Introduction to Programming

Dr Ben Dudson

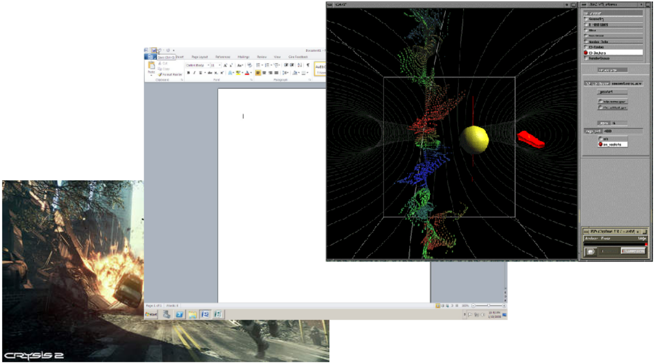
Department of Physics, University of York

21st January 2011

Why programming?

Why do people learn programming?

- Computers are everywhere (you may have noticed this)
- Games, word processors, web browsers, simulations,...
  Programming is a skill needed to create all these things



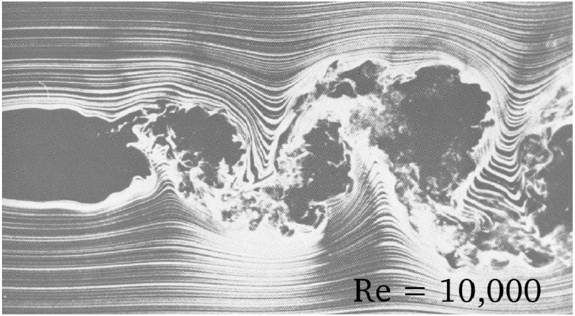
Why programming?

Why do people learn programming?

- Computers are everywhere (you may have noticed this)
- Games, word processors, web browsers, simulations,... Programming is a skill needed to create all these things
- So why should *you* learn how to do this?
  - Problem solving: explaining how to solve a problem to a computer means you have to completely understand it yourself
  - A “thinking” skill rather than a “computer” skill: designing a program requires creativity as well as logical and critical thinking
  - Being able to write your own programs can make your life easier: you can get a computer to do exactly what you want
  - It’s highly likely that you’ll need some level of programming as part of your research or career

Programming in physics

- Many problems in physics and engineering cannot be solved analytically, particularly messy real-life situations
  - **Example**: (slow) Flow of fluids around a cylinder can be calculated analytically, but flow through a jet engine cannot
  - **NB**: Analytic skills are still vital to derive the equations, and to understand the physics by studying simpler systems



Programming in physics

- Many problems in physics and engineering cannot be solved analytically, particularly messy real-life situations
  - **Example**: (slow) Flow of fluids around a cylinder can be calculated analytically, but flow through a jet engine cannot
  - **NB**: Analytic skills are still vital to derive the equations, and to understand the physics by studying simpler systems
- Often these problems are very specific
  - Standard programs (e.g. Excel) can't solve them, not fast enough, not enough memory etc.
  - Need to write our own programs
  - Almost every researcher will need to write programs at some point, probably quite regularly.

Programming in physics

- Many problems in physics and engineering cannot be solved analytically, particularly messy real-life situations
  - **Example**: (slow) Flow of fluids around a cylinder can be calculated analytically, but flow through a jet engine cannot
  - **NB**: Analytic skills are still vital to derive the equations, and to understand the physics by studying simpler systems
- Often these problems are very specific
  - Standard programs (e.g. Excel) can't solve them, not fast enough, not enough memory etc.
  - Need to write our own programs
  - Almost every researcher will need to write programs at some point, probably quite regularly.
- Programming is a tool to make your life easier

Course aims

- Problem solving
  - The hardest part is to turn a problem you want to solve into a set of steps to solve it
  - A skill which is useful in many areas
- Programming in the Python language
- Solving physics problems using computers
- Visualising results in 2D and 3D

Course outline

- Lectures
  - Today: Introduction and problem solving

- Marks: worth 50% of the Professional Skills I module
  - Problem sheet (10%) on problem solving

