COMP108
Algorithmic Foundations

Basics

Prudence Wong

http://www.csc.liv.ac.uk/~pwong/teaching/comp108/201314

Crossing Bridge @ Night

1 min

2 min

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

Target: all cross
the bridge

Can you do it
in 17 mins?

5 min

10 min

Module Information

Dr Prudence Wong

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

office hours: Mon 3-5pm

Demonstrators

Mr David Hamilton, Miss Alison Liu, Mr Jude-Thaddeus Ojiaku

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

- 33 lectures, 11 tutorials
- 2 assessments (20%), 1 written exam (80%)
- Office hours, email

Tutorials/Labs

- Location :
  - Lecture/Seminar Rooms (theoretical) or
  - Lab 1 (practical)
- Week 2: Theoretical – Lecture/Seminar Rooms

Module Information (3)

- Each assessment has two components
  - Tutorial participation (25%)
  - Class Test (75%)
- Assessment 1
  - Tutorials 1 – 5 (Weeks 2-6)
  - Class Test 1: Week 6, **Thu 6th Mar**
- Assessment 2
  - Tutorials 6 – 11 (Weeks 7-12)
  - Class Test 2: Week 12, **Thu 8th 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

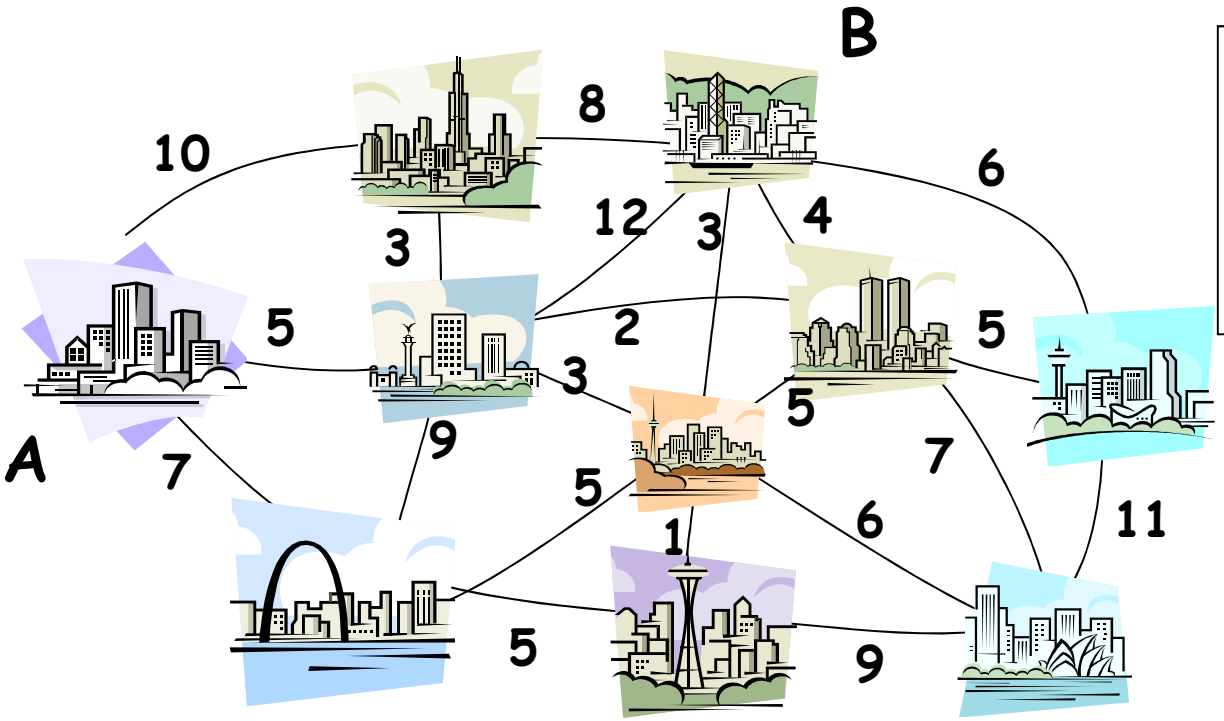
Daily life examples: cooking recipe, furniture assembly manual
(What are input / output in each case?)

