INT102
Algorithmic Foundations And Problem Solving

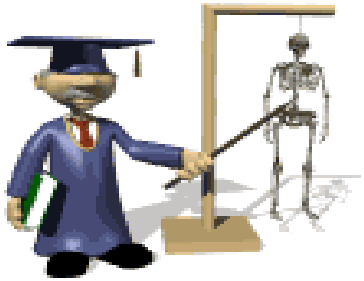
Lecture 1 Introduction

Dr Yushi Li
Department of Intelligent Science

西交利物浦大學
Xi'an Jiaotong-Liverpool University

Module information ...

Teaching Team



Module leader: Dr Jia Wang

Office Room SD537

Email: jia.wang02@xjtlu.edu.cn

Office hours: 10:00-12:00, Monday

Dr. Yushi Li

Office Room SD561

Email: yushi.li@xjtlu.edu.cn

Office hours: 10:00-12:00, Monday

Dr. Pengfei Fan

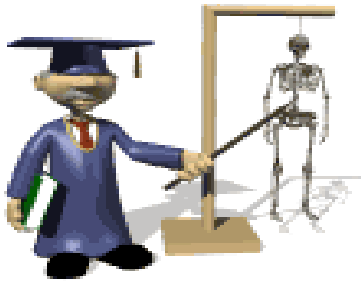
Office Room SD559

Email: pengfei.fan@xjtlu.edu.cn

Office hours: 15:00-17:00 Wednesday

Teaching & Learning: Structure

Tutorials



Lectures

Help channels

Assignments /Class Tests

Examination

Teaching & Learning: Teaching

Lectures

Group 1
Venue: SIP-SC176
Time: 9:00 -11:00 Tuesday
Group 2
Venue: SIP-SC176
Time: 14:00 -16:00 Tuesday

Tutorials:

Group 1
Venue: SIP-SC176
Time: 11:00 -13:00 Friday
Group 2
Venue: SIP-SC176
Time: 14:00 -16:00 Friday

Teaching & Learning: Assessment

- **2 assessments (20% of the final mark)**
  Assessment 1: week 6 - week 7 (10%)
  Assessment 2: week 12 – week 13 (10%)
- **Final Examination (80% of the final mark)**
  A written examination
  Date: TBA.

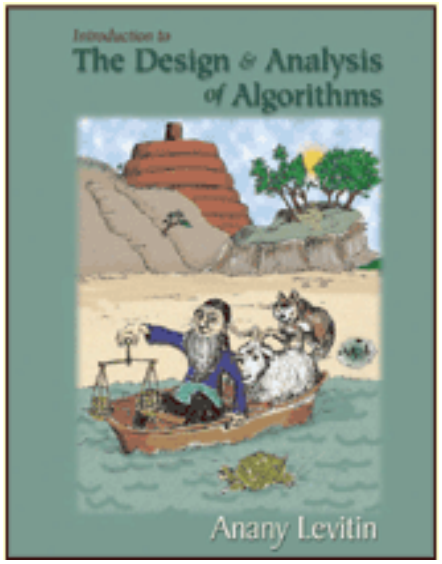
More about Assessment

- plagiarism is the practice of presenting the thoughts or writings of ***another or others*** as original
  - consequence? Please refer to the student handbook for answer.
- What happen if you are ill when there is a deadline for assignment?
  - Get **medical proof** and ask for extension
  - This is not an excuse to commit plagiarism

Contents (Tentative)

methodology / problems	Asymptotic idea	Brute force	Divide & Conquer	Dynamic Programming	Greedy	Space/Time	Branch & Bound	Complexity Theory
Efficiency	Big-O							
Sorting		Selection / Bubble/insertion	Merge-sort			Count sorting		
Searching			Binary-searching					
String marching						Horspool algorithm		
Graph		DFS/BFS		Bellman-ford/Warshall Assembly-line	MST Dikstra's For Shortest path		Traveling salesman, Job assignment	
Complexity								P/NP