Course material

`http://www-users.york.ac.uk/∼bd512/teaching.shtml`

Course outline

- Lectures
  - Today: Introduction and problem solving
  - Next time: Programming basics and Python

- Marks: worth 50% of the Professional Skills I module
  - Problem sheet (10%) on problem solving

Course material

`http://www-users.york.ac.uk/~bd512/teaching.shtml`

Course outline

- Lectures
  - Today: Introduction and problem solving
  - Next time: Programming basics and Python
- Labs - 2 hours per week in room **G169**
  - **Group PS1** 9:15 - 11:15
  - **Group PS2** 11:15 - 13:15
- Marks: worth 50% of the Professional Skills I module
  - Problem sheet (10%) on problem solving
  - Lab 1 (15%), 28th January
  - Lab 2 (15%), 4th February
  - Lab 3 (20%), 11th February
  - Lab 4 (20%), 18th February
  - Lab 5 (20%), 25th February

Course material

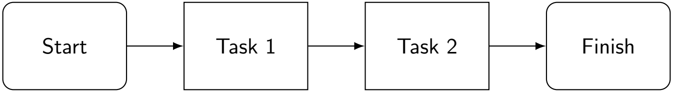
`http://www-users.york.ac.uk/∼bd512/teaching.shtml`

Problem solving with computers

- Computers are machines which can be re-configured to do a wide variety of tasks: Word processor, scientific number-cruncher, internet telephone, games console, ...
  - In this course you will learn how to make computers do what **you** want them to
- Computers are very stupid and have no common sense. They must be given detailed and unambiguous, step-by-step instructions
  - These instructions are called an algorithm
  - To be understood, these instructions must be written in a programming language

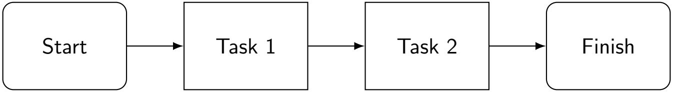
Computer programs

Programs are a set of instructions which are executed (carried out) one after another to reach a result.



Computer programs

Programs are a set of instructions which are executed (carried out) one after another to reach a result.



Yorkshire Pie recipe:

- In a skillet, cook beef and onion over medium heat until meat is no longer pink; drain.
- Add vegetables and 1/4 cup of gravy; set aside
- Place butter in a 10-in. pie plate and place in a 425 oven until melted
- In a bowl, beat milk and eggs
- ...

Computer programs

Unfortunately, computers have no common sense...

- In a skillet, cook beef and onion over medium heat until meat is no longer pink; drain.
- Add vegetables and 1/4 cup of gravy; set aside
- Place butter in a 10-in. pie plate and place in a 425 oven until melted
- In a bowl, beat milk and eggs
- ...

What is a skillet? Where do I get one? Which one? What is "medium" heat? What is the color code for pink? How do you drain a skillet?

The problems we want to solve are often quite vague. Computers need completely unambiguous step-by-step instructions.

Step 1: Identify the problem

In order to tell a computer what to do, we need to fully understand exactly what the problem is. If we don't then the instructions we give the computer won't work.

When given a problem to solve:

- Read carefully through the whole description - English instructions aren't always in order.
- Understand the situation, possibly drawing a diagram
- List the inputs and outputs, and give each one a name
- If you're modelling a physical situation, list which quantities need to be kept track of. Give these quantities names.

Example 1: Calculating marks

Given the scores for the exam, two problem sheets, and a report, write a program that will compute for the final mark based on the following computation: 50% for the exam $+10\%$ average of 2 problem sheets $+40\%$ project

Example 1: Calculating marks

Given the scores for the exam, two problem sheets, and a report, write a program that will compute for the final mark based on the following computation: 50% for the exam +10% average of 2 problem sheets +40% project

- Inputs: Percentage marks for exam e, problem sheets s_1 and s_2 and project p
- Output: Overall percentage mark R
- Calculations: Average p_1 and p_2, add marks together

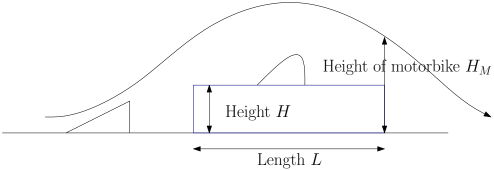
Exercise: Jumping the shark

Exercise: You want to jump over a shark tank on your motorbike which can do 140km/h. A shark tank salesman has given you a brochure with different sized tanks (sharks not included). Find the one which will allow you to jump over by the narrowest margin.