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

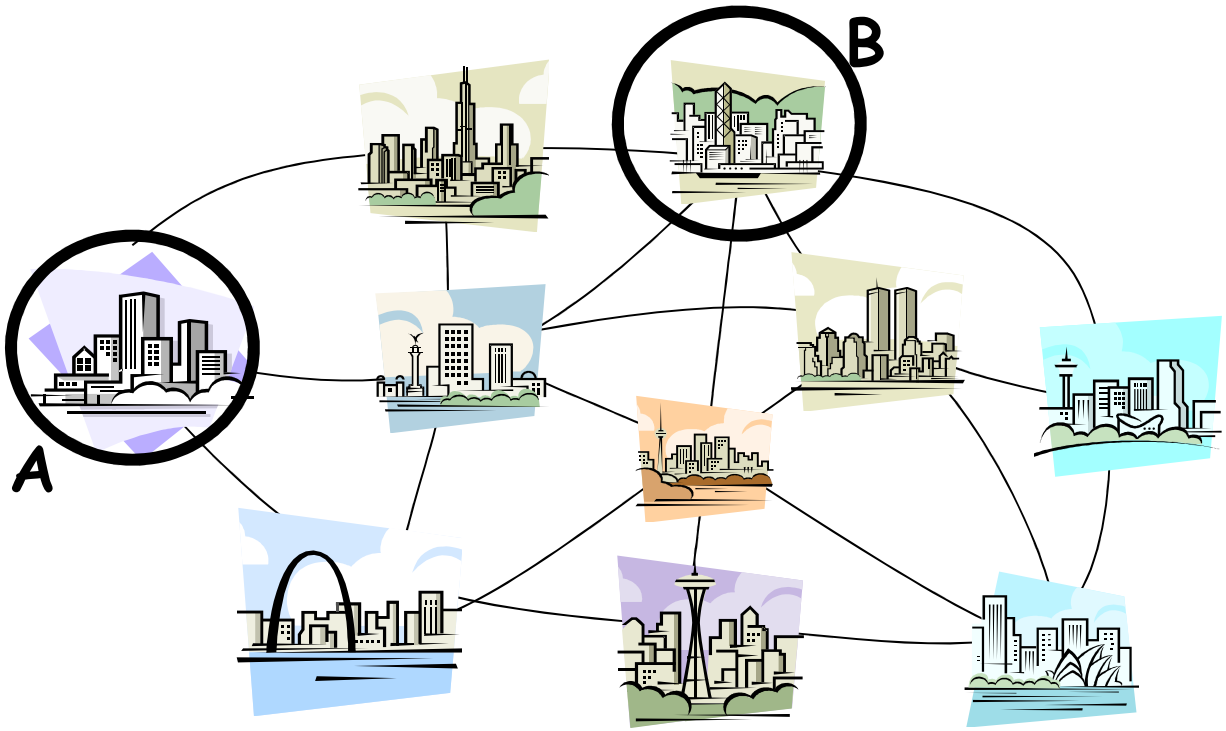
Shortest path to go from A to B

The obvious solution to a problem may not be efficient

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

3?

5?