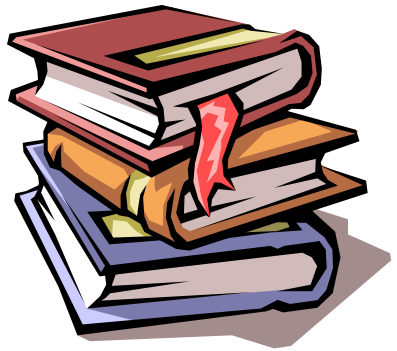
Text Book and Reference

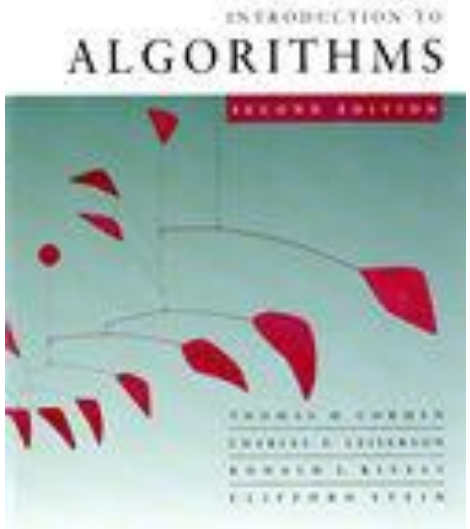


Textbook

Introduction to the Design and Analysis of Algorithms

Anany V. Levitin
Villanova University
Addison Wesley





Additional Reading

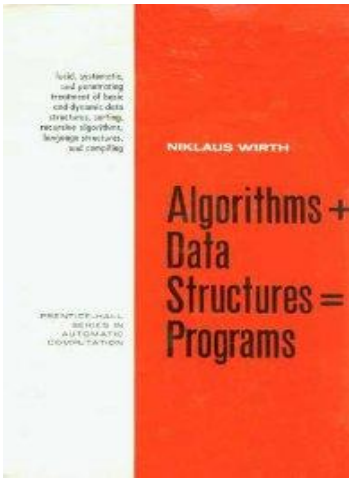
Introduction to Algorithms

Thomas H. Cormen,
Charles E. Leiserson,
Ronald L. Rivest
The MIT Press

Questions?

Why INT102?

- Algorithm design is a foundation for efficient and effective programs



Algorithms + Data Structures = Programs

- More from Donald Knuth*:

" A person well-trained in computer science knows how to deal with algorithms: how to **construct them**, **understand them, manipulate them, analyze them**. ...

(This knowledge) is a mental tool that will be a definite aid to the understanding of other subjects, whether they be chemistry, linguistics, or music, etc. …"

*Donald Knuth is the author of the famous book "The Art of Computer Programming"

Aims

What do we mean by good?

- To give an overview of the study of algorithms in terms of their ***efficiency***.

How to achieve?

- To introduce the standard algorithmic ***design patterns*** employed in the development of efficient algorithmic solutions.

Can we prove?

- To describe the ***analysis*** of algorithms in terms of the use of formal models of Time and Space.

Can all problems be solved efficiently?

- To give a brief introduction to the subject of ***computational complexity theory*** and its use in classifying computational problems.

Ready to start ...

