

# Chapter 1

## The Role of Algorithms in Computing

1

### Algorithms

- ❖ A tool for solving a well-specified computational problem

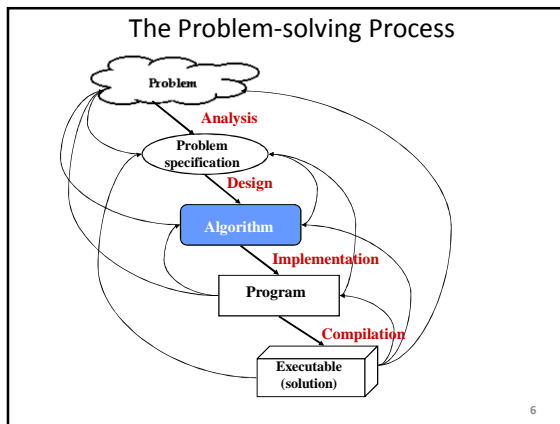
- ❖ An algorithm is said to be correct if, for every input instance, it halts with the correct output
- ❖ Algorithms must be:
  - Correct: For each input produce an appropriate output
  - Efficient: run as quickly as possible, and use as little memory as possible – more about this later

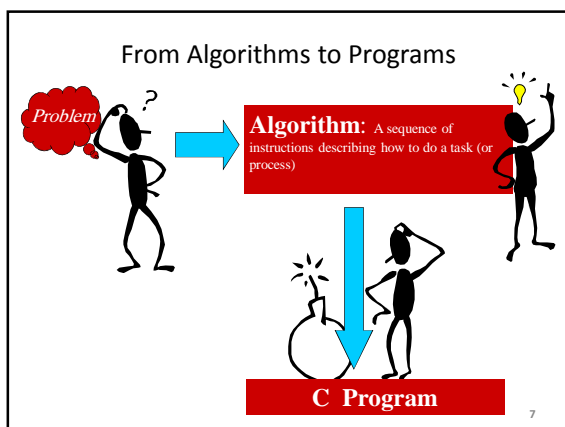
4

- ### Outlines: Introduction
- ❖ Algorithm: Definitions
  - ❖ Algorithm: Characteristics & Components
  - ❖ Algorithm: Examples
    - Finding a *smallest* number among a sequence of *n* numbers
  - ❖ Algorithm: Efficiency
- 2

- ### Problems and Algorithms
- ❖ We need to solve a computational problem
    - “Convert a weight in pounds to Kg”
  - ❖ An algorithm specifies how to solve it, e.g.:
    1. Read weight-in-pounds
    2. Calculate weight-in-Kg = weight-in-pounds \* 0.455
    3. Print weight-in-Kg
  - ❖ A computer program is a computer-executable description of an algorithm
- 5

- ### Computational problems
- ❖ A computational problem specifies an input-output relationship
    - What does the input look like?
    - What should the output be for each input?
  - ❖ Example:
    - Input: an integer number *n*
    - Output: Is the number prime?
  - ❖ Example:
    - Input: A list of names of people
    - Output: The same list sorted alphabetically
  - ❖ Example:
    - Input: A picture in digital format
    - Output: An English description of what the picture shows
- 3





### Variables

Are containers for values – places to store values

Example:

```

procedure Sum1_to_n(num)
{
    count = 1
    sum = 0
    while (count <= num)
    {
        add count to sum
        add 1 to count
    }
}
    
```

10

- ### Components of an Algorithm
- ❖ Variables and values
  - ❖ Instructions
  - ❖ Sequences
  - ❖ Procedures
  - ❖ Selections
  - ❖ Repetitions
  - ❖ Documentation
- 8

- ### Components of an Algorithm
- ❖ **Values and Variables**
  - ❖ Instruction
  - ❖ Sequence (of instructions)
  - ❖ Procedure (involving instructions)
  - ❖ Selection (between instructions)
  - ❖ Repetition (of instructions)
  - ❖ Documentation (beside instructions)
- 11

### Values

- ❖ Represent quantities, amounts or measurements
- ❖ May be numerical or alphabetical (or other things)
- ❖ Often have a unit related to their purpose
- ❖ Example:

```

procedure Sum1_to_n(num)
{
    count = 1
    sum = 0
    while (count <= num)
    {
        add count to sum
        add 1 to count
    }
}
    
```

9

### Instructions (Primitives)

- ❖ Some action that is simple...
- ❖ ...and unambiguous...
- ❖ ...that the system knows about...
- ❖ ...and should be able to actually do
- ❖ Examples:

```

procedure Sum1_to_n(num)
{
    count = 1
    sum = 0
    while (count <= num)
    {
        add count to sum
        add 1 to count
    }
}
    
```

Directions to perform specific actions on values and variables.

