G6007 – Programming Concepts

Dr Philip Saville

Organisation 1

This is the first time I'm teaching this course

- Any issues, email me: p.saville@sussex.ac.uk
- Any suggestions, email me: p.saville@sussex.ac.uk
- Panopto, PIN, solutions $\rightsquigarrow$ remind me!

  (but come to the lectures!)

Organisation 2

- Lectures
- Large Group Tutorials - worksheets
- Helpdesk / small group tutorials
- Self-study exercises
- Assessment: unseen examination

Note:

- 15 credits
- Check your timetable on Sussex Direct
- Assume a register will always be taken

What you need to do

- Come to the lectures and large group tutorials
- Work through the worksheets and homework
- Read the handouts
- Highly recommended: implement the things we do in Java / Python

Reading list

It is important to read the notes and work through the exercise sheets for this module. This is all you need.

If you want to read around the subject, here are some suggestions:

- D. Harel. Algorithmics: The Spirit of Computing. Addison Wesley, 2012.
- G. Dowek. Principles of Programming Languages. Springer 2009.
- D. Makinson. Sets, Logic and Maths for Computing. Springer 2020.

More details are on the module web page, and this will be updated throughout the semester.

Module outline

An introduction to algorithmic problem solving. It will answer the following questions:

- What is a problem specification, an algorithm, a computation?
- How does one develop an algorithm?
- How can one rigorously argue that an algorithm computes correct solutions to a given problem?
- How can one measure the efficiency of an algorithm and the complexity of a problem?

Module outline, continued

- Focus is on algorithmic thinking, not coding.
- Basic data structures are explained in an informal way: tuples, lists, and arrays. Searching, sorting and other simple (and intuitive) algorithms.
- Two important properties of algorithms are *correctness* and *complexity*. Algorithms should only compute correct solutions of a problem. To establish correctness, some relevant (propositional and predicate) logic is introduced in an informal style again (focusing on logical reasoning principles rather than logical calculi).
- The concept of time complexity of an algorithm is presented and asymptotic complexity classes are discussed.

Why?

- Starting point for many fields in computer science
  (programming languages, verification, complexity, algorithms, . . .)
  and for the other CS modules at Sussex
- About how to design an algorithm to solve a problem and how to see if it's a good algorithm
- Languages change fast; concepts change slow
  in your own study time: implement what we do here in Java / Python

This lecture

- Understand the difference between a specification and an implementation
- Have a sense of what an algorithm is
- See lots of (somewhat silly) examples

Next lecture, and in the seminars: start to work on examples that look more like programs.

Specifications and algorithms

The high-level picture:

- A specification is about what you want done:
  it's the problem to solve
- An implementation is about how you do it:
  it's how you solve it
- An algorithm is a special kind of solution
  to an algorithmic problem

We'll start off thinking very generally, then gradually specialise to programs.

Specifications

What is a specification?

1. A description of **WHAT** a system is supposed to do
2. Expressed in terms meaningful to the **USER** of the system
3. Independent from any details of how the system actually works

What is a system ?

What is a system?

A physical process for carrying out a range of tasks

- computer
- web service
- assembly plant
  production of dvds, cars, chips, . . .
- weaving machine
  construction of carpets, rugs, . . .
- cook
  preparing of cakes, spaghetti dishes, . . .
- home help
  for preparing coffee, cleaning house, . . .

Specifications versus Implementations

Two distinct levels of task descriptions

Implementation: a detailed description of the various steps
the system must perform to carry out the task
how the task is performed

Specification: a high-level external description
what is the task to be performed

Example

Specification: A distributed database for handling health
records of UK population
accessible to doctors, and hospital staff

Implementation: a long list of hardware and software systems,
their functionality, their interconnection, . . .

Describing tasks

Example

Specification: Bake a chocolate mousse
Implementation:

1. Melt 8 ounces of chocolate.
2. When melted stir in 4 cups of powdered sugar and 2 tablespoons of butter.
3. In a bowl beat 6 egg yolks until thick
4. and so on . . .

Example

Specification: Build a assembly plant for producing aircraft capable of carrying 600 passengers from London to Sydney non-stop
Implementation: See Airbus production facilities at Toulouse

Large and small undertakings

Programming in the large

- construction of large complex systems
- large programming teams; extended development times; managerial oversight
- partitioning of system into modules; module interaction problems; many programmers; many modules; many users; many tools; many versions; many years

Programming in the small

- individual software component
- well-defined, well-specified, algorithmic problems
- one person task; use of simple programming constructs

This module is only concerned with *programming in the small*

What is an (algorithmic) problem?

