

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.