

1. Using Figure 1.2 as a model, illustrate the operation of INSERTION-SORT on the array $A = \langle 31, 41, 59, 26, 41, 58 \rangle$.

The following Java code implements INSERTION-SORT:

```

1 import java.util.List;
2
3 public class Sort {
4     public <T extends Comparable<T>> void sort(List<T> m) {
5         //! Guard against IndexOutOfBoundsException
6         if ( m.size() <= 1 )
7             return;
8
9         //! Implement INSERTION-SORT
10        for ( int i=1; i < m.size(); ++i ) {
11            T elem = m.get(i);
12            int j = i-1;
13            while ( j >= 0 && m.get(j).compareTo(elem) > 0 ) {
14                m.set(j+1, m.get(j)); j--;
15            }
16            m.set(j+1, elem);
17        }
18    }
19 }

```

This is an in-place implementation of INSERTION-SORT, with the output overwriting the input and at most a constant amount of secondary memory being allocated from the heap. The following illustrates the state of the input array as the algorithm runs:

$i = 1, j = 0, elem = 41, A = \langle 31, 41, 59, 26, 41, 58 \rangle$
 $i = 1, j = 0, elem = 41, A = \langle 31, \text{41}, 59, 26, 41, 58 \rangle \rightarrow A(1) = 41$
 $i = 2, j = 1, elem = 59, A = \langle 31, 41, 59, 26, 41, 58 \rangle$
 $i = 2, j = 1, elem = 59, A = \langle 31, 41, \text{59}, 26, 41, 58 \rangle \rightarrow A(2) = 59$
 $i = 3, j = 2, elem = 26, A = \langle 31, 41, 59, 26, 41, 58 \rangle$
 $i = 3, j = 2, elem = 26, A = \langle 31, 41, 59, \text{59}, 41, 58 \rangle \rightarrow A(3) = 59$
 $i = 3, j = 1, elem = 26, A = \langle 31, 41, \text{41}, \text{59}, 41, 58 \rangle \rightarrow A(2) = 41$
 $i = 3, j = 0, elem = 26, A = \langle 31, \text{31}, \text{41}, \text{59}, 41, 58 \rangle \rightarrow A(1) = 31$
 $i = 3, j = -1, elem = 26, A = \langle \text{26}, \text{31}, \text{41}, \text{59}, 41, 58 \rangle \rightarrow A(0) = 26$
 $i = 4, j = 3, elem = 41, A = \langle 26, 31, 41, 59, 41, 58 \rangle$
 $i = 4, j = 3, elem = 41, A = \langle 26, 31, 41, 59, \text{59}, 58 \rangle \rightarrow A(4) = 59$
 $i = 4, j = 2, elem = 41, A = \langle 26, 31, 41, \text{41}, \text{59}, 58 \rangle \rightarrow A(3) = 41$
 $i = 5, j = 4, elem = 58, A = \langle 26, 31, 41, 41, 59, 58 \rangle$
 $i = 5, j = 4, elem = 58, A = \langle 26, 31, 41, 41, 59, \text{59} \rangle \rightarrow A(5) = 59$
 $i = 5, j = 4, elem = 58, A = \langle 26, 31, 41, 41, \text{58}, \text{59} \rangle \rightarrow A(4) = 58$

The final result is therefore:

$$A = \langle 26, 31, 41, 41, 58, 59 \rangle$$

2. Rewrite the INSERTION-SORT procedure to sort into non-increasing instead of non-decreasing order.

The following Java code provides an implementation of INSERTION-SORT that sorts arrays into non-increasing order:

```
1 import java.util.List;
2
3 public class Sort {
4     public <T extends Comparable<T>> void sort(List<T> m) {
5         //! Guard against IndexOutOfBoundsException
6         if ( m.size() <= 1 )
7             return m;
8
9         //! Implement INSERTION-SORT
10        for ( int i=1; i < m.size(); ++i ) {
11            T elem = m.get(i);
12            int j = i-1;
13            while ( j >= 0 && m.get(j).compareTo(elem) < 0 ) {
14                m.set(j+1, m.get(j)); j--;
15            }
16            m.set(j+1, elem);
17        }
18    }
19 }
```

The only change is at line 11 where

```
m.get(j).compareTo(elem) > 0
```

has been changed to:

```
m.get(j).compareTo(elem) < 0
```

3. Consider the *searching problem*:

Input: A sequence of n numbers $A = \langle a_1, a_2, \dots, a_n \rangle$ and a value v .

Output: An index i such that $v = A[i]$ or the special value NIL if v does not appear in A .

Write pseudocode for *linear search*, which scans through the sequence looking for v .

The following Python code implements the algorithm:

```
1 def search(A, v):
2     j = None
3     for i in range(len(A)):
4         if A[i] == v:
5             j = i
6             break;
7     return j
```

4. Consider the problem of adding two n -bit binary numbers, stored in two n -element arrays A and B . The sum of the two integers should be stored in an $(n+1)$ -element array C . State the problem formally and write pseudocode for adding the two integers.

We can reason about this problem inductively (i.e., recursively).

Suppose we start by adding two 1-bit binary numbers.

There are four cases to consider:

$$\begin{aligned}[0] + [0] &= [00] \\ [1] + [0] &= [01] \\ [0] + [1] &= [01] \\ [1] + [1] &= [10]\end{aligned}$$

From this pattern we can surmise that:

$$\begin{aligned}C[i] &= (A[i] + B[i]) \% 2 \\ C[i+1] &= (A[i] + B[i]) / 2\end{aligned}$$

We can implement the desired algorithm in Python as follows:

```
1 def add(A, B, n):
2     A_ = A[:]; A_.reverse()
3     B_ = B[:]; B_.reverse()
4     C = [0] * (n+1)
5     for i in range(n):
6         sum_ = A_[i] + B_[i] + C[i]
7         C[i] = int(sum_%2)
8         C[i+1] = int(sum_/2)
9     C.reverse()
10    return C
```