Learning outcomes

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

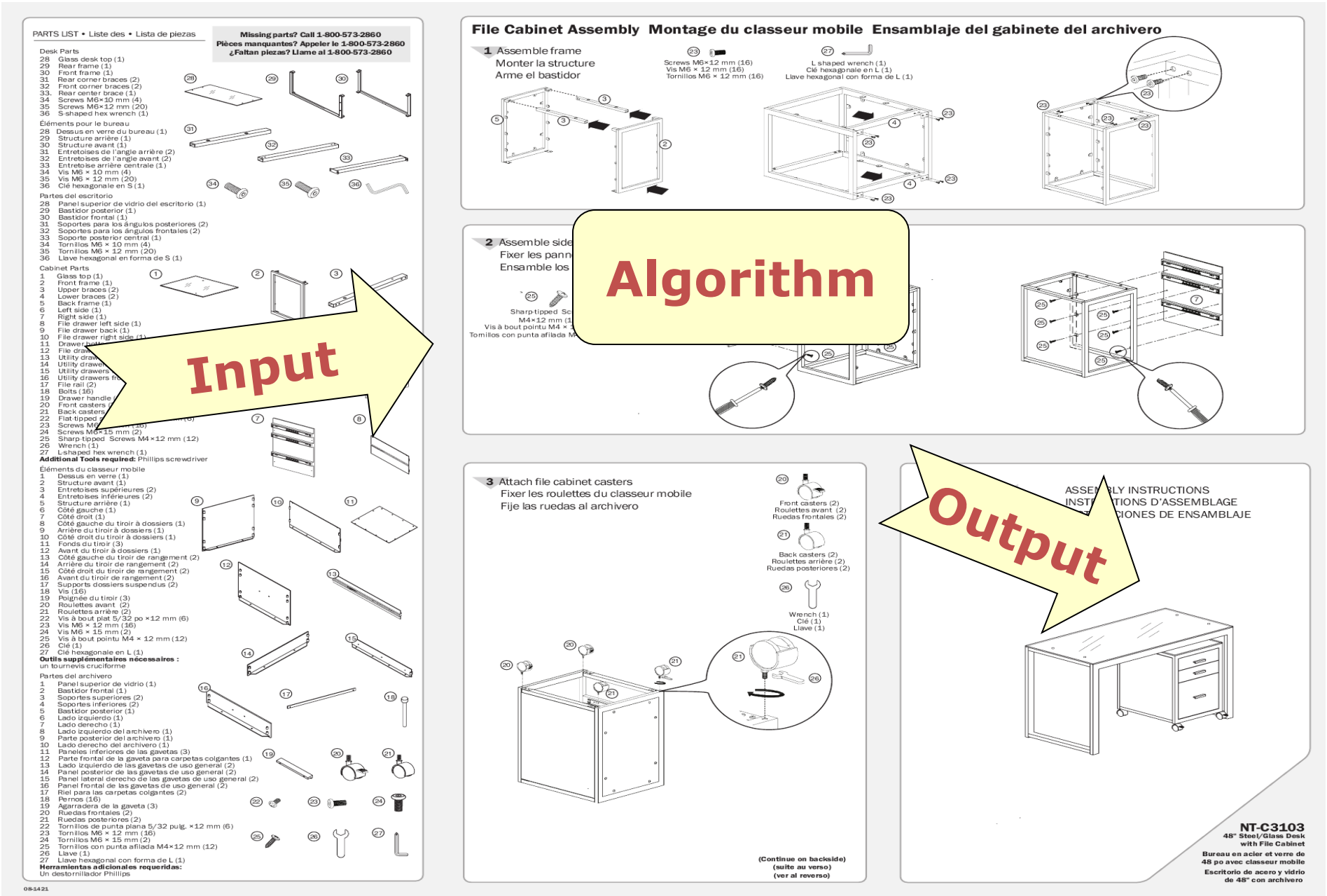
- Able to use pseudo code to describe algorithm

What is an algorithm?

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

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

Desk Assembly Manual



Input

Algorithm

Output

Why do we study algorithms?

- Example: given a sequence of numbers, say,

  1, 3, 15, 90, 100, 101, 203, 305

  and a number X check if X is in the sequence.
  - How do you check it?
  - If X=1, how many comparisons do you need?
  - If X=100, how many comparisons do you need?
  - If X=305, how many comparisons do you need?
  - If X=102, how many comparisons do you need?

A Straightforward Solution

Compare X with number in the sequence one by one.
If X equals to a number, then answer "Yes", stop.
Otherwise "No."

- Example: given a sequence of numbers, say,

  1, 3, 15, 90, 100, 101, 203, 305

  and a number X check if X is in the sequence.
  - How do you check it?
  - If X=1, how many comparisons do you need?
  - If X=100, how many comparisons do you need?
  - If X=305, how many comparisons do you need?
  - If X=102, how many comparisons do you need?

A Trick Solution

Observation: the sequence is ordered (sorted).

1. If the sequence is empty, then answer "No", stop.
2. Compare X with number in the middle of the sequence.
3. If X equals to the number, then answer "Yes", stop.
4. If X is greater than the number, go to step 1 with the subsequence on the right of the number.
5. If X is less than the number, go to step 1 with the subsequence on the left of the number.

