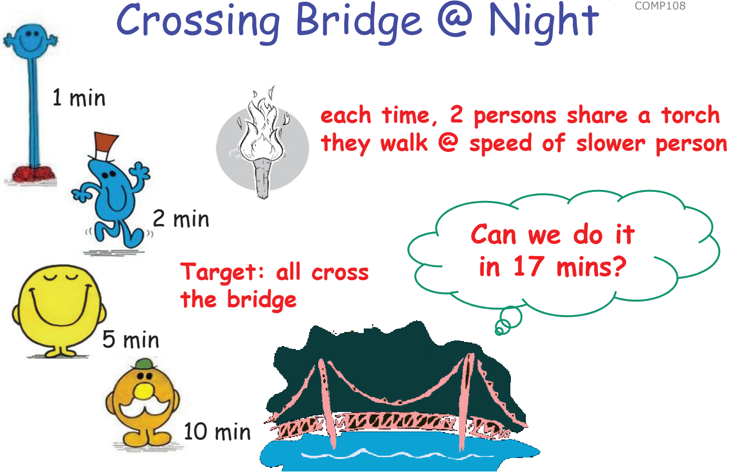
COMP108
Algorithmic Foundations

Basics

Prudence Wong

http://www.csc.liv.ac.uk/~pwong/teaching/comp108/201617

Crossing Bridge @ Night



each time, 2 persons share a torch
they walk @ speed of slower person

Target: all cross the bridge

Can we do it in 17 mins?

Module Information

Professor Prudence Wong

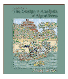
Rm 3.18 Ashton Building, pwong@liverpool.ac.uk

office hours: Tue 10-11am

Demonstrators

Mr Thomas Carroll, Mr Reino Niskanen

References



Main: Introduction to the Design and Analysis of Algorithms. A. V. Levitin. Addison Wesley.

Reference: Introduction to Algorithms. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. The MIT Press

Module Information (2)

Teaching, Assessments and Help

36 lectures, 11 tutorials

2 assessments (20%), 1 written exam (80%)

Office hours, email

Tutorials/Labs

Location :

Lecture Rooms (theoretical) or

Lab (practical)

Week 2: Theoretical – Lecture Rooms

Module Information (3)

- Each assessment has two components
  - Tutorial participation (25%)
  - Class Test (75%)
- Assessment 1
  - Tutorials 1 – 6 (Weeks 2-7)
  - Class Test 1: Week 8, **Thu 23rd Mar**
- Assessment 2
  - Tutorials 7 – 11 (Weeks 8-12)
  - Class Test 2: Week 12, **Thu 11th May**

Aims

- To give an overview of the study of algorithms in terms of their ***efficiency***. **What do we mean by good?**
- To introduce the standard algorithmic ***design paradigms*** employed in the development of efficient algorithmic solutions. **How to achieve?**
- To describe the ***analysis*** of algorithms in terms of the use of formal models of Time and Space. **Can we prove?**

Ready to start …

Learning outcomes

⇨ Able to tell what an algorithm is & have some understanding why we study algorithms

- Able to use pseudo code to describe algorithm

What is an algorithm?

A sequence of *precise and concise* instructions that guide you (or a computer) to solve a *specific* problem

Input

Algorithm

Output

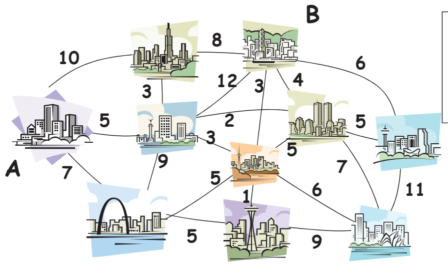
Daily life examples: cooking recipe, furniture assembly manual
(What are input / output in each case?)

Why do we study algorithms?

The obvious solution to a problem may not be efficient

Given a map of n cities & traveling cost between them.

What is the cheapest way to go from city A to city B?



