1. We determine the backward induction form of the definition by substituting n for $n-1$ and if we let F represent the square function. We have

$$\begin{aligned} F(n) &= F(n-1) + 2(n-1) + 1 \\ &= F(n-1) + 2n - 2 + 1 \\ &= F(n-1) + 2n - 1. \end{aligned}$$

a)

```
int square(int n)
{
    if (n <= 1) /* defense */
       return 1;
    else
       return square(n-1) + 2*n - 1;
}
```

b)

STATEMENT $S(n)$: The recursive definition of n^2 given in exercise 2.7.1 correctly computes n^2.

BASIS. The basis is true immediately from the definitions.

INDUCTION. Assume the recursive definition correctly computes squares of $j \leq n$. We shall prove $S(n+1)$. Let F be the function that computes squares as given by the recursive definition. By the inductive hypothesis, we know $F(n) = n^2$. Therefore

$$\begin{aligned} F(n+1) &= F(n) + 2(n+1) - 1 \\ &= n^2 + 2n + 2 - 1 \\ &= (n+1)^2. \end{aligned}$$

Hence $F(n+1)$ correctly computes $(n+1)^2$, which proves $S(n+1)$. We conclude that the recursive definition correctly computes n^2 for all $n \geq 1$. ✦

2. For now we use a whitespace-separated list enclosed in braces to denote the elements of an array.

```
recSS({10 13 4 7 11}, 0, 5)
recSS({4 13 10 7 11}, 1, 5)
recSS({4 7 10 13 11}, 2, 5)
recSS({4 7 10 13 11}, 3, 5)
recSS({4 7 10 11 13}, 4, 5)
```

3.

```
BOOLEAN find(LIST L, int n)
{
    if (L == NULL) return FALSE;
    else if (L->element == n) return TRUE;
    else return find(L->next, n);
```

```
}

BOOLEAN find1698(LIST L)
{
    return find(L, 1698);
}
```

4.

```
int add(LIST L)
{
    if (L == NULL) return 0;
    else return L->element + add(L->next);
}
```

5.

```
void selectionsort(LIST L)
{
    LIST small;
    int temp;

    if (L == NULL) return;
    else {
        small = smallest(L, L->next);
        temp = L->element;
        L->element = small->element;
        small->element = temp;
        selectionsort(L->next);
    }
}

LIST smallest(LIST small, LIST current)
{
    if (current == NULL) return small;
    else if (current->element < small->element)
         return smallest(current, current->next);
    else return smallest(small, current->next);
}
```

6.

```
void recSS(T A[], int i, int n)
{
    int j, small;
    T temp;
    if (i < n-1) {
        small = i;
```

```
        for (j = i+1; j < n; j++)
            if (lt(key(A[j]), key(A[small])))
                small = j;
        temp = A[small];
        A[small] = A[i];
        A[i] = temp;
        recSS(A, i+1, n);
    }
}
```

7.

```
void binary(int i)
{
    if (i == 0) printf("0");
    else b(i);
}

void b(int i)
{
    if (i <= 0) return;
    else {
        printf("%d", i%2);
        b(i/2);
    }
}
```

8.

```
int gcd(int i, int j)
{
    if (i%j == 0) return j;
    else return gcd(j, i%j);
}
```

9. We must prove only that the recursive definition of GCD gives the same result as the nonrecursive definition, and not the converse.

STATEMENT $S(n)$: If g is determined to be the GCD of i and j by n applications of the recursive rule, then g is the largest integer dividing both i and j evenly.

BASIS. If $n = 0$, then $g = j = gcd(i, j)$. If there is a larger integer $\ell > g$ that divides i and j, then $\ell > j$, which would then not divide j. Therefore g is the largest integer dividing both i and j evenly.

INDUCTION. Assume $n \geq 0$ and that $S(n)$ holds. We shall prove $S(n+1)$. Suppose g is determined to be the GCD of i and j by $n + 1$ applications of the recursive rule. Since $g = gcd(i, j) = gcd(j, k)$, then the inductive hypothesis holds for g from one application of the recursive rule to the next. On the $n + 1$th application we compute $gcd(j, i \mod j)$. Then by the basis of the recursive definition we have $g = gcd(i, j)$. Hence g is the GCD by the nonrecursive definition. Therefore $S(n)$ is true for $n \geq 0$. ✦

10.

```
BOOLEAN lesser(LIST W, LIST X)
{
    if (X == NULL) return FALSE;
    else if (W == NULL) return TRUE;
    else if (W->element == X->element)
        return lesser(W->next, X->next);
    else return (W->element < X->element);
}
```

11.