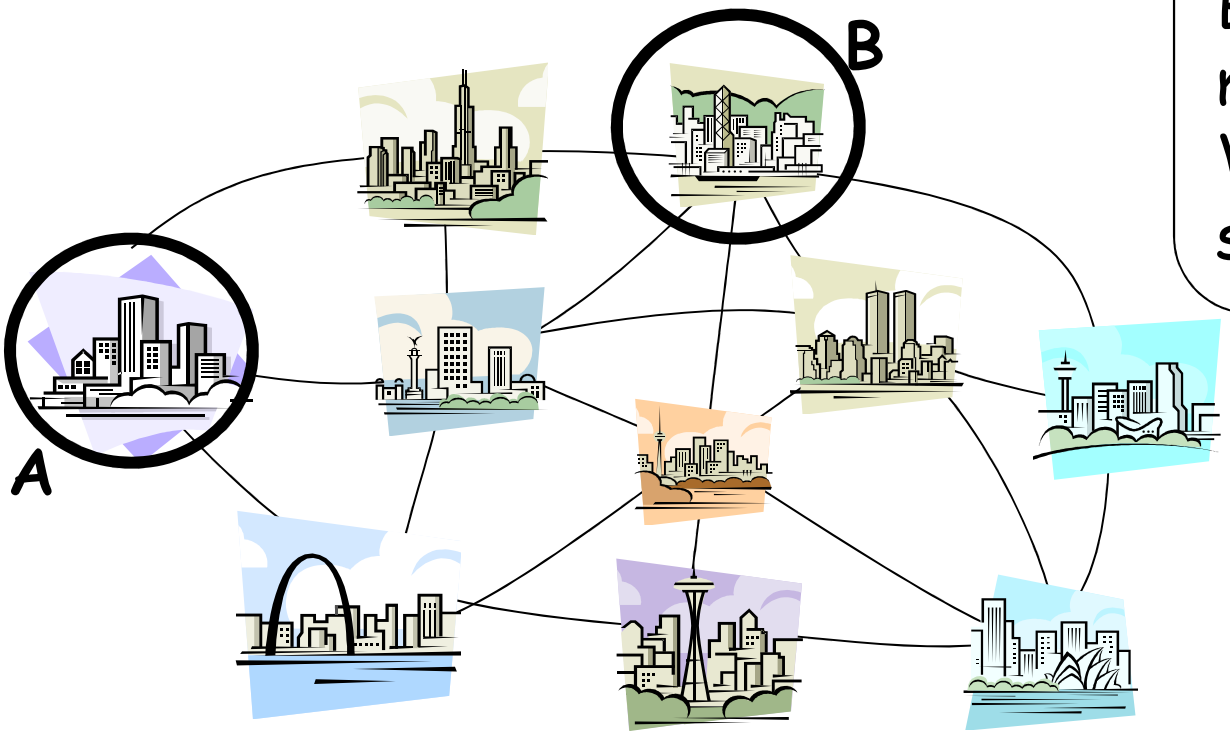


TOO MANY!!

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);
```

Pascal:

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

C++:

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

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
```

suppose **n=4, x=3**

iteration	i	p
start		1
1	1	3
2	2	9
3	3	27
4	4	81
end	5	

trace table

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

repeat
   statement
until condition
```

var automatically increased by 1 after each iteration

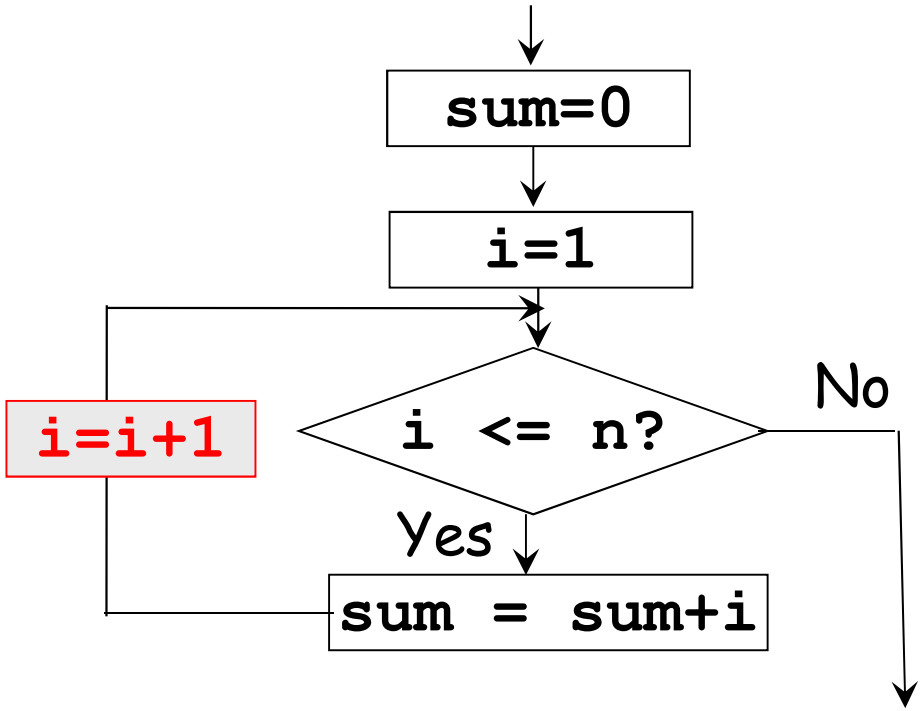
condition to CONTINUE the loop

condition to STOP the loop

condition for while loop is NEGATION of condition for repeat-until 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	

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
```