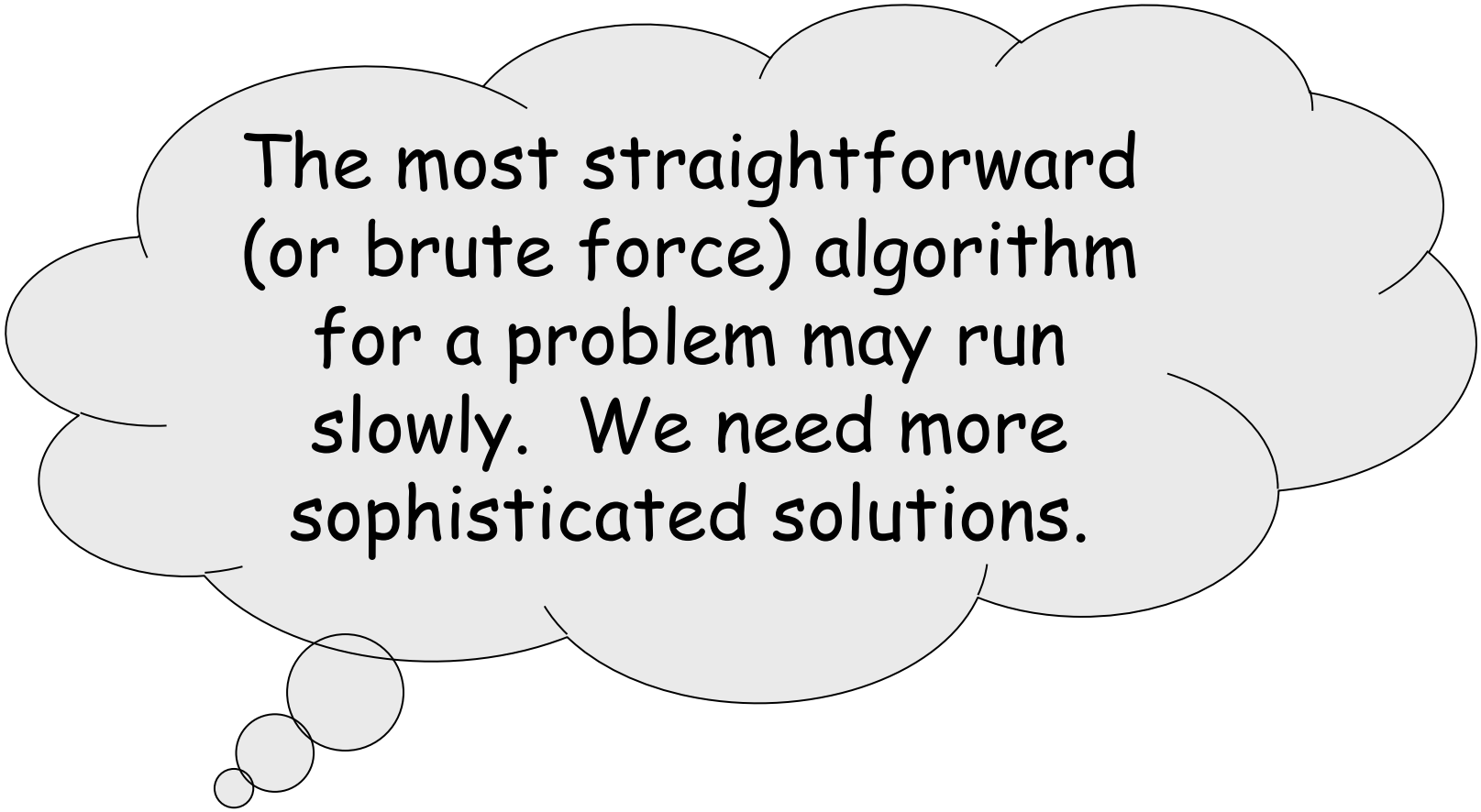
A Trick Solution

- Example: given a sequence of numbers, say,

  1, 3, 15, 90, 100, 101, 203, 305

  and a number X check if X is in the sequence.
  - How do you check it?
  - If X=1, how many comparisons do you need?
  - If X=100, how many comparisons do you need?
  - If X=305, how many comparisons do you need?
  - If X=102, how many comparisons do you need?

Lesson to learn



Some Well-known Computational Problems

- Sorting
- Searching
- Graph and Combinatorial Problems
  - Shortest paths in a graph
  - Minimum spanning tree
  - Traveling salesman problem
  - Knapsack problem
  - ……
- Problem in Number Theory
  - Primality testing
  - ……

How to represent algorithms ...