Simple solution
- Compute the cost of *each path* from A to B
- Choose the cheapest one

Shortest path to go from A to B

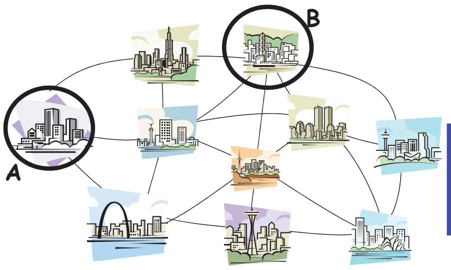
The obvious solution to a problem may not be efficient

How many paths between A & B? involving **1** intermediate city?

3?

5?

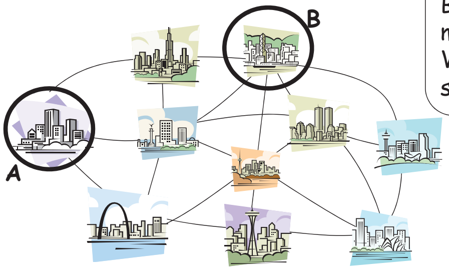
TOO MANY!!



For large n, it's impossible to check all paths!
We need more sophisticated solutions

Shortest path to go from A to B

There is an algorithm, called **Dijkstra's algorithm**, that can compute this shortest path ***efficiently***.



Lesson to learn:
Brute force algorithm may run slowly.
We need more sophisticated algorithms.

How to represent algorithms ...

✓ Able to tell what an algorithm is and have some understanding why we study algorithms

⇨ Able to use pseudo code to describe algorithm

Algorithm vs Program

An algorithm is a sequence of precise and concise instructions that guide a person/computer to solve a specific problem

Algorithms are free from grammatical rules
- Content is more important than form
- Acceptable as long as it tells people how to perform a task

Programs must follow some syntax rules
- Form is important
- Even if the idea is correct, it is still not acceptable if there is syntax error

Compute the n-th power

Input: a number **x** & a non-negative integer **n**

Output: the n-th power of x

Algorithm:

1. Set a temporary variable p to 1.
2. Repeat the multiplication **p = p * x** for **n** times.
3. Output the result p.

Pseudo Code

```
pseudo code:
p = 1
for i = 1 to n do
  p = p * x
output p
```

```
C:
p = 1;
for (i=1; i<=n; i++)
  p = p * x;
printf("%d\n", p);
```

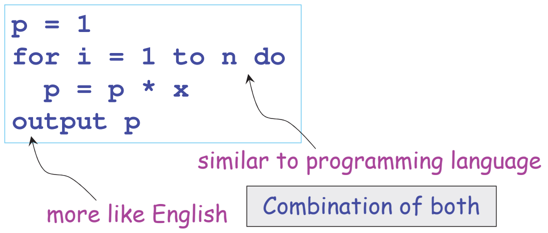
```
C++:
p = 1;
for (i=1; i<=n; i++)
  p = p * x;
cout << p << endl;
```

```
Pascal:
p := 1;
for i := 1 to n do
  p := p * x;
writeln(p);
```

```
Java:
p = 1;
for (i=1; i<=n; i++)
  p = p * x;
System.out.println(p);
```

Pseudo Code

Another way to describe algorithm is by pseudo code



```
p = 1
for i = 1 to n do
  p = p * x
output p
```

similar to programming language

more like English

Combination of both

Pseudo Code: conditional

Conditional statement

```
if condition then
  statement

if condition then
  statement
else
  statement
```

```
if a < 0 then
  a = -a
b = a
output b
```

```
if a > 0 then
  b = a
else
  b = -a
output b
```

What is computed?

Pseudo Code: iterative (loop)

Iterative statement

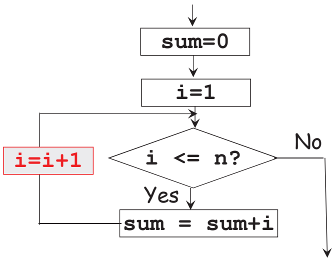
```
for var = start_value to end_value do
  statement

while condition do
  statement
```

var **automatically increased** by 1 after each iteration

condition to **CONTINUE** the loop

for loop

```
for var = start_value to end_value do
  statement
```



Sum of 1st n nos.:

```
input: n
sum = 0
for i = 1 to n do
begin
  sum = sum + i
end
output sum
```