Exercise: Jumping the shark

Exercise: You want to jump over a shark tank on your motorbike which can do 140km/h. A shark tank salesman has given you a brochure with different sized tanks (sharks not included). Find the one which will allow you to jump over by the narrowest margin.

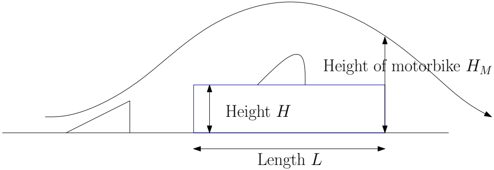
- First draw a diagram of the problem



Exercise: Jumping the shark

Exercise: You want to jump over a shark tank on your motorbike which can do 140km/h. A shark tank salesman has given you a brochure with different sized tanks (sharks not included). Find the one which will allow you to jump over by the narrowest margin.

- First draw a diagram of the problem



- Label quantities we need to know or calculate
  - Inputs: Height H and length L of each tank
  - Calculate: Height of the motorcycle's path H_M
  - Output: The tank which results in the smallest positive $H_M - H$

Step 2: Go through the problem step-by-step

Once you know what it is that you want to achieve

- Think through how you would solve the problem by hand with pen and paper. Try solving a small version of the problem and see how you do it.
- Imagine explaining what you have done to someone who takes everything completely literally.
- This set of instructions is called an algorithm

How do we tell a computer what to do?

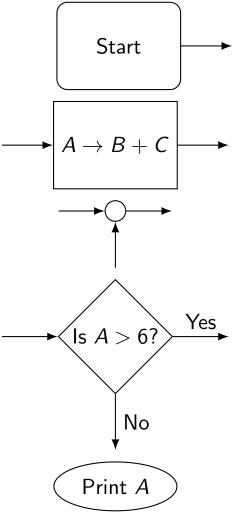
Computers can only perform a small number of tasks, and need our problem explained in terms of some simple operations

- Calculations like addition, subtraction, multiplication and division. You can also assume that you can have common mathematical functions like sin, log, *tanh*
- Yes or no decisions, comparing numbers e.g. "Is number A greater than B?"
- Loops - repeating the same set of steps over and over.

That's it! Everything a computer does from spreadsheets to games is just a combination of these things.

Flow charts

In this course, we're going to start by using flow charts:



Start and end points

processing step

Connector

Make a decision

Get input or print output

These are useful ways to draw the steps. Later we'll see how they translate into Python...

Problem-solving strategies

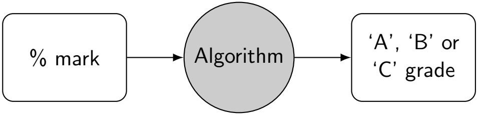
Unfortunately there is no algorithm for creating algorithms, and so some creativity and experimentation is needed. There are some common strategies:

- Refinement. Break up a large problem into smaller pieces, then solve each one separately
- Work backwards. Work out what is needed to get the result, then what is needed to get those results and so on
- Transform into an already solved problem. Can your problem be changed or re-phrased as a problem which has already been solved?

Example: Decide a student's grade

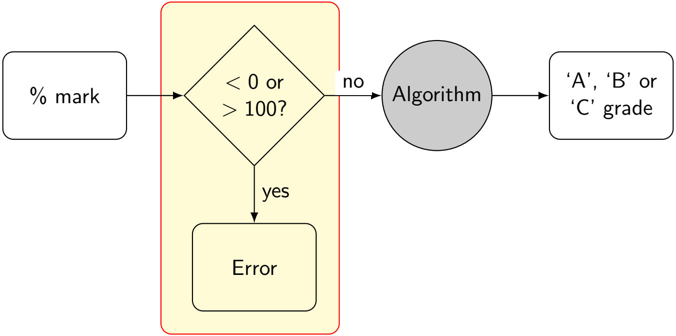
Problem description: Given a percentage mark, assign a student an 'A' if over 70%, 'B' if over 40% and 'C' otherwise.