Specifying an algorithmic problem requires

1. a characterisation of all legal inputs - possibly an infinite collection
2. a description of the required outputs as a function of the inputs

Example: Largest element in a set

Legal inputs: Any finite set of elements; elements must be *comparable*

Required outputs: the largest element in the input set

Example problem specifications

Calculating salary bill

Legal inputs: any list of employee records; each record contains their salary

Required outputs: the total salary bill

Sorting:

Legal inputs: Any finite set of elements; elements must be comparable

Required outputs: a list

- containing exactly the same elements as in the input set
- whose elements appear in increasing order: $e_1, \ldots, e_n$, satisfying $e_i \leq e_j$ whenever $i \leq j$

What is an algorithm?

A preliminary definition:

An algorithm consists of

an ordered list of basic operations, which for every valid input

- determines a finite sequence of operations to be performed
- halts
- produces some output

Recipe for a *system* to carry out a task

Named after al-Khwarizmi – 7th century Persian mathematician

Example algorithm: multiplication of numbers

Problem specification:

Legal inputs: any two numbers; the *multiplicand* K and the *multiplier* M

Required output: the multiplicand repeatedly added to itself multiplier times ($K + K + \cdots + K$)

Example algorithm: multiplication of numbers

Problem specification:

Legal inputs: any two numbers; the *multiplicand K* and the *multiplier M*

Required output: the multiplicand repeatedly added to itself multiplier times ($K + K + \cdots + K$)

Algorithm?

1. get a calculator
2. type in both inputs
3. press the * button
4. return result as answer

Example algorithm: multiplication of numbers

Problem specification:

Legal inputs: any two numbers; the *multiplicand K* and the *multiplier M*

Required output: the multiplicand repeatedly added to itself multiplier times ($K + K + \cdots + K$)

Algorithm?

1. get a calculator
2. type in both inputs
3. press the * button
4. return result as answer

Algorithm?

1. find a computer programmer
2. get them to write a Java program
3. give them the two numbers
4. ask them for the result of running the program

High-school algorithm for multiplication

1. write down multiplicand K
2. underneath write down multiplier M
3. draw line underneath
4. multiply K by leftmost digit of M write down result
5. repeat this operation for each digit of M, left to right shift one space to right when writing down result
6. add up all resulting numbers and output the sum

```
123
456
------
492
 615
  738
------
56088
```

Is this correct? Why does it work? Is this how you would have done it?

What is an algorithm? (preliminary)

Specifying an algorithmic problem requires

1. a characterisation of all legal inputs - possibly an infinite collection
2. a description of the required outputs as a function of the inputs

An algorithm consists of

an ordered list of basic operations, which for every valid input

- determines a finite sequence of operations to be performed
- halts
- produces some output

What is an algorithm? (preliminary)

An algorithm consists of

an ordered list of basic operations, which for every valid input

- determines a finite sequence of operations to be performed
- halts
- produces some output

Some things to think about:

- Still need to explain what a basic operation is
- Shouldn’t be too vague
- Is it always clear what counts as ‘basic’ or ‘vague’?

Next lecture: we’ll look at some more examples, and extract more points from those

Algorithm?

Making an omelette

Legal input: 3 eggs, 100 gms cheese, 100cl milk, 10 gms butter, 2 gms salt, 2 gms pepper

Required output: A lovely omelette

- break eggs into bowl
- add milk, salt, pepper
- whisk contents rapidly for 3 minutes
- melt butter in a pan at 120^o C
- transfer contents of bowl to pan, with the cheese
- stir contents of pan for 4 minutes, at 120^o C
- serve on hot plate

Algorithm?

Making an omelette

Legal input: 2/3 eggs, some cheese and milk, small amount of butter, pinch of salt, pinch of pepper

Required output: A lovely omelette

- break eggs into bowl
- add milk, salt, pepper
- whisk contents rapidly until thoroughly mixed
- melt butter in a pan at low heat transfer contents of bowl to pan, with the cheese
- stir contents of pan until consistent
- serve on hot plate

Algorithm?

Making an omelette

Legal input: 2/3 eggs, some cheese and milk, small amount of butter, pinch of salt, pinch of pepper

Required output: A lovely omelette

- break eggs into bowl
- add milk, salt, pepper
- whisk contents rapidly until thoroughly mixed
- melt butter in a pan at low heat transfer contents of bowl to pan, with the cheese
- stir contents of pan until consistent
- serve on hot plate

The right level of description depends on the user

Algorithms?

Bake a cake

- bake a light sponge cake
- prepare chocolate icing
- cover cake with icing
- set aside to settle

Algorithms?