Algorithm vs Program

Still remember? An algorithm is a sequence of precise and concise instructions that guide a 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

More Generally: Program = Data Structure + Algorithm

Computing 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

Another way to describe algorithm is by pseudo code

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

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

suppose n=4, x=3

trace table

similar to programming language

more like English

Combination of both

Pseudo Code: statement

Assignment statement

```
variable = expression
```

e.g., assign the expression 3 * 4 to the variable **result**

```
result = 3 * 4
```

compound statements

```
begin
   statement1
   statement2
end
```

Pseudo Code: conditional

Conditional statement

```
if condition then
   statement

if condition then
   statement
else
   statement
```

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

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

What do these two algorithms compute?

Pseudo Code: iterative (loop)

Iterative statement

```
for var = start_value to end_value do
   statement

while condition do
   statement

repeat
   statement
until condition
```

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
```

1. var is first assigned the value start_value.
2. If var ≤ end_value, execute the statement.
3. var is incremented by 1 and then go back to step 2.

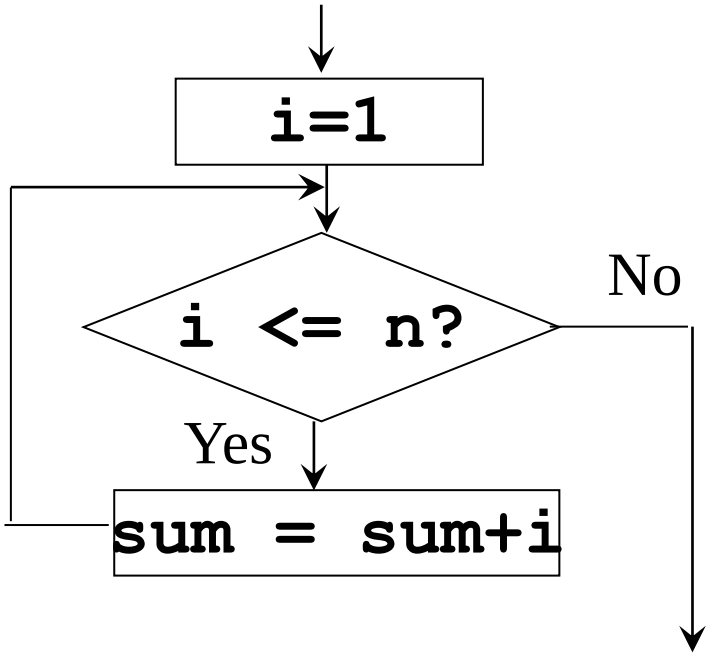
the **loop** is executed for **n** times

Computing sum of the first n numbers:

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

for loop

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



Computing sum of the first n numbers:

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

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

for loop

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

suppose n = 4

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

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

Computing sum of the first n numbers:

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

while loop

condition to CONTINUE the loop

```
while condition do
   statement
```

1. if condition is true, execute the statement
2. else stop
3. go back to step 1

Computing sum of the first n numbers:

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

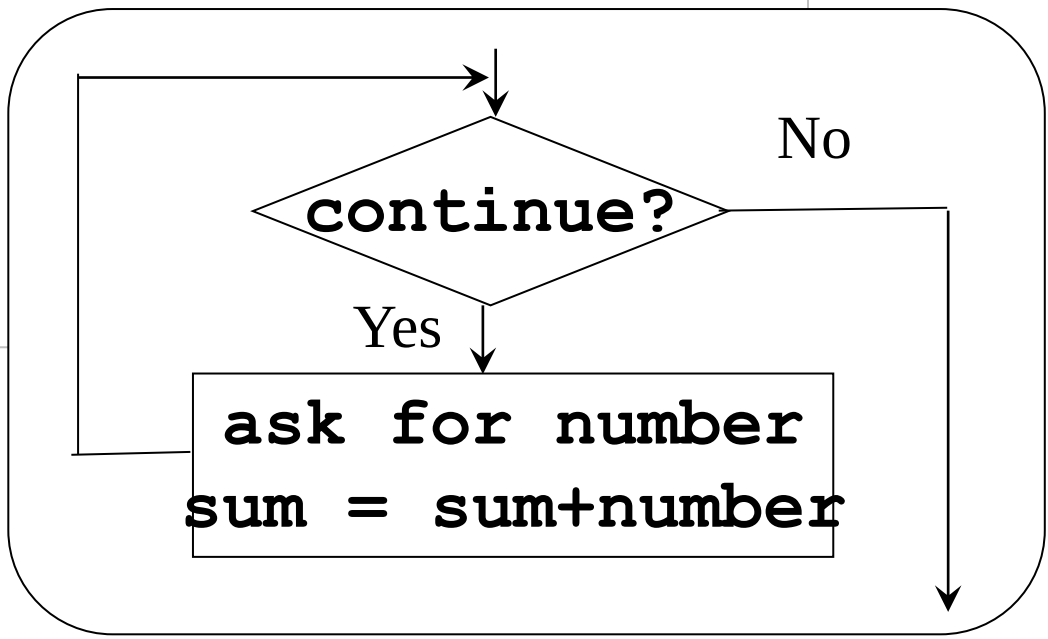
this while-loop performs the same task as the for-loop in the previous slides

while loop – example 2

this **loop** is executed for **undetermined** number of times

Computing sum of all (keyboard) input numbers:

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



repeat-until