Step 1: State what the inputs and outputs are



Example: Decide a student's grade

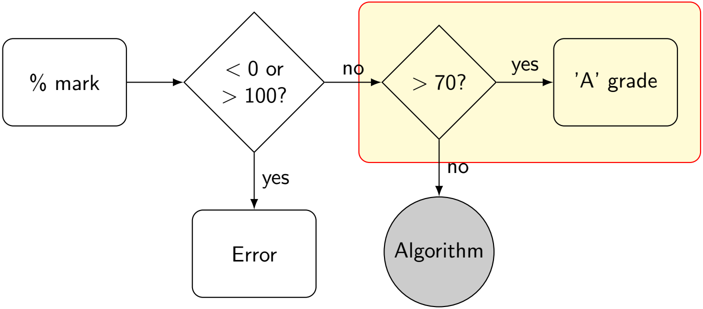
Problem description: Given a percentage mark, assign a student an 'A' if over 70%, 'B' if over 40% and 'C' otherwise.



Always check your inputs to make sure they are what you expect

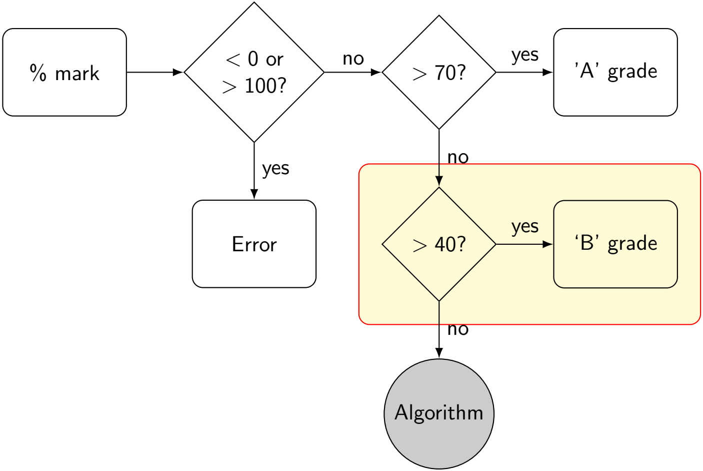
Example: Decide a student's grade

Problem description: Given a percentage mark, assign a student an 'A' if over 70%, 'B' if over 40% and 'C' otherwise.



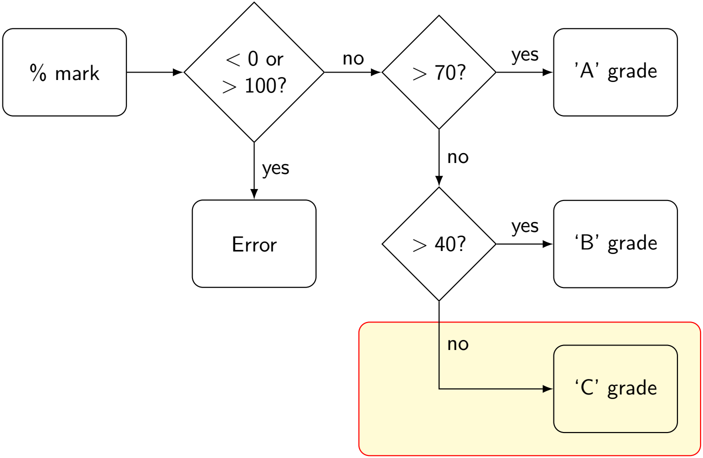
Example: Decide a student's grade

Problem description: Given a percentage mark, assign a student an 'A' if over 70%, 'B' if over 40% and 'C' otherwise.