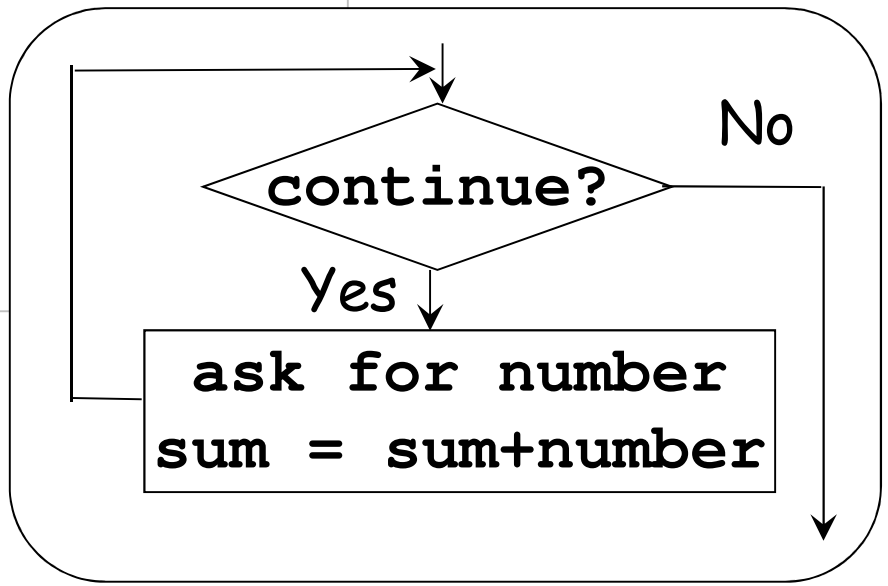
- Do the same as for-loop in previous slides
- It requires to increment i explicitly

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
```

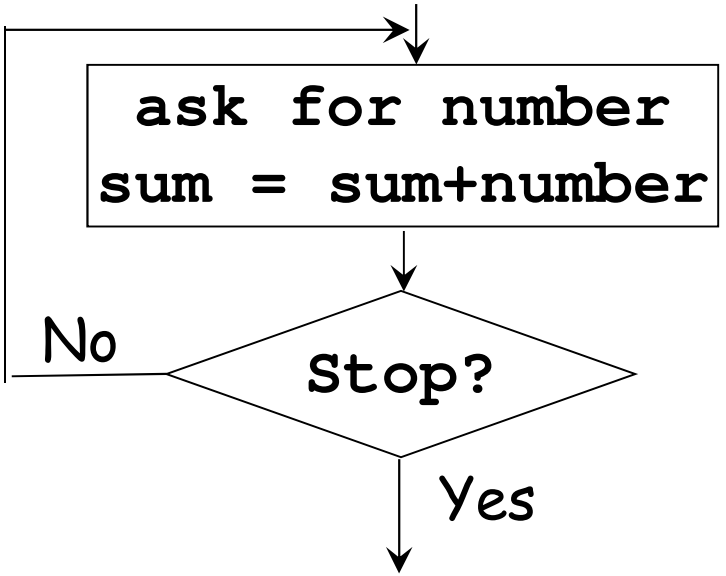
execute **undetermined** number of times



repeat-until