12

### Components of an Algorithm

- ❖ Values and Variables
- ❖ Instruction
- ❖ Sequence (of instructions)
- ❖ Procedure (involving instructions)
- ❖ Selection (between instructions)
- ❖ Repetition (of instructions)
- ❖ Documentation (beside instructions)

13

### Components of an Algorithm

- ❖ Values and Variables
- ❖ Instruction
- ❖ Sequence (of instructions)
- ❖ Procedure (involving instructions)
- ❖ Selection (between instructions)
- ❖ Repetition (of instructions)
- ❖ Documentation (beside instructions)

16

### Sequence

- ❖ A series of instructions
- ❖ ...to be carried out one after the other...
- ❖ ...without hesitation or question
- ❖ Example:
  - How to cook Gourmet Meal

14

### Procedure

- ❖ A named sequence of instructions
- ❖ So that you can
  - Refer to it collectively (by name)
  - ...instead of individually (by each instruction in the sequence)
- ❖ Example:
  - Drive\_To\_Uni

17

### Sequence -- Example

1. Open freezer door
2. Take out Gourmet Meal™
3. Close freezer door
4. Open microwave door
5. Put Gourmet Meal™ on carousel
6. Shut microwave door
7. Set microwave on high for 5 minutes
8. Start microwave
9. Wait 5 minutes
10. Open microwave door
11. Remove Gourmet Meal™
12. Close microwave door

15

### Procedure -- Example

```

procedure Drive_To_Uni
{
  1. find car keys
  2. disable car alarm
  3. open car door
  4. get in car
  5. shut car door
  6. put keys in ignition
  7. start car
  8. back car out of driveway
  9. drive to end of street
  10. turn right
  11. drive to end of street
  12. turn left
  ...etc...etc...etc
  52. find parking space
  53. pull into parking space
  54. turn off engine
  55. remove keys from ignition
  56. open car door
  57. get out
  58. shut car door
  59. lock car door
  60. enable alarm
}
    
```

18

### Procedure – Example (cont)

```

procedure Do_Wednesday
{
  Wake_up
  Have_Shower
  Eat_Breakfast
  Drive_To_Uni
  Sit_1301_Lecture
  ...etc...etc...etc...
  Drive_From_Uni
  ...etc...etc...etc...
}
    
```

```

procedure Do_Week
{
  Do_Monday
  Do_Tuesday
  Do_Wednesday
  Do_Thursday
  ...etc...etc...etc...
}
    
```

19

### Procedure – Example (cont)

```

procedure Do_Wednesday
{
  Wake_up
  Have_Shower
  Eat_Breakfast
  Drive_To_Uni
  Sit_1301_Lecture
  ...etc...etc...etc...
  Drive_From_Uni
  ...etc...etc...etc...
}
    
```

An instruction invoking a procedure is known as a **“procedure call”**

22

### Procedure – Example (cont)

```

procedure Do_Wednesday
{
  Wake_up
  Have_Shower
  Eat_Breakfast
  Drive_To_Uni
  Sit_1301_Lecture
  ...etc...etc...etc...
  Drive_From_Uni
  ...etc...etc...etc...
}
    
```

In this subject, we also use the following words to refer to a “Procedure” :

**Sub-routine**  
**Module**  
**Function**

20

### Procedure

- ❖ A procedure may have a set of parameters

```

procedure customerService ( myName ,timeOfDay )
{
  say “Good timeOfDay”
  say “My name is myName”
  say “How can I help you?”
}
    
```

```

customerService ( “Ann” ; “Morning ”)
customerService ( “Ann” ; “Afternoon ”)
customerService ( “Jeff” ; “Afternoon ”)
    
```

23

### Procedure – Example (cont)

```

procedure Do_Wednesday
{
  Wake_up
  Have_Shower
  Eat_Breakfast
  Drive_To_Uni
  Sit_1301_Lecture
  ...etc...etc...etc...
  Drive_From_Uni
  ...etc...etc...etc...
}
    
```

We use **brackets** to mark the beginning and end of a sequence.