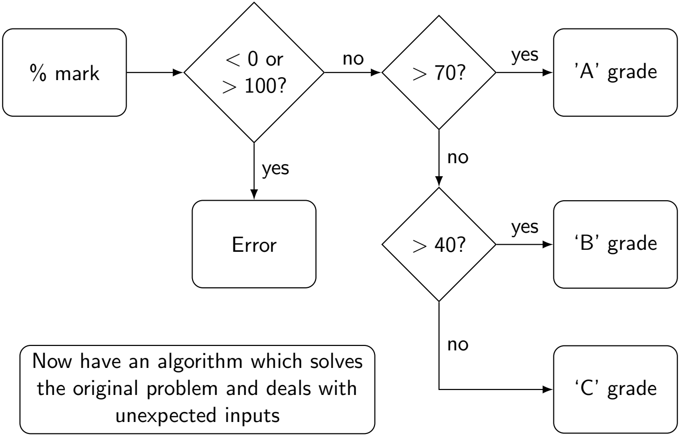
Example: Decide a student's grade

Problem description: Given a percentage mark, assign a student an 'A' if over 70%, 'B' if over 40% and 'C' otherwise.



Example: Decide a student's grade

Problem description: Given a percentage mark, assign a student an ‘A’ if over 70%, ‘B’ if over 40% and ‘C’ otherwise.



Now have an algorithm which solves the original problem and deals with unexpected inputs

More complicated example: Sorting numbers

If I gave you a list of numbers to sort from lowest to highest, how would you do it?

63, 2, 26, 19, 52, 84, 33, 6, 94

More complicated example: Sorting numbers

If I gave you a list of numbers to sort from lowest to highest, how would you do it?

63, 2, 26, 19, 52, 84, 33, 6, 94

Find the lowest number '2', and remove it from the list

63, 26, 19, 52, 84, 33, 6, 94

More complicated example: Sorting numbers

If I gave you a list of numbers to sort from lowest to highest, how would you do it?

63, 2, 26, 19, 52, 84, 33, 6, 94

Find the lowest number '2', and remove it from the list

63, 26, 19, 52, 84, 33, 6, 94

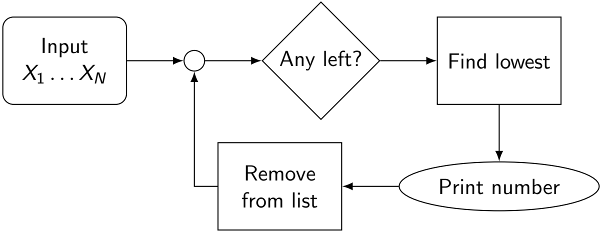
Repeat, removing '6'

63, 26, 19, 52, 84, 33, 94

and so on until there are none left.

More complicated example: Sorting numbers

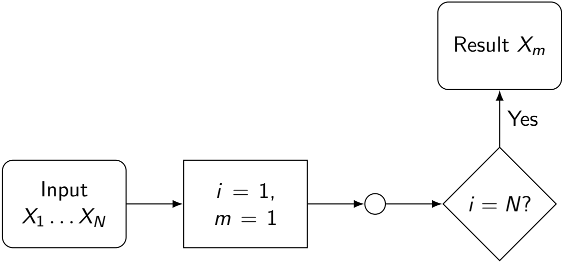
We could show this algorithm as



But how do we find the lowest number?

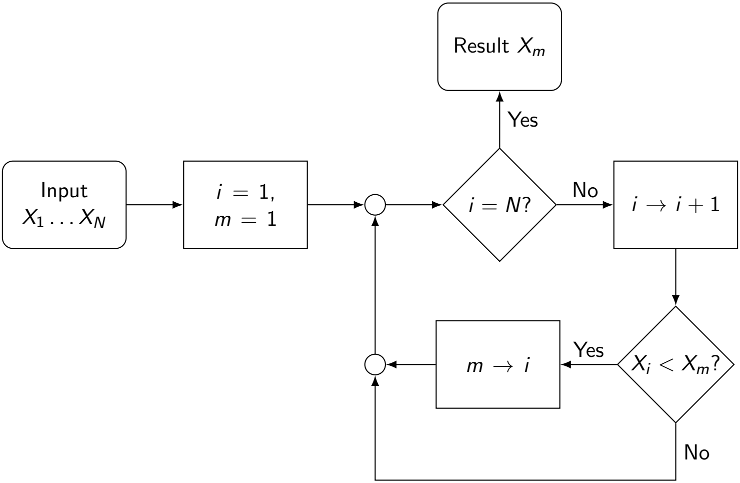
More complicated example: Sorting numbers

Finding the lowest number: Start from the beginning and go through each number, comparing it to the current minimum.