Recette d'omelette Ingredients : 3 oeufs, 100g de fromage, 1l de lait, 10g de beurre, sel, poivre

1. casser les oeufs dans une jatte
2. ajouter le lait, une poignee de sel et de poivre
3. battre le tout energiquement 3 minutes
4. dans une poele amenee a 120^o C, faire fondre le beurre
5. rajouter le contenu de la jatte dans la poele
6. laisser cuire 4 minutes a 120^o C en remuant
7. servir sur une assiette chaude

Algorithms?

Recette d’omelette Ingredients : 3 oeufs, 100g de fromage, 1l de lait, 10g de beurre, sel, poivre

1. casser les oeufs dans une jatte
2. ajouter le lait, une poignee de sel et de poivre
3. battre le tout energiquement 3 minutes
4. dans une poele amenee a 120^o C, faire fondre le beurre
5. rajouter le contenu de la jatte dans la poele
6. laisser cuire 4 minutes a 120^o C en remuant
7. servir sur une assiette chaude

Language matters!

Calculating salary bill

Problem description

Legal inputs: any list of employee records; each record contains their salary

Required output: the total salary bill

Algorithm

- Make a note of number 0
- Proceed through the employee list, each time
    - adding salary to noted number
- When end of list is reached
    - output noted number

Name	Salary
Tom	12000
Mary	17000
Shaun	16000
...	
Lisa	23000

What is an algorithm? (preliminary)

An algorithm consists of

an ordered list of basic operations, which for every valid input

- determines a finite sequence of operations to be performed
- halts
- produces some output

What is an algorithm? (revisited)

An algorithm consists of

an ordered list of basic operations, which for every valid input

- determines a finite sequence of operations to be performed
- halts
- produces some output

Requirements

- basic operations must be
    - elementary and unambiguous – require no creativity
    - understandable by system executing the algorithm
- level of detail appropriate to system executing the algorithm

Some of these things depend on the context

Choice of algorithms

Lots of algorithms for the same problem

Choice of algorithms

Lots of algorithms for the same problem

High-school algorithm for multiplication (UK version)

1. write down multiplicand *K*
2. underneath write down multiplier *M*
3. draw line underneath
4. multiply *K* by rightmost digit of *M* write down result
5. repeat this operation for each digit of *M*, right to left shift one space to left when writing down result
6. add up all resulting numbers
7. output the sum

Example: multiply 123 by 456

$$
\begin{array}{ccccc}
 & & 1 & 2 & 3 \\
 & & 4 & 5 & 6 \\
\hline
 & & 7 & 3 & 8 \\
 & 6 & 1 & 5 & \\
4 & 9 & 2 & & \\
\hline
5 & 6 & 0 & 8 & 8
\end{array}
$$

Choice of algorithms: Multiplication à la russe

1. write down multiplicand *K* and multiplier *M* side by side
2. form two columns underneath by repeating the following until number in left-hand column is 1:
   2.1 divide number in left-hand column by 2, ignoring fractions
   2.2 double number in right-hand column
3. cross out each row where left-hand column is even
4. add up remaining numbers in right-hand column

Basic actions

- multiply by 2
- divide by 2
- add up a list of numbers

123	456	456
61	912	912
30	1824	−−
15	3648	3648
7	7296	7296
3	14592	14592
1	29184	29184
		56088

Choice of algorithms

Two multiplication algorithms: which would you prefer to do? Which is better?

Should we prefer one over the other?

Algorithms can get complicated

Counting happy employees

Legal inputs: any list of employee records; each record contains their salary and name of boss

Required output: number of employees earning more than their boss

Algorithm

1. Make a note of counter, initially set to 0
2. Proceed through the employee list, each time
    - 2.1 Note name of boss, and salary of current employee
    - 2.2 Find record of boss in list
    - 2.3 If salary of boss is less than that of current employee, increase the counter
3. When end of list is reached, output value of counter

Name	Salary	Boss
Tom	1200	James
Mary	17000	Cindy
Shaun	16000	Tom
...	...	...
Lisa	2300	Mary

Is this an algorithm?

What this module is about

- The description of algorithmic problems
- Their solution using algorithms
- The design of algorithms
- The comparison between different algorithms for the same problem
  - How do we choose between them?
  - When is one algorithm better than another?
- Reasoning about algorithms
  - Does the algorithm solve the given problem?
  - In all cases?
  - How can we guarantee this?

Step from local to global facts about program behaviour

Summary

- Introduction to the way we want to think about problems
- Many algorithms for the same problem
- Many ways to talk about the same algorithm:
  different languages, different levels of detail

Next topic:

- Describing algorithms in pseudo code

To Do:

1. Check the resources on the web page and look into the recommended texts
2. Lecture notes and exercises will be available before the sessions