1. Lines (1) through (3) and (7) each take 1 time unit. The loop on lines (4) through (6) executes $n - i + 1$ times. But i is given and is 2, thus the loop executes $n - 1$ times. Each instruction, including the condition, takes 1 time unit. But the condition also is executed by one more than how many times the body is executed. Hence the loop takes $3(n - 1) + 1$ time units. Together with the 4 time units spent in the instructions other than the loop, the total time taken by the program is $3n + 2$. Since n is the size of the data the program operates on, then the running time $T(n) = 3n + 2$.

2. Consider Exercise 2.5.1. Line (4) is executed n times taking 1 time unit each. The incrementation on line (3) takes 1 time unit as well and executes n times. The condition on line (3) is executed $n + 1$ times and takes 1 time unit each. Lines (1) and (2) each take 1 time unit. The initialization on line (3) takes 1 time unit. The size of the data is n. Thus we have the running time $T(n) = 3n + 4$.

Consider Fig. 2.14. Lines (1) and (2) each take 1 time unit. Lines (4) and (5) each take 1 time unit and execute n times. Line (3) takes 1 time unit and executes $n + 1$ times. The size of the data is n, the number of elements to be used as input. The running time of the program $T(n) = 3(n + 1)$.

3. We use a program to determine this. For $1 \leq n \leq 29$, program A takes less time than program B. The value of the procedure call given below is 30 which represents the first number n such that program A takes more time than program B.

```
(define (cmp A B)
  (define (cmp-iter n)
    (if (> (A n) (B n))
        n
        (cmp-iter (+ n 1))))
  (cmp-iter 1))

(cmp (lambda (n) (/ (expt 2 n) 1000))
     (lambda (n) (* 1000 (square n))))
```

The procedure may never terminate, given certain inputs.

4. We use a program to determine this. For program A, the maximum problem sizes are (a) 29, (b) 39, (c) 49. For program B, the maximum problem sizes are (a) 31, (b) 1000, (c) 31622.

```
(define (max-size seconds)
  (lambda (exp)
    (define (ms n seconds)
      (cond ((null? seconds) '())
            ((> (exp n) (car seconds))
             (cons (- n 1) (ms n (cdr seconds))))
            (else (ms (+ n 1) seconds))))
    (ms 1 seconds)))

(define big-max-size
  (max-size (list (expt 10 6) (expt 10 9) (expt 10 12))))
```

```
(big-max-size (lambda (n) (/ (expt 2 n) 1000)))
(big-max-size (lambda (n) (* 1000 (square n))))
```

The definition `(define small-max-size (max-size (list 1 10 100 1000)))` is how we can determine those maximum problem sizes found in Fig. 3.3.