21

### Components of an Algorithm

- ❖ Values and Variables
- ❖ Instruction
- ❖ Sequence (of instructions)
- ❖ Procedure (involving instructions)
- ❖ Selection (between instructions)
- ❖ Repetition (of instructions)
- ❖ Documentation (beside instructions)

24

### Selection

- ❖ An instruction that decides which of two possible sequences is executed
- ❖ The decision is based on a single **true/false** condition
- ❖ Examples:
  - Car repair
  - Reciprocals

25

### Selection Example -- Car Repair (cont)

```

if (motor turns)
then
{
  CheckFuel
  CheckSparkPlugs
  CheckCarburettor
}
else
{
  CheckDistributor
  CheckIgnitionCoil
}
    
```

28

Sequence if the condition is true.

### Selection Example -- Car Repair

```

if (motor turns)
then
{
  CheckFuel
  CheckSparkPlugs
  CheckCarburettor
}
else
{
  CheckDistributor
  CheckIgnitionCoil
}
    
```

26

### Selection Example -- Car Repair (cont)

```

if (motor turns)
then
{
  CheckFuel
  CheckSparkPlugs
  CheckCarburettor
}
else
{
  CheckDistributor
  CheckIgnitionCoil
}
    
```

29

Sequence if the condition is false.

### Selection Example -- Car Repair (cont)

```

if (motor turns)
then
{
  CheckFuel
  CheckSparkPlugs
  CheckCarburettor
}
else
{
  CheckDistributor
  CheckIgnitionCoil
}
    
```

27

Should be a true or false condition.

### Selection Example -- Reciprocals

**Q.** Give an algorithm for computing the reciprocal of a number.

*Examples:*

Reciprocal of 2:  $1/2$

Reciprocal of  $-3/4$ :  $1/(-3/4) = -4/3$

Reciprocal of 0: "undefined"

30

### Selection Example – Reciprocals (cont)

**Q.** Give an algorithm for computing the reciprocal of a number.

*Algorithm:*

```

input Num
if (Num is not equal 0)
then
{
    output 1/Num
}
else
{
    output "infinity"
}
    
```

31

### Selection Example – Reciprocals (cont)

*Algorithm:*

```

input Num
if (Num is not equal 0)
then
{
    output 1/Num
}
else
{
    output "infinity"
}
    
```

For a given value of **Num**, only one of these two sequences can be executed

34

### Selection Example-- Reciprocals

*Algorithm:*

```

input Num
if (Num is not equal 0)
then
{
    output 1/Num
}
else
{
    output "infinity"
}
    
```

**Num** is a variable whose value depends on the actual number the user provides.

32

### Selection Example – Reciprocals (cont)

*Algorithm:*

```

input Num
if (Num is not equal 0)
then
{
    output 1/Num
}
else
{
    output "infinity"
}
    
```

Executed if **Num** is not equal to 0

35

### Selection Example – Reciprocals (cont)

*Algorithm:*

```

input Num
if (Num is not equal 0)
then
{
    output 1/Num
}
else
{
    output "infinity"
}
    
```

Condition depends on the value of **Num**

33

### Selection Example – Reciprocals (cont)

*Algorithm:*

```

input Num
if (Num is not equal 0)
then
{
    output 1/Num
}
else
{
    output "infinity"
}
    
```

Executed if **Num** is equal to 0

36

### Selection -- Exercise

Will the following algorithms produce the same output?

*Algorithm 1:*

```

input Num
if (Num is not equal 0)
then
{
  output 1/Num
}
else
{
  output "infinity"
}
            
```

*Algorithm 2:*

```

input Num
if (Num is not equal 0)
then
{
  output 1/Num
}
output "infinity"
            
```

37

### Selection – At Least One of Several Conditions

- ❖ What if at least one of several conditions needs to be satisfied?

```

if ( I feel hungry or the time is 1.00pm or my mate has his eye on my lunch )
then
{
  Eat my lunch now
}
            
```

40

### Selection – Several Conditions

- ❖ What if several conditions need to be satisfied?

```

if ( today is Wednesday and the time is 10.00am )
then
{
  Go to CSE1301 Lecture
}
else
{
  Go to Library
}
            
```

