Algorithms and Data Structures 2

1 - Introduction

Dr Michele Sevegnani

School of Computing Science
University of Glasgow

michele.sevegnani@glasgow.ac.uk

Outline

- **Admin**
  - Course structure
  - Materials
  - Contents
  - Assessment
  - Context: prerequisites and relation with other courses
- **Foundations**
  - What is an algorithm?
  - What is a data structure?
  - Why studying algorithms is an essential aspect of the CS curriculum
  - A first example: array sorting

Course structure

- **22 lectures (2 per week)**
  - Tuesdays at 11:00 (offline video lectures)
  - Thursdays at 11:00 (online live recap and Q&A)

- **1-hour weekly labs at various times with different tutors**
  - Start week 2 (week beginning 18th January)
  - Timetable will be posted on moodle
  - Via Teams

Materials

- **Lecture slides**
  - Available on Moodle and Teams files at the beginning of each week
- **Video lectures**
  - Available on Moodle and Teams files
- **Lab sheets**
  - Via Moodle and Teams files
- **Course text**
  - Goodrich, Michael T., Roberto Tamassia, and Michael H. Goldwasser. *Data structures and algorithms in Java*. John Wiley & Sons, 2014
- **Other suggested texts**
  - Cormen, Thomas H., et al. *Introduction to algorithms*. MIT press, 2009
  - Sedgewick, Robert. *Algorithms in Java, Parts 1–4 (Fundamental Algorithms, Data Structures, Sorting, Searching)*. Addison Wesley, 2002

Contents

Lecture	Topics
1	Introduction
2	Analysis techniques
3	
4	Sorting algorithms Recursion
5	
6	
7	
8	Linked lists
9	
10	
11	Abstract Data Types Queues, Stacks
12	

Lecture	Topics
13	Lists and iterators
14	
15	Maps and Hash tables
16	
17	Trees Binary search trees Balanced trees
18	
19	
20	
21	Revision
22	

May be subject to minor changes

Assessment

- **Final exam (3 hours)**
  - Counts for 80%
  - We will go over past exams during the revision classes
- **Two assessed coursework assignments**
  - Count for 20% (10% each)
  - Build on work carried out during tutorials

Context

- **First course in Algorithms and Data structures for students with experience with programming in Java**
- **Introduces new concepts such as**
  - Data structures
  - ADTs (abstract data types)
  - Algorithms
  - Complexity analysis (to study algorithms' performance)
- **Prerequisites: knowledge of the basics of Java and basic calculus**
- **Main concepts will be revised before use**

Context

- **Object-Oriented Software Engineering (OOSE2)**
  - Will use some of the data structures and algorithms we introduce
- **Algorithmics I and Algorithmics II**
  - Extends the algorithms side, assumes knowledge of the data structures and complexity analysis learned in this course
- **Project work in Levels 3 and/or 4**
- **Basic algorithms and data structures are important in many other courses**
  - Network routing protocols
  - Operating systems scheduling algorithms
  - Information retrieval
  - ...

Foundations

What is an algorithm?

- **An algorithm is a step-by-step procedure for solving a problem in a finite amount of time**
  - Alternatively: a well-defined computational procedure that takes some value (or set of values), as input and produces some value (or set of values), as output (Cormen)
- **Some examples**
  - Sorting
  - String matching
  - Finding the greatest common divisor of two numbers (Euclid's algorithm)
- **An algorithm can be specified in many ways**
  - Natural language (English)
  - Computer program (implementation) using e.g. Java or pseudocode
  - Hardware design

Practical applications

- **Internet**
  - Routing protocols (finding good routes on which the data will travel)
  - Indexing for search engines
  - Public-key cryptography
  - Data compression
- **Biology**
  - DNA analysis
- **Optimisation problems and AI**
- **Navigation**
  - Shortest route
  - Path finding
- **The interview process in most tech companies often involves questions on algorithms!**

What is a data structure?