```
repeat
   statement
until condition
```

condition to STOP the loop

1. execute the statement
2. if condition is true, stop
3. go back to step 1

Computing sum of all (keyboard) input numbers:

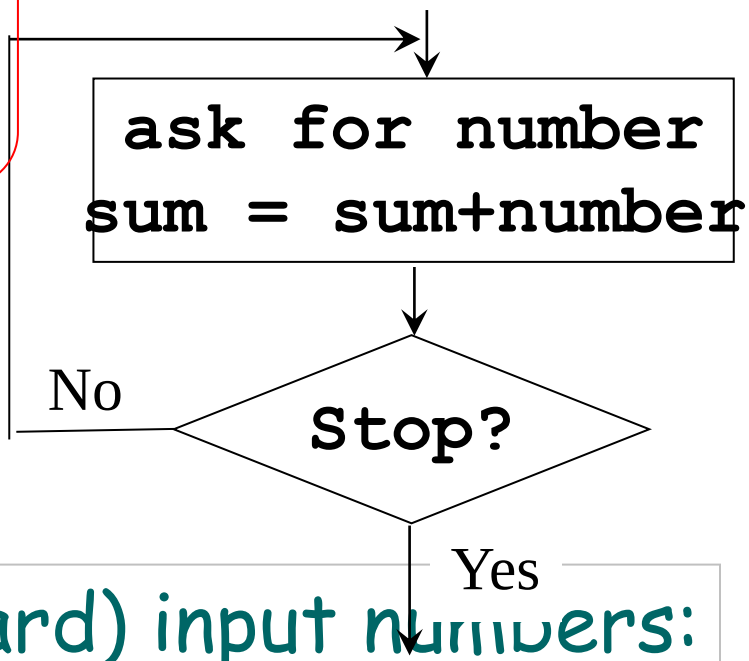
```
sum = 0
repeat
   ask for a number
   sum = sum + number
until (user wants to stop)
output sum
```

- this **loop** is also executed for **undetermined** number of times
- How it differs from while-loop?

repeat-until

