# Chapter 2 - Getting Started
# Exercises & Problems

## Exercises

### 2.1 Insertion Sort

#### Exercise 2.1-1

Sorting the array A = {31, 41, 59, 26, 41, 58} using insertion sort would look something like the following. Highlighted values are the values currently being compared and each value is placed in its final spot at the end of that step.

| | | | | | |
|---|---|---|---|---|---|
| 31 | 41 | 59 | 26 | 41 | 58 |
| 31 | 41 | 59 | 26 | 41 | 58 |
| 26 | 31 | 41 | 59 | 41 | 58 |
| 26 | 31 | 41 | 41 | 59 | 58 |
| 26 | 31 | 41 | 41 | 58 | 59 |

#### Exercise 2.1-2

INSERTION-SORT($A$)

```
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1..j-1].
4      i = j - 1
5      // Switch the comparison operator to sort from greatest to least.
6      while i > 0 and A[i] < key
7          A[i+1] = A[i]
8          i = i - 1
9      A[i+1] = key
```

#### Exercise 2.1-3

The **loop invariant** for linear search is that at the start of each iteration of the **for** loop on lines 1-3, the element in question has not yet been found, since it will have returned if it were found on a previous iteration. We will demonstrate this over the following three scenarios.
**Initialization:** When we initialize $i = 1$, before checking that index there is no way of knowing, and after checking whether $A[i]$ is equal to $v$, the function will return $i = 1$ if true, or we will continue otherwise.
**Maintenance:** At the start of each iteration of the **for** loop, it must be true that the element being

searched for has not yet been found, or the loop would have returned, thus the invariant holds true.
**Termination:** Upon termination of the for loop, one of two things must be true: (1) we have found the element in $A$ at index $i$ and $i$ has been returned, or (2) the element was not found in $A$ and we return NIL.

Finally, the below psuedocode gives an illustration of the linear search algorithm.

```
LINEAR-SEARCH(A, v)
1  for i = 1 to A.length
2        if A[i] == v
3              return i
4  // v was not found
5  return NIL
```

**Exercise 2.1-4**

We formally define the problem of adding two $n$-bit binary integers, stored in two $n$-element arrays $A$ and $B$ as follows:
**Input:** Two $n$-element arrays $A$ and $B$ representing two $n$-bit binary integers
**Output:** An $(n+1)$-element array $C$ representing the binary integer sum of $A$ and $B$

```
BINARY-ADDITION(A, B)
1  carry = 0
2  for i = A.length − 1 downto 0
3        if A[i] == element
4              return i
5  // Element was not found
6  return NIL
```

## 2.2 Analyzing Algorithms

**Exercise 2.2-1**

$n^3/1000 - 100n^2 - 100n + 3$ can be expressed in terms of $\Theta$-notation as $\Theta(n^3)$, since eventually the $n^3$ term will dominate the others.

**Exercise 2.2-2**

Selection sort maintains the invariant the for each iteration of the **for** loop, the elements $A[1\,..\,i-1]$ are in sorted order. The algorithm sorts through the first $n-1$ elements of the array, since at that point the last element remaining is as a result already sorted.
It is however a very inefficient algorithm, needing $\Theta(n^2)$ in the best, worst, and average cases. Even if the array is already sorted, the algorithm must still step over each element to learn that fact on each iteration.

Below is the psuedocode for selection sort.

```
SELECTION-SORT(A)
1  for i = 1 to A.length − 1
2      min = A[i]
3      mindex = i
4      for j = i to A.length
5          if A[j] < min
6              min = A[j]
7              mindex = j
8      swap(A[i], A[mindex])
```

**Exercise 2.2-3**

**Exercise 2.2-4**

### 2.3 Designing Algorithms

**Exercise 2.3-1**

**Exercise 2.3-2**

**Exercise 2.3-3**

**Exercise 2.3-4**

**Exercise 2.3-5**

**Exercise 2.3-6**

**Exercise 2.3-7**

## Problems

### Problem 2-1: Insertion Sort on Small Arrays in Merge Sort

### Problem 2-2: Correctness of Bubble Sort

### Problem 2-3: Correctness of Horner's Rule

### Problem 2-4: Inversions