```
repeat
   statement
until condition
```

condition to STOP the loop



Sum of all input numbers:

```
sum = 0
repeat
   ask for a number
   sum = sum + number
until (user wants to stop)
output sum
```

- also execute **undetermined** number of times
- How it differs from while-loop?

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**

More Example 2

```
input: x, y
if x < y then
    swap x & y
i = y
while i >= 1 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
		4
1	4	3
2		2
3	2	1
4	1	0

suppose x=15, y=6

		6
1		5
2		4
3		3
4	3	2
5		1
6	1	0

More Example 3

```
input: x, y
if x < y then
    swap 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
```

What value is output?

Questions:

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

Pseudo Code: Exercise

Write a **while**-loop to

assuming x and y are both integers

1. Find the **product** of all integers in interval [x, y]
   - e.g., if x is **2** & y is **5**, then output is **2*3*4*5 = 120**

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

Pseudo Code: Exercise 2

Write a while-loop for this:

2. Given two positive integers x and y, list **all factors** of x which are **not** factors of y
   - if x is **30** & y is **9**, output is **2, 5, 6, 10, 15, 30** (not 1 or 3)

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

Challenges ...

Convert while-loops to for-loops
& repeat-loop

Convert to **for/repeat** loops

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

assuming x and y are both integers

Convert to **for/repeat** loops (2)

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

Searching ...

Searching

- Input: n numbers $a_1, a_2, \ldots, a_n$; and a number X
- Output: determine if X is in the sequence or not
- Algorithm (Sequential search):
  1. From i=1, compare X with a_i one by one as long as i <= n.
  2. Stop and report "Found!" when $X = a_i$.
  3. Repeat and report "Not Found!" when i > n.

Sequential Search

➢	12 7	34	2	9	7	5 ← six numbers ← number X
➢	12	34 7	2	9	7	5
➢	12	34	2 7	9	7	5
➢	12	34	2	9 7	7	5
➢	12	34	2	9	7 7	5 found!

Sequential Search (2)

- 12 34 2 9 7 5
  10
- 12 34 2 9 7 5
  10
- 12 34 2 9 7 5
  10
- 12 34 2 9 7 5
  10
- 12 34 2 9 7 5
  10
- 12 34 2 9 7 5
  10

not found!

Sequential Search – Pseudo Code

```
i = 1
while i <= n do
begin
  if X == a[i] then
     report "Found!" and stop
  else
     i = i+1
end
report "Not Found!"
```

Challenge: Modify it to include stopping conditions in the while loop

Number of comparisons?

```
i = 1
while i <= n do
begin
  if X == a[i] then
    report "Found!" & stop
  else
    i = i+1
end
report "Not Found!"
```

How many comparisons this algorithm requires?

Best case: X is 1st no.
⇒ ??? comparison

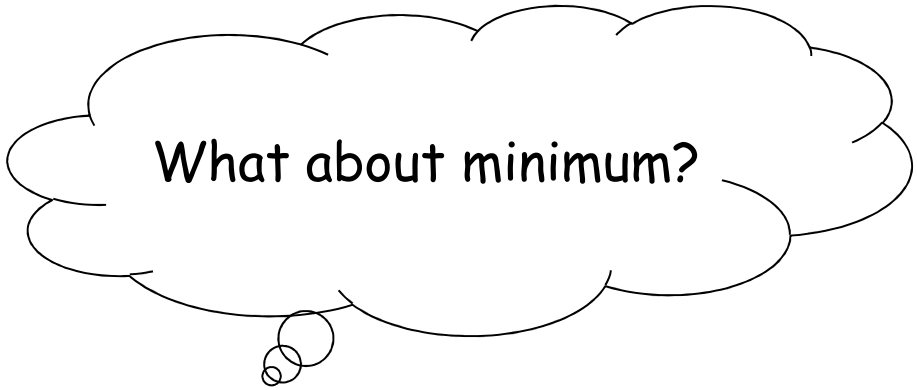
Worst case: X is last
OR X is not found
⇒ ??? comparisons

Finding maximum / minimum...