the **loop** is executed **n** times

for loop

```
for var = start_value to end_value do
  statement
```

suppose n=4

iteration	i	sum
start		0
1	1	1
2	2	3
3	3	6
4	4	10
end	5	

trace table

Sum of 1st n nos.:

```
input: n
sum = 0
for i = 1 to n do
begin
  sum = sum + i
end
output sum
```

the **loop** is executed **n** times

while loop

```
while condition do
  statement
```

condition to **CONTINUE** the loop

Sum of 1st n numbers:

```
input: n
sum = 0
i = 1
while i <= n do
begin
  sum = sum + i
  i = i + 1
end
output sum
```

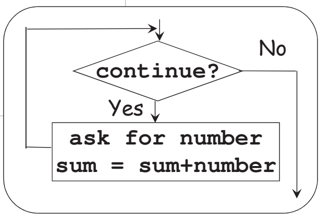
- Do the same as for-loop in previous slides
- It requires to increment i explicitly

while loop – example 2

Sum of all input numbers:

```
sum = 0
while (user wants to continue) do
begin
  ask for a number
  sum = sum + number
end
output sum
```

execute **undetermined** number of times



More Example 1

```
input: x, y
r = x
q = 0
while r >= y do
begin
  r = r - y
  q = q + 1
end
output r and q
```

What is computed?

suppose **x=14, y=4**

(@ end of) iteration	r	q
	14	0
1	10	1
2	6	2
3	2	3

suppose **x=14, y=5**

(@ end of) iteration	r	q
1	9	1
2	4	2

suppose **x=14, y=7**

(@ end of) iteration	r	q
1	7	1
2	0	2

More Example 1 - Note

```
input: x, y
r = x
q = 0
while r >= y do
begin
  r = r - y
  q = q + 1
end
output r and q
```

if condition is **r >= ??** then in the loop we need to **decrease r**

if condition is **r <= ??** then in the loop we need to **increase r**

More Example 2

```
input: x, y
i = 1
while i <= y do
begin
  if x%i==0 && y%i==0
  then output i
  i = i+1
end
```

a%b
remainder of
a divided b

What values are output?

suppose x=12, y=4

(@ end of) iteration	output (this iteration)	i
		1
1	1	2
2	2	3
3		4
4	4	5

suppose x=15, y=6

		1
1	1	2
2		3
3	3	4
4		5
5		6
6		7

More Example 3

What value is output?

```
input: x, y
i = y
found = false
while i >= 1 && !found do
begin
  if x%i==0 && y%i==0
  then found = true
  else i = i-1
end
output i
```

Questions:
- what value of found makes the loop stop?
- when does found change to such value?

Developing pseudo code

Write a **while**-loop to

assuming x and y are both integers

Find the **product** of all integers in interval [x, y]

- Examples

x	y	calculation	product
2	5	2 x 3 x 4 x 5	120
10	12	10 x 11 x 12	1320
-4	-2	-4 x -3 x -2	-24
-6	-5	-6 x -5	30
-2	1	-2 x -1 x 0 x 1	0

Developing pseudo code

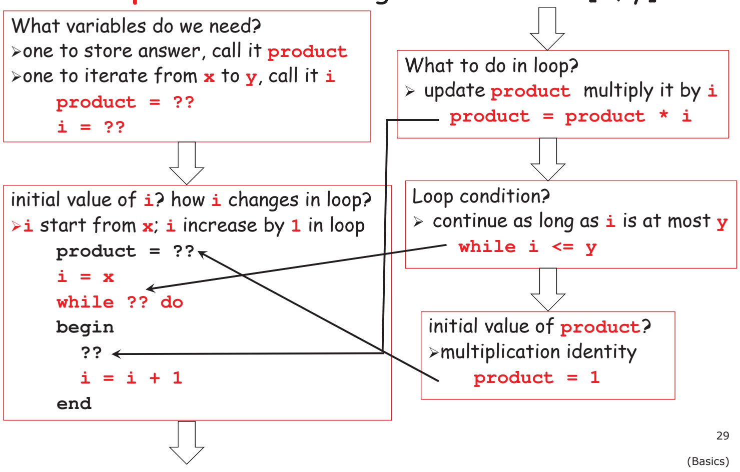
Write a **while**-loop to

assuming x and y are both integers

Find the **product** of all integers in interval [x, y]

```
product = ??
i = ??
while ?? do
begin
  ??
  i = ??
end
output ??
```