Solution 1

38

### Components of an Algorithm

- ❖ Values and Variables
- ❖ Instruction
- ❖ Sequence (of instructions)
- ❖ Procedure (involving instructions)
- ❖ Selection (between instructions)
- ❖ Repetition (of instructions)
- ❖ Documentation (beside instructions)

41

### Selection – Several Conditions (cont)

```

if ( today is Wednesday )
then
{
  if ( the time is 10.00am )
  then
  {
    Go to CSE1301 Lecture
  }
}
else
...etc...etc...etc...
            
```

Often called a **“nested selection”**

Solution 2

39

### Repetition

- ❖ Repeat an instruction...
  - ...while (or maybe until) some true or false condition occurs
  - Test the condition each time before repeating the instruction
- ❖ Also known as iteration or loop
- ❖ Example:
  - Algorithm for finding the summation from 1 to a given number

42

### Repetition -- Example

```

procedure Sum1_to_n(num)
{
  count = 1
  sum = 0
  while (count <= num)
  {
    add count to sum
    add 1 to count
  }
}
    
```

43

### Repetition -- Example (cont)

```

procedure Sum1_to_n(num)
{
  count = 1
  sum = 0
  while (count <= num)
  {
    add count to sum
    add 1 to count
  }
}
    
```

Ensure initial values of variables used in the conditions are set correctly

46

### Repetition -- Example (cont)

```

procedure Sum1_to_n(num)
{
  count = 1
  sum = 0
  while (count <= num)
  {
    add count to sum
    add 1 to count
  }
}
    
```

Condition is tested before sequence

44

### Repetition -- Example (cont)

```

procedure Sum1_to_n(num)
{
  count = 1
  sum = 0
  while (count <= num)
  {
    add count to sum
    add 1 to count
  }
}
    
```

Ensure the variables used in the conditions are updated in each iteration

47

### Repetition -- Example (cont)

```

procedure Sum1_to_n(num)
{
  count = 1
  sum = 0
  while (count <= num)
  {
    add count to sum
    add 1 to count
  }
}
    
```

Sequence may not get executed at all

45

### Repetition -- Example (cont)

What if we don't increment count?

```

procedure Sum1_to_n(num)
{
  count = 1
  sum = 0
  while (count <= num)
  {
    add count to sum
  }
}
    
```

Infinite loop

48



## Components of an Algorithm

- ❖ Values and Variables
- ❖ Instruction
- ❖ Sequence (of instructions)
- ❖ Procedure (involving instructions)
- ❖ Selection (between instructions)
- ❖ Repetition (of instructions)
- ❖ Documentation (beside instructions)

49

## Components of an Algorithm

- ❖ Values and Variables
- ❖ Instruction
- ❖ Sequence (of instructions)
- ❖ Procedure (involving instructions)
- ❖ Selection (between instructions)
- ❖ Repetition (of instructions)
- ❖ Documentation (beside instructions)

52

## Documentation

- ❖ Records what the algorithm does
- ❖ Describes how it does it
- ❖ Explains the purpose of each component of the algorithm
- ❖ Notes restrictions or expectations

50

## A Simple Algorithm

- ❖ INPUT: a sequence of  $n$  numbers,
  - $T$  is an array of  $n$  elements
  - $T[1], T[2], \dots, T[n]$
- ❖ OUTPUT: the smallest number among them

```
min = T[1]
for i = 2 to n do
{
  if T[i] < min
    min = T[i]
}
Output min
```

- ❖ Performance of this algorithm is a function of  $n$

53

## Documentation -- Example

```
// This procedure finds the summation from 1 to a given number
procedure Sum1_to_n(num)
{
  // do initialization
  count = 1
  sum = 0
  while (count <= num)
  {
    // add count to sum
    sum = sum + count
    // increment count by 1
    count = count + 1
  }
}
```

51

## Describing Algorithms

- ❖ Algorithms can be implemented in any programming language
- ❖ Usually we use "pseudo-code" to describe algorithms
- ❖ Testing whether input  $n$  is prime:

```
for j = 2 to n-1
{
  if mod(n, j) = 0
    output "n is non prime" and halt
}
output "n is prime"
```

54

## Algorithm Efficiency

- ❖ Consider