122COM: Introduction to algorithms

David Croft

Coventry University

david.croft@coventry.ac.uk

2017

Overview

Introduction

Introduction to algorithms module.

- What is an algorithm?

Introduction

Introduction to algorithms module.

- What is an algorithm?
- Not the same as code.
- Not the same as a program.

Introduction

Introduction to algorithms module.

- What is an algorithm?
- Not the same as code.
- Not the same as a program.

Task/algorithm/code

A task is a problem that needs to be solved.

- I.e. bake me a cake.

Task/algorithm/code

A task is a problem that needs to be solved.

- I.e. bake me a cake.

An algorithm is a generalised set of instructions to perform a specific task.

- A strategy to solve a given problem.
  - Many different strategies to solve same task.
- Like a recipe.

Task/algorithm/code

A task is a problem that needs to be solved.

- I.e. bake me a cake.

An algorithm is a generalised set of instructions to perform a specific task.

- A strategy to solve a given problem.
  - Many different strategies to solve same task.
- Like a recipe.

Code is a specific set of instructions to perform a specific task.

- An implementation of a strategy in a specific language/system.
- Have to adapt the recipe to your kitchen/oven/bowls/pans etc.

Fibonacci sequence algorithm

C

Task - calculate the fibonacci sequence.

Fibonacci sequence algorithm

C

Task - calculate the fibonacci sequence.

Algorithm

1. Starting with 0 and 1.
2. Sum the two numbers to make a third.
3. Discard the lowest number.
4. Repeat from step 2.

Fibonacci sequence algorithm

C

Task - calculate the fibonacci sequence.

Algorithm

1. Starting with 0 and 1.
2. Sum the two numbers to make a third.
3. Discard the lowest number.
4. Repeat from step 2.

Recursive Python

```
def fibonacci( a, b ):
    c = a + b
    a, b = b, c

    print( a )
    fibonacci( a, b )

fibonacci( 0, 1 )
```

Iterative C++

```
for( int a=0, b=1, c;
     a>=0;
     c=a+b, a=b, b=c )
{
    cout « a « endl;
}
```

Solvable/Unsolvable

C

Some problems we can solve perfectly.

- Easy problems.
  - Fibonacci sequence.
  - Searching algorithms.
  - Polynomial time.

Some problems we can't solve.

- Hard problems.
- Provably unsolvable.
  - Investigate the Halting State problem.

Heuristics

Some problems we could solve perfectly if only we had infinite computers/time.

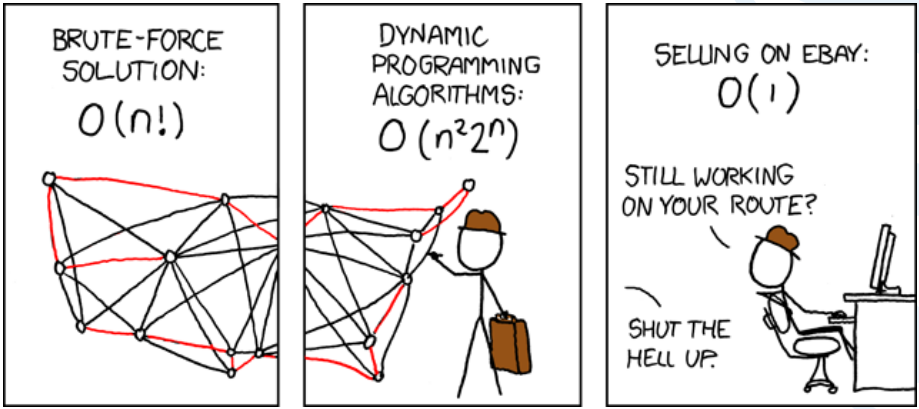
- Travelling salesman.
  - Hard problem, non-polynomial (will discuss in later weeks).
  - Can only solve very simple versions of the problem perfectly.
  - 5 cities = 120 possible solutions, 20 cities = 2 432 902 008 176 640 000 possible solutions.

Heuristic algorithms.

- Don't promise to find the best solution.
- Quickly find a 'good enough' solution.

Heuristics

A



https://xkcd.com/399/

Module content

Looking at searching and sorting algorithms in later weeks.

Will be tested on some algorithmic concepts.

- Implement simple algorithms.
- Describe advantages/disadvantages of certain algorithms.
- Big *O* notation.
  - How algorithms scale.
- Calculate an algorithm's *O*() notation.

Why do I care?

Everyone

- Thinking algorithmically is critical programming skill.
- Learning how to break down a problem into small steps.
  - Functional decomposition.
- Evaluate algorithms.
  - Does this algorithm actually work?
- Employability skill
  - Interview questions.

Recap

C

- What is an algorithm.
- Code vs. algorithms.
- Heuristics = good enough solutions.
  - Rules of thumb
- Polynomial = easy problems.
- Non-polynomial = hard problems.

The End