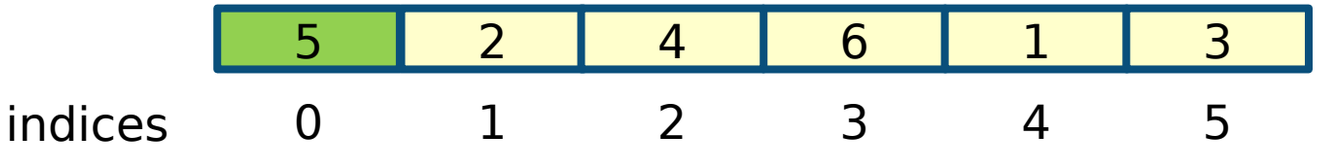
- **A data structure is a method to store and organise data in order to facilitate access and modifications**
- **Most algorithms involve the creation of data structures to organise the data involved in the computations**
- **Examples**
  - Arrays
  - Linked lists
  - Stack
  - Binary trees
  - …
- **No single data structure works well for all purposes**
- **Important to know limitations and strengths of several of them**

Example: array sorting

- **Input: an array A of integers (with indices between 0 and n-1)**
- **Output: a permutation of the input such that A[0] ≤ A[1] ≤ … ≤ A[n-1]**
- **Description of the algorithm as a program written in pseudocode**

