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

The following Java code implements INSERTION-SORT:

```java
public class Sort {
  public <T extends Comparable<T>> void sort(List<T> m) {
    //! Guard against IndexOutOfBoundsException
    if ( m.size() <= 1 )
      return;

    //! Implement INSERTION-SORT
    for ( int i=1; i < m.size(); ++i ) {
      T elem = m.get(i);
      int j = i-1;
      while ( j >= 0 && m.get(j).compareTo(elem) > 0 ) {
        m.set(j+1, m.get(j)); j--;
      }
      m.set(j+1, elem);
    }
  }
}
```

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:

$i = 1$, $j = 0$, $elem = 41$, $A =< 31, 41, 59, 26, 41, 58 >$
$i = 1$, $j = 0$, $elem = 41$, $A =< 31, 41, 59, 26, 41, 58 > \rightarrow A(1) = 41$
$i = 2$, $j = 1$, $elem = 59$, $A =< 31, 41, 59, 26, 41, 58 >$
$i = 2$, $j = 1$, $elem = 59$, $A =< 31, 41, 59, 26, 41, 58 > \rightarrow A(2) = 59$
$i = 3$, $j = 2$, $elem = 26$, $A =< 31, 41, 59, 26, 41, 58 >$
$i = 3$, $j = 2$, $elem = 26$, $A =< 31, 41, 59, 59, 41, 58 > \rightarrow A(3) = 59$
$i = 3$, $j = 1$, $elem = 26$, $A =< 31, 41, 41, 59, 41, 58 > \rightarrow A(2) = 41$
$i = 3$, $j = 0$, $elem = 26$, $A =< 31, 31, 41, 59, 41, 58 > \rightarrow A(1) = 31$
$i = 3$, $j = -1$, $elem = 26$, $A =< 26, 31, 41, 59, 41, 58 > \rightarrow A(0) = 26$
$i = 4$, $j = 3$, $elem = 41$, $A =< 26, 31, 41, 59, 41, 58 >$
$i = 4$, $j = 3$, $elem = 41$, $A =< 26, 31, 41, 59, 59, 58 > \rightarrow A(4) = 59$
$i = 4$, $j = 2$, $elem = 41$, $A =< 26, 31, 41, 41, 59, 58 > \rightarrow A(3) = 41$
$i = 5$, $j = 4$, $elem = 58$, $A =< 26, 31, 41, 41, 59, 58 >$
$i = 5$, $j = 4$, $elem = 58$, $A =< 26, 31, 41, 41, 59, 59 > \rightarrow A(5) = 59$
$i = 5$, $j = 4$, $elem = 58$, $A =< 26, 31, 41, 41, 58, 59 > \rightarrow A(4) = 58$

The final result is therefore:

$$A =< 26, 31, 41, 41, 58, 59 >$$

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:

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

The only change is at line 11 where

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

has been changed to:

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

3. Consider the ***searching problem***:

Input: A sequence of n numbers $A =< a_1, a_2, ..., a_n >$ 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:

```python
def search(A, v):
  j = None
  for i in range(len(A)):
    if A[i] == v:
      j = i
      break;
  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:

[0] + [0] = [00]
[1] + [0] = [01]
[0] + [1] = [01]
[1] + [1] = [10]

From this pattern we can surmise that:

$$C[i] = (A[i] + B[i])\%2$$
$$C[i+1] = (A[i] + B[i])/2$$

We can implement the desired algorithm in Python as follows:

```python
def add(A, B, n):
  A_ = A[:]; A_.reverse()
  B_ = B[:]; B_.reverse()
  C = [0] * (n+1)
  for i in range(n):
    sum_ = A_[i] + B_[i] + C[i]
    C[i] = int(sum_%2)
    C[i+1] = int(sum_/2)
  C.reverse()
  return C
```