More complicated example: Sorting numbers

Finding the lowest number: Start from the beginning and go through each number, comparing it to the current minimum.



More complicated example: Sorting numbers

So how long does this take to do? How much longer would it take if I doubled the size of the list of numbers?

63, 26, 19, 52, 84, 33, 6, 94, 64, 82, 56, 65, 69, 1, 22, 60

More complicated example: Sorting numbers

So how long does this take to do? How much longer would it take if I doubled the size of the list of numbers?

63, 26, 19, 52, 84, 33, 6, 94, 64, 82, 56, 65, 69, 1, 22, 60

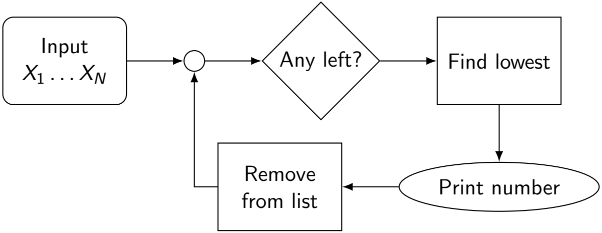
How about 10 times more?

63, 26, 19, 52, 84, 33, 6, 94, 64, 82, 56, 65, 69, 1, 22, 60, 67, 25, 38, 13, 25, 7, 73, 93, 37, 36, 37, 7, 65, 43, 66, 91, 79, 72, 46, 12, 95, 59, 33, 83, 73, 88, 18, 79, 59, 5, 14, 85, 36, 3, 43, 30, 35, 98, 70, 18, 83, 32, 8, 57, 60, 57, 69, 98, 48, 47, 58, 2, 34, 99, 28, 91, 66, 28, 23, 22, 56, 26, 16, 20

Often need to keep in mind how efficient our method is

More complicated example: Sorting numbers

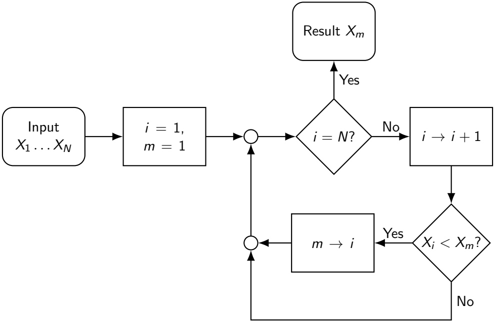
Start with the main loop:



- Each time we go around this loop, one number is removed from the list
- If we start with N numbers, then we will go around N times
- Each of these N times find the lowest number left in the list
- How does this find step depend on the size of the list?

More complicated example: Sorting numbers

How about our method for finding the smallest number?



Not interested in tasks which are only done once, as these will not take longer as we make the list bigger. Only need to look at loops.

- Each time, this searches through every number in the list
- The size of this list reduces each time: $N, N - 1, \ldots, 1$
- Total time therefore goes like $\frac{1}{2}N\left(N+1\right)$
- For large N, this is written $O\left(N^2\right)$

Divide...

Is there a way we can improve on this method? Every time we want to find the next highest, we're going through the entire list again. Instead, what we can do is split our big problem (sorting a big list) into two smaller problems (sorting smaller lists)

63, 26, 19, 52, 84, 33, 6, 94

Divide...

Is there a way we can improve on this method? Every time we want to find the next highest, we're going through the entire list again. Instead, what we can do is split our big problem (sorting a big list) into two smaller problems (sorting smaller lists)

63, 26, 19, 52, 84, 33, 6, 94

Pick a number *i* (e.g. the first one 63). Go through the list of numbers and split into two: those lower than *i* (red) and those higher (blue).

63, 26, 19, 52, 84, 33, 6, 94

Put all the red on the left, all the blue on the right

26, 19, 52, 33, 6 — **63** — 84, 94

Have now got two lists to sort, each (on average) half the length of the original

Divide...and Conquer

We can now repeat the trick with each of the separate lists

26, 19, 52, 33, 6 — 63 — 84, 94

26, 19, 51, 33, 6 — 63 — **84**, 94

Each smaller list again splits into two

19, 6 — **26** — 51, 33 — 63 — **84** — 94

Divide...and Conquer

We can now repeat the trick with each of the separate lists

26, 19, 52, 33, 6 — 63 — 84, 94

26, 19, 51, 33, 6 — 63 — **84**, 94

Each smaller list again splits into two

19, 6 — **26** — 51, 33 — 63 — **84** — 94

and again...