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

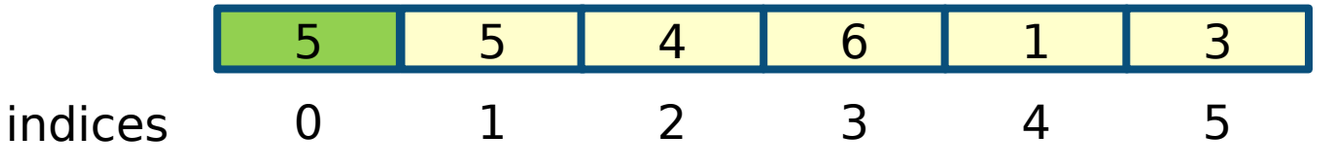
Operation of INSERTION-SORT



n = 6
j = 1
key = 2
i = 0

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

Operation of INSERTION-SORT

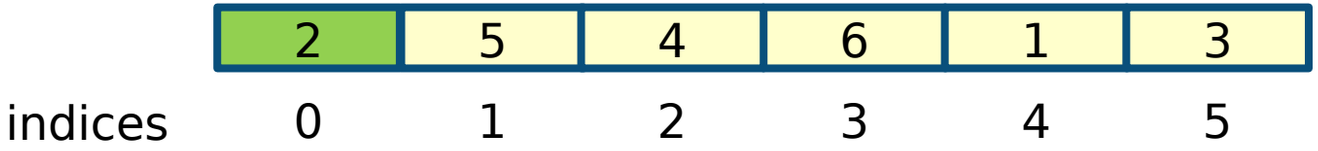


n = 6
j = 1
key = 2
i = -1

Exit while loop

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

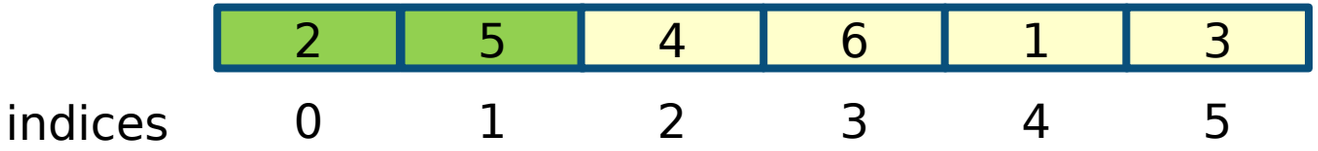
Operation of INSERTION-SORT



n = 6
j = 1
key = 2
i = -1

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

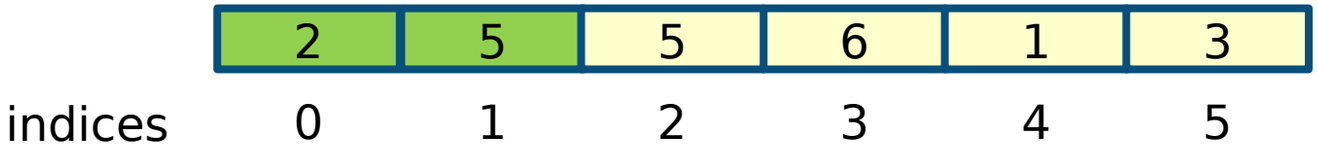
Operation of INSERTION-SORT



n = 6
j = 2
key = 4
i = 1

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

Operation of INSERTION-SORT

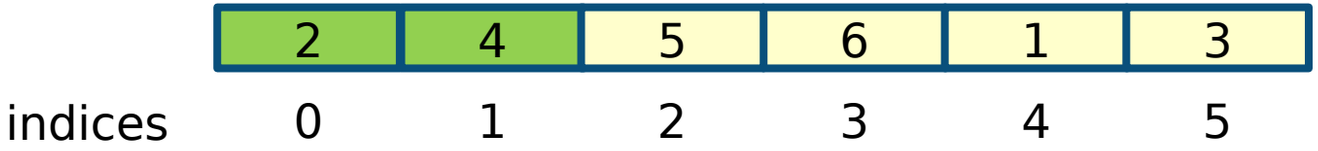


n = 6
j = 2
key = 4
i = 0

Exit while loop

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

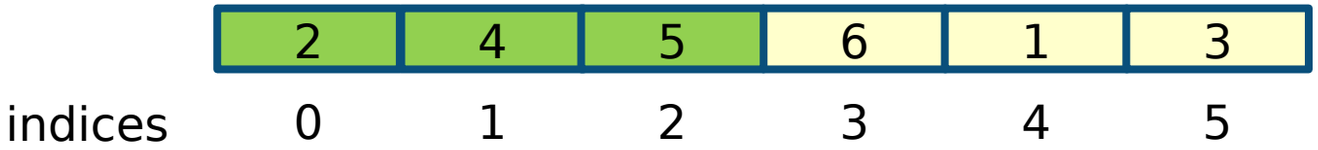
Operation of INSERTION-SORT



n = 6
j = 2
key = 4
i = 0

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

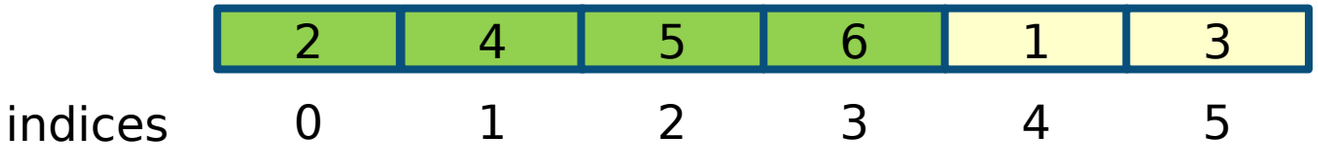
Operation of INSERTION-SORT



n = 6
j = 3
key = 6
i = 2 Exit while loop

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

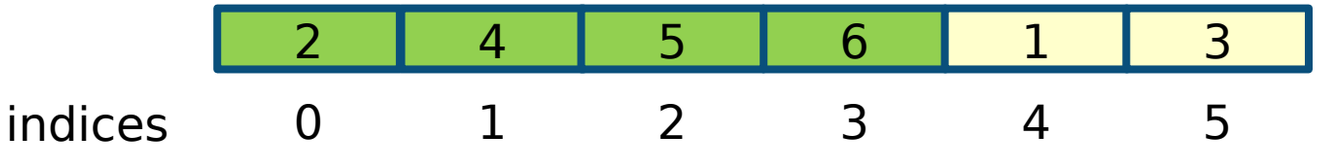
Operation of INSERTION-SORT



n = 6
j = 3
key = 6
i = 2

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

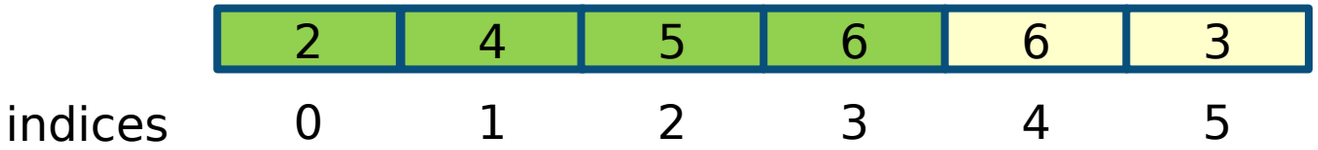
Operation of INSERTION-SORT



n = 6
j = 4
key = 1