2^{nd} max / min...

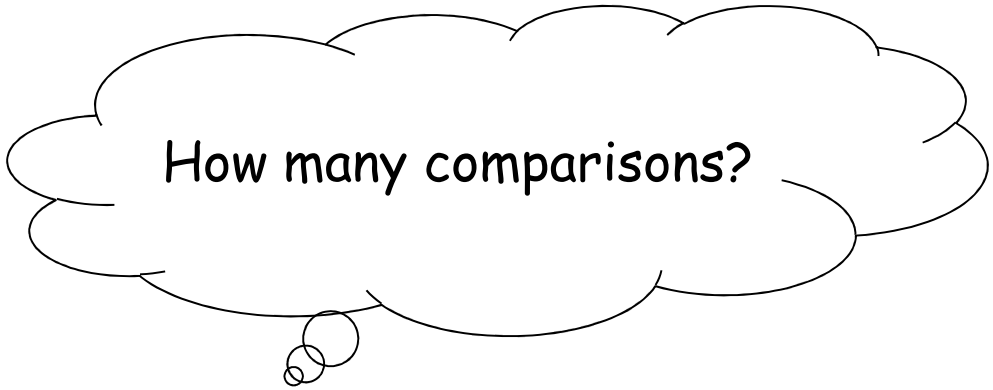
Finding max from n +ve numbers

```
input: a[1], a[2], ..., a[n]
M = 0
i = 0
while (i < n) do
begin
   i = i + 1
   M = max(M, a[i])
end
output M
```



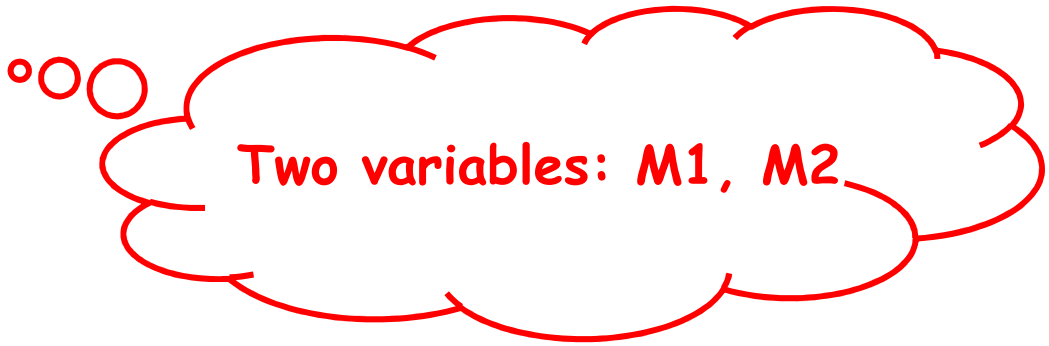
Finding **min** from n +ve numbers

```
input: a[1], a[2], ..., a[n]
M = ???
i = ???
while (i < n) do
begin
   i = i + 1
   M = min(M, a[i])
end
output M
```



Finding 1st and 2nd min

```
input: a[1], a[2], ..., a[n]
M1 = ???
M2 = ???
i = ???
while (i < n) do
begin
   i = i + 1
   if (???) then
      ???
   else if (???) then
           ???
end
output M1, M2
```



Finding **location** of minimum

```
input: a[1], a[2], ..., a[n]
loc = 1 // location of the min number
i = 1
while (i < n) do
begin
   i = i + 1
   if (a[i] < a[loc]) then
      loc = i
end
output a[loc]
```

Example

a[]={50,30,40,20,10}

(@ end of) Iteration	loc	a[loc]	i
	1	50	1
1	2	30	2
2	2	30	3
3	4	20	4
4	5	10	5

Finding min using for-loop

- Rewrite the above while-loop into a for-loop