19, 6 — 26 — **51**, 33 — 63 — 84 — 94

Divide...and Conquer

We can now repeat the trick with each of the separate lists

26, 19, 52, 33, 6 — 63 — 84, 94

26, 19, 51, 33, 6 — 63 — **84**, 94

Each smaller list again splits into two

19, 6 — **26** — 51, 33 — 63 — **84** — 94

and again...

19, 6 — 26 — **51**, 33 — 63 — 84 — 94

to finally get:

6 — 19 — 26 — 33 — 51 — 63 — 84 — 94

Performance

- How long does this new method take?

Performance

- How long does this new method take?
- Each time we split the list in two, we need to go through all the numbers to see if they're greater than or less than our chosen number
  ⇒ for a list of length N we make around N comparisons each time it is split in two

Performance

- How long does this new method take?
- Each time we split the list in two, we need to go through all the numbers to see if they're greater than or less than our chosen number

  ⇒ for a list of length N we make around N comparisons each time it is split in two
- How many times do we need to split the list in 2 before we get down to lists of length 1?

$$N/2^x = 1 \Rightarrow x = \log_2(N)$$

- We split our lists in two about $\log_2 N$ times, and each time we need to make around N comparisons.

  ⇒ The time for our new algorithm goes like $O(N \log_2 N)$.

What difference does it make?

Why do we care whether an algorithm is $O\left(N^2\right)$ or $O\left(N\log N\right)$? Say it takes 1 second to sort 10 numbers using our first method, and (highly pessimistically) 10 seconds using our second method.

What difference does it make?

Why do we care whether an algorithm is $O\left(N^2\right)$ or $O\left(N\log N\right)$? Say it takes 1 second to sort 10 numbers using our first method, and (highly pessimistically) 10 seconds using our second method.

N	Time $O\left(N^2\right)$	Time $O\left(N\log N\right)$
10	1 s	10 s
100	100 s	200 s
1000	2hr 46 min	50 min
10,000	11 days 13 hr	11 hr
100,000	3 years 62 days	5 day 19 hr
1,000,000	316 years 321 days	69 days 11 hr

What difference does it make?

Why do we care whether an algorithm is $O\left(N^2\right)$ or $O\left(N \log N\right)$? Say it takes 1 second to sort 10 numbers using our first method, and (highly pessimistically) 10 seconds using our second method.

N	Time $O\left(N^2\right)$	Time $O\left(N \log N\right)$
10	1 s	10 s
100	100 s	200 s
1000	2hr 46 min	50 min
10,000	11 days 13 hr	11 hr
100,000	3 years 62 days	5 day 19 hr
1,000,000	316 years 321 days	69 days 11 hr

If you want to solve big problems, the constant at the front doesn't matter all that much. Even if we allow the N^2 algorithm a big head start, it quickly gets overtaken by the more efficient algorithm.

Divide and Conquer methods

- This type of algorithm which repeatedly breaks a big problem into smaller sub-problems is very common
- Often results in efficient algorithms
- You'll come across it in this course, particularly when sorting or searching are involved.
- A common theme in programming is this process of solving complicated problems by breaking them into smaller, simpler problems

Summary

- Programming is a useful skill, and can be very rewarding
- Problem analysis and solving is vital for programming
- This lecture we looked at some problem solving methods
- Next time, we'll start looking at the Python language

Summary

- Programming is a useful skill, and can be very rewarding
- Problem analysis and solving is vital for programming
- This lecture we looked at some problem solving methods
- Next time, we'll start looking at the Python language

This week's problem sheet

The problem sheet questions follow the examples in this lecture:

- Given a problem, identify the key inputs, outputs and steps
- Design an algorithm, drawing a flow diagram
- Analyse how efficient an algorithm is: $O(N)$, $O\left(N^2\right)$?