```
repeat
   statement
until condition
```

condition to STOP the loop



Computing sum of all (keyboard) input numbers:

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

- this **loop** is also executed for **undetermined** number of times
- How it differs from while-loop?

Pseudo Code: exercise

Write for-loops for the followings

1. Find the product of all integers in the interval [x, y], assuming that x and y are both integers
   - e.g., if x and y are 2 and 5, then the output should be 2*3*4*5 = 120
2. List all factors of a given **positive integer** x
   - e.g., if x is 60, then the output should be 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60

Hint (1)

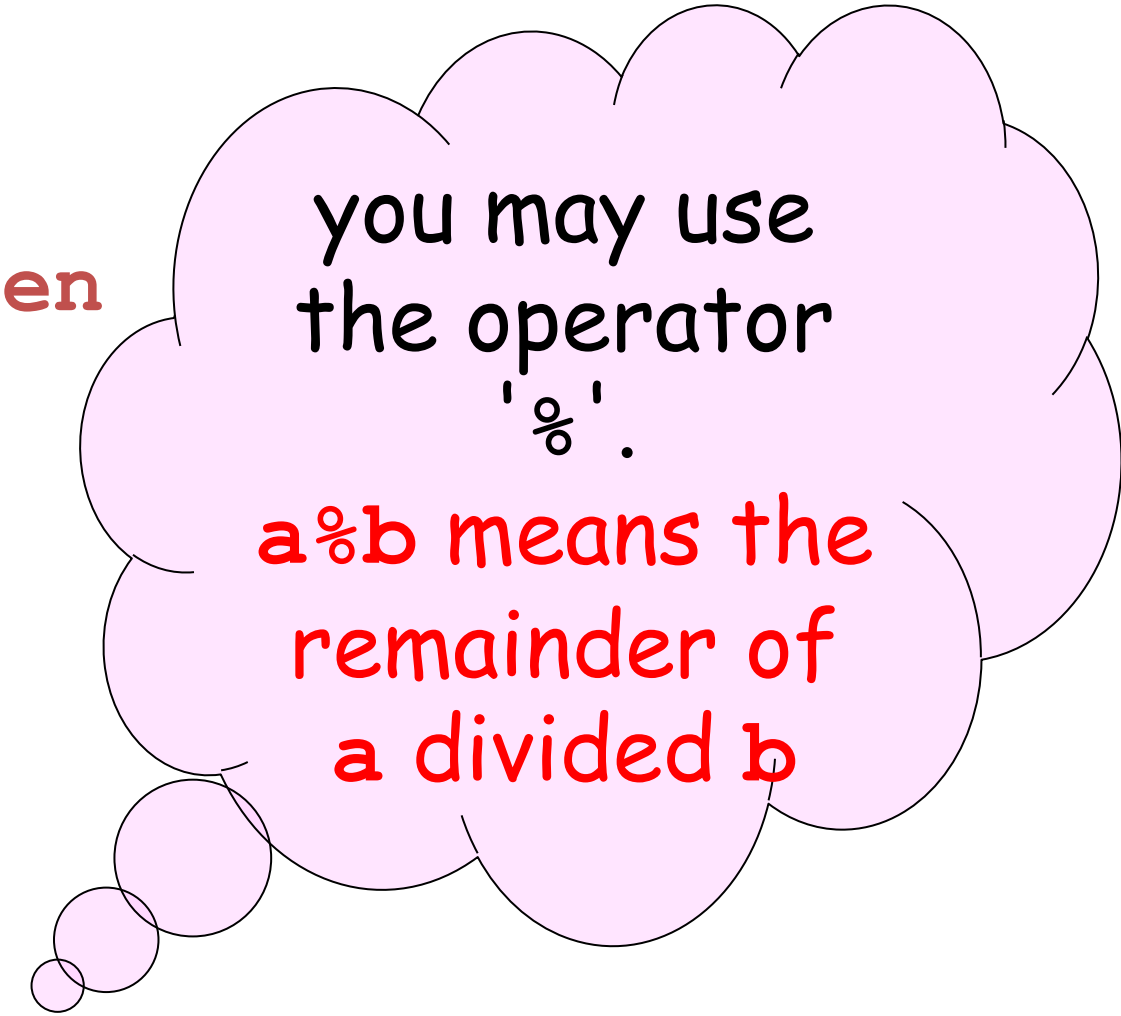
Find the product of all integers in the interval [x, y], assuming that x and y are both integers

```
product = 1
for i = x to y do
begin
   product = product * i
end
output product
```

Hint (2)

List all factors of a given **positive integer** x

```
for i = 1 to x do
begin
   if (x%i == 0) then
      output i
end
```



Convert to **while** loops

Find the product of all integers in the interval [x, y], assuming that x and y are both integers

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

Convert to **while** loops (2)

List all factors of a given **positive integer** x

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

Convert to **repeat** loops

Find the product of all integers in the interval [x, y], assuming that x and y are both integers

```
product = 1
if x <= y then begin
   i = x
   repeat
      product = product * i
      i = i+1
   until i > y
   output product
end
```

Convert to **repeat** loops (2)

List all factors of a given **positive integer** x

```
i = 1
repeat
   if x%i == 0 then
      output i
   i = i+1
until i > x
```

Learning outcomes

- ✓ Able to tell what an algorithm is and have some understanding why we study algorithms
- ✓ Able to use pseudo code to describe algorithm