	$s1, $s1, -1	# j -= 1
		j	for2tst	# jump to test of inner loop
Outer loop	exit2:	addi	$s0, $s0, 1	# i += 1
		j	for1tst	# jump to test of outer loop

Restoring registers

	exit1:	lw	$s0, 0($sp)	# restore $s0 from stack
		lw	$s1, 4($sp)	# restore $s1 from stack
		lw	$s2, 8($sp)	# restore $s2 from stack
		lw	$s3,12($sp)	# restore $s3 from stack
		lw	$ra,16($sp)	# restore $ra from stack
		addi	$sp,$sp, 20	# restore stack pointer

Procedure return

		jr	$ra	# return to calling routine

FIGURE 2.36 MIPS assembly version of procedure `sort` **in Figure 2.35 on page 124.**

Elaboration: One optimization that works with this example is *procedure inlining*, mentioned in Section 2.11. Instead of passing arguments in parameters and invoking the code with a `jal` instruction, the compiler would copy the code from the body of the `swap` procedure where the call to `swap` appears in the code. Inlining would avoid four