Find the **product** of all integers in interval [x, y]



What variables do we need?
- one to store answer, call it product
- one to iterate from x to y, call it i

```
product = ??
i = ??
```

initial value of i? how i changes in loop?
- i start from x; i increase by 1 in loop

```
product = ??
i = x
while ?? do
begin
  ??
  i = i + 1
end
```

What to do in loop?
- update product multiply it by i

```
product = product * i
```

Loop condition?
- continue as long as i is at most y

```
while i <= y
```

initial value of product?
- multiplication identity

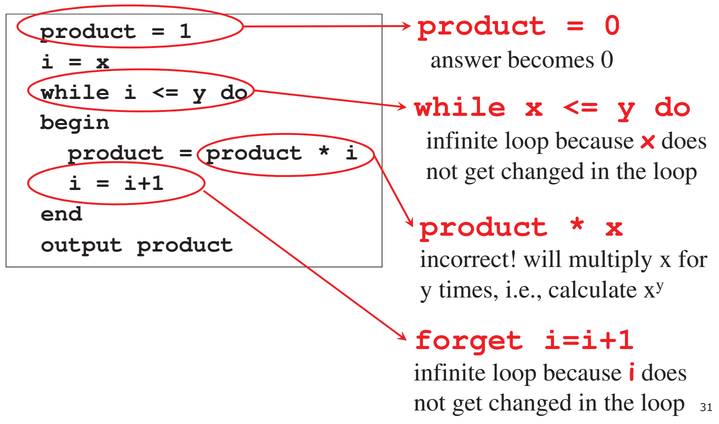
```
product = 1
```

Developing pseudo code

Find the **product** of all integers in interval [x, y]

```
product = 1
i = x
while i <= y do
begin
  product = product * i
  i = i+1
end
output product
```

Common Mistakes



```
product = 1
i = x
while i <= y do
begin
  product = product * i
  i = i+1
end
output product
```

product = 0
answer becomes 0

while x <= y do
infinite loop because **x** does not get changed in the loop

product * x
incorrect! will multiply x for y times, i.e., calculate x^y

forget i=i+1
infinite loop because **i** does not get changed in the loop

Pseudo Code: Exercise

Write a while-loop for this:

Given two positive integers x and y, list **all factors** of x which are **not** factors of y

- Examples

x	y	factors of x	output
6	3	1, 2, 3, 6	2, 6
30	9	1, 2, 3, 5, 6, 10, 15, 30	2, 5, 6, 10, 15, 30
3	6	1, 3	-

Pseudo Code: Exercise

Write a while-loop for this:

Given two positive integers x and y, list **all factors** of x which are **not** factors of y

```
i = ??
while ?? do
begin
  if ?? then
    output ??
  i = ??
end
```

Pseudo Code: Exercise

Write a while-loop for this:

Given two positive integers x and y, list **all factors** of x which are **not** factors of y

Two subproblems:

- find **all factors** of x
- if it is not a factor of y, output it

Find all factors of x

factor of x must be between 1 and x

- variable i to iterate from 1 to x

```
i = 1
while i <= x do
begin
  …
  i = i + 1
end
```

if i is divisible by x, then it is a factor of x

- remainder of i divided by x is 0

```
if x%i==0 then
  output i
```

Therefore:

```
i = 1
while i <= x do
begin
  if x%i==0 then
    output i
  i = i + 1
end
```

1. All factors of x

```
i = 1
while i <= x do
begin
  if x%i==0 then
    output i
  i = i + 1
end
```

2. Factors of x but not factor of y

- remainder of i divided by x is 0
- remainder of i divided by y is **not** 0

```
if x%i==0 && y%i!=0 then
  output i
```

3. Finally,

```
i = 1
while i <= x do
begin
  if x%i==0 && y%i!=0 then
    output i
  i = i + 1
end
```