## Section 2.1 – Insertion sort

2.1-1 Using Figure 2.2 as a model, illustrate the operation of INSERTION-SORT on the array $A = \langle 31, 41, 59, 26, 41, 58 \rangle$.

| | | | | | | |
|---|---|---|---|---|---|---|
| (a) | 31 | 41 | 59 | 26 | 41 | 58 |
| (b) | 31 | 41 | 59 | 26 | 41 | 58 |
| (c) | 31 | 41 | 59 | 26 | 41 | 58 |
| (d) | 26 | 31 | 41 | 59 | 41 | 58 |
| (e) | 26 | 31 | 41 | 41 | 59 | 58 |
| (f) | 26 | 31 | 41 | 41 | 58 | 59 |

2.1-2 Rewrite the INSERTION-SORT procedure to sort into non-increasing instead of non-decreasing order.

The pseudocode is stated below.

```
Input: Array A
for j = 2 to A.length do
    key = A[j]
    i = j − 1
    while i > 0 and A[i] > key do
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

2.1-3 Consider the ***searching problem***:

**Input:** A sequence of $n$ numbers $A = \langle a_1, a_2, \ldots, a_n \rangle$ and a value $\nu$.
**Output:** An index $i$ such that $\nu = A[i]$ or the special value NIL if $\nu$ does not appear in $A$.

Write pseudocode for linear search, which scans through the sequence, looking for $\nu$. Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.

The pseudocode is stated below.

```
Input  : Array A
         Desired value ν
Output: Index i or NIL
1 for i = 1 to A.length do
2     if A[i] == ν then
3         return i
4 return NIL
```

Here is the *loop invariant*. At the start of each iteration of the **for** loop of lines 1–3, the algorithm assures that the subarray $A[1, \ldots, i - i]$ does not contain the element $\nu$. Within each iteration, if $A[i]$ corresponds to the $\nu$ element, its index is returned.

**Initialization.** Before the **for** loop, $i = 1$ and $A[1, \ldots, i - 1]$ constains no element (therefore does not contain $\nu$).

**Maintenance.** The body of the **for** loop verifies if $A[i]$ corresponds to the $\nu$ element. If the element correspond to $\nu$, its index is returned. Otherwise, incrementing $i$ for the next iteration of the **for** loop then preserves the loop invariant.

**Termination.** The **for** loop can terminate in one of the following conditions: (1) $A[i] = \nu$, which means that $\nu$ was found and its index is returned; (2) $i > A.length$ and, since each loop iteration increases $i$ by 1, at that time we have $i = A.length + 1$ which assures (from the previous property) that $A[1, \ldots, A.length]$ does not contain the element $\nu$.

2.1-4 Consider the problem of adding two $n$-bit binary integers, stored in two $n$-element arrays $A$ and $B$. The sum of the two integers should be stored in binary form in an $(n + 1)$-element array $C$. State the problem formally and write pseudocode for adding the two integers.

The pseudocode is stated below.

**Input :** Two integers, stored in two n-bit arrays (little endian) $A = \langle a_1, a_2, \ldots, a_n \rangle$ and $B = \langle b_1, b_2, \ldots, b_n \rangle$.
**Output:** A $(n+1)$-bit array $C = \langle c_1, c_2, \ldots, c_{n+1} \rangle$ storing the sum of the two aforementioned integers.

```
let C[1, ..., n+1] be a new array
C[1] = 0
for i = 1 to A.length do
    s = A[i] + B[i] + C[i]
    C[i] = s mod 2
    C[i+1] = s / 2
return C
```