i = 3

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

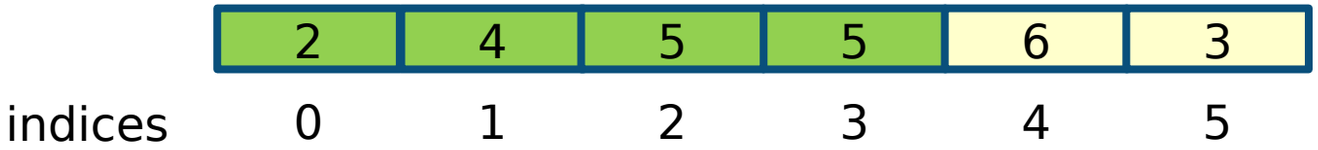
Operation of INSERTION-SORT



n = 6
j = 4
key = 1
i = 2

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

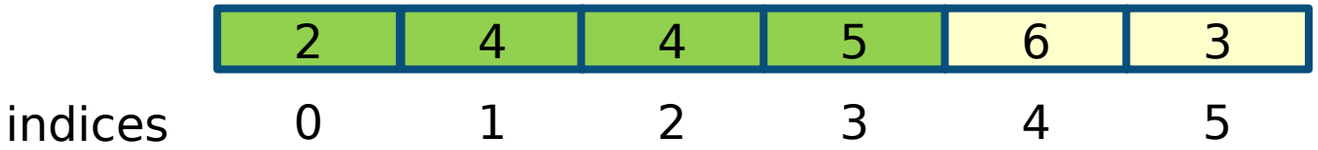
Operation of INSERTION-SORT



n = 6
j = 4
key = 1
i = 1

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

Operation of INSERTION-SORT



n = 6
j = 4
key = 1
i = 0

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

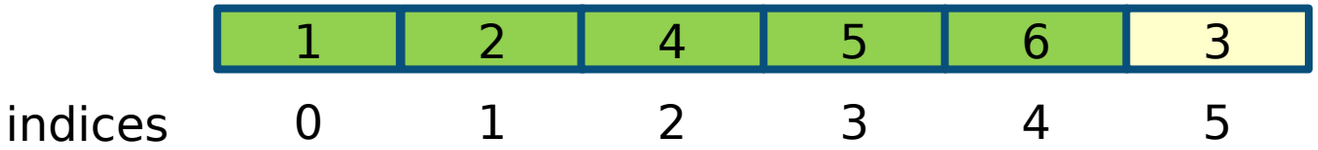
Operation of INSERTION-SORT

n = 6
j = 4
key = 1
i = -1

Exit while loop

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

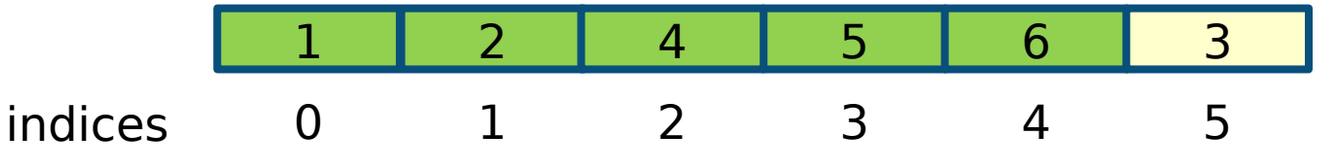
Operation of INSERTION-SORT



n = 6
j = 4
key = 1
i = -1

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

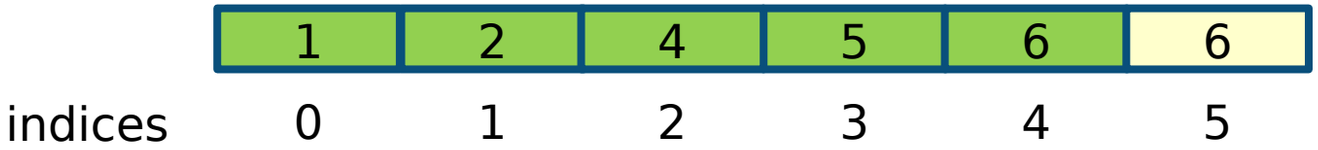
Operation of INSERTION-SORT



n = 6
j = 5
key = 3
i = 4

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

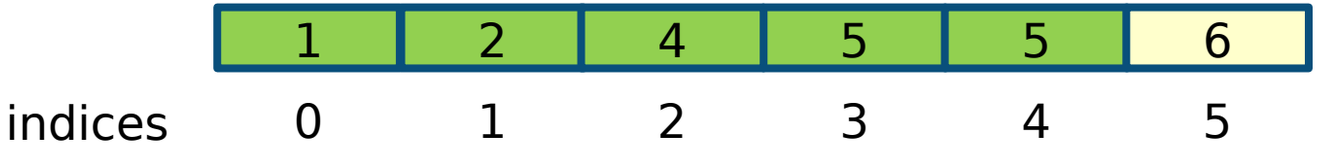
Operation of INSERTION-SORT



n = 6
j = 5
key = 3
i = 3

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

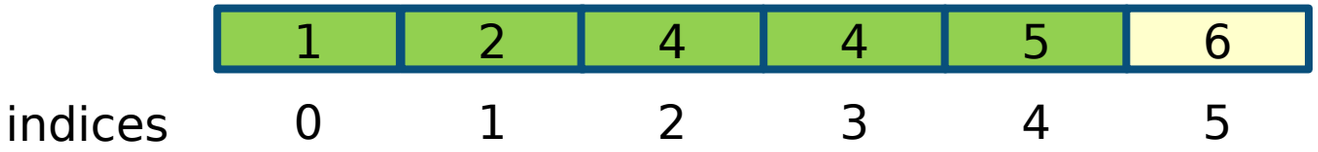
Operation of INSERTION-SORT



n = 6
j = 5
key = 3
i = 2

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

Operation of INSERTION-SORT

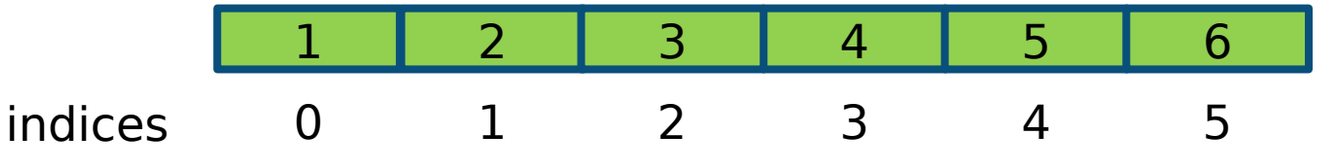


n = 6
j = 5
key = 3
i = 1

Exit while loop

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

Operation of INSERTION-SORT



n = 6
j = 5
key = 3
i = 1

Termination

```
INSERTION-SORT(A)
  for j = 1 to n-1
      key := A[j]
      i := j-1
      while i ≥ 0 and A[i] > key
        A[i+1] := A[i]
        i := i-1
      A[i+1] := key
```

Properties of INSERTION-SORT

- **Efficient algorithm for sorting a small number of elements**
  - We will prove this formally in the next lectures
- **Stable: it does not change the relative order of elements with equal keys**
  - Exercise: check this fact
- **In-place: it only requires a constant amount of additional memory space**
  - Besides the space for input array A, it only stores variables i,j and key

Algorithm analysis

- **Correctness: an algorithm is correct if for every input instance it halts with the correct output**
  - We say that a correct algorithms solves a given computational problem
- **The running time of an algorithm typically grows with the input size**
  - For huge instances, we want to develop methods to use time and space as efficiently as possible
- **Algorithm analysis allows us to predict the resources that the algorithm requires**
- **In the next classes, we will study the mathematical tools we need to carry out this kind of analysis**

Summary

- **What is an algorithm?**
- **Definition of data structure**
- **A first example: INSERTION-SORT**
- **Properties**
  - Stable
  - In-place
- **Pseudocode**
- **Algorithm analysis**