2.3 Implementing algorithms

In the real world you need to be able to use theoretical knowledge in practise.

For example: apply a given sorting algorithm ins a certain programming problem

- numbers are rarely sorted alone, we sort structures with
  - a *key*
  - *satellite data*
- the key sets the order
  $\Rightarrow$ it is used in the comparisons
- the satellite data is not used in the comparison, but it must be moved around together with the key

The INSERTION-SORT algorithm from the previous chapter would change as follows if there were some satellite data used:

```
1   for j := 2 to A.length do
2       temp := A[j]
3       i := j − 1
4       while i > 0 and A[i].key > temp.key do
5           A[i + 1] := A[i]
6           i := i − 1
7       A[i + 1] := temp
```

- An array of pointers to structures should be used with a lot of satellite data. The sorting is done with the pointers and the structures can then be moved directly to their correct locations.

The programming language and the problem to be solved also often dictate other implementation details, for example:

- Indexing starts from 0 (in pseudocode often from 1)
- Is indexing even used, or some other method of accessing data (or do we use arrays or some other data structures)
- (C++) Is the data really inside the array/datastructure, or somewhere else at the end of a pointer (in which case the data doesn't have to be moved and sharing it is easier). Many other programming languages always use pointers/references, so you don't have to choose.
- If you refer to the data indirectly from elsewhere, does it happen with
  - Pointers (or references)
  - Smart pointers (C++, `shared_ptr`)
  - Iterators (if the data is inside a datastructure)
  - Index (if the data is inside an array)
  - Search key (if the data is insde a data structure with fast search)

- Is recursion implemented really as recursion or as iteration
- Are algorithm "parameters" in pseudocode really parameters in code, or just variables

In order to make an executable program, additional information is needed to implement INSERTION-SORT

- an actual programming language must be used with its syntax for defining variables and functions
- a main program that takes care of reading the input, checking its legality and printing the results is also needed
  - it is common that the main is longer than the actual algorithm

The implementation of the program described abowe in C++:

```
#include <iostream>
#include <vector>
typedef std::vector<int> Array;

void insertionSort( Array & A ) {
  int key, i; unsigned int j;
  for( j = 1; j < A.size(); ++j ) {
    key = A.at(j); i = j-1;
    while( i >= 0 && A.at(i) > key ) {
      A.at(i+1) = A.at(i); --i;
    }
    A.at(i+1) = key;
  }
}

int main() {
  unsigned int i;
  // getting the amount of elements
  std::cout << "Give the size of the array 0...: "; std::cin >> i;
```

```
  Array A(i); // creating the array
  // reading in the elements
  for( i = 0; i < A.size(); ++i ) {
    std::cout << "Give A[" << i+1 << "]: ";
    std::cin >> A.at(i);
  }
  insertionSort( A );   // sorting

  // print nicely
  for( i = 0; i < A.size(); ++i ) {
    if( i % 5 == 0 ) {
      std::cout << std::endl;
    }
    else {
      std::cout << " ";
    }
    std::cout << A.at(i);
  }
  std::cout << std::endl;
}
```

The program code is significantly longer than the pseudocode. It is also more difficult to see the central characteristics of the algorithm.

This course concentrates on the principles of algorithms and data structures. Therefore using program code doesn't serve the goals of the course.

$\Rightarrow$ From now on, program code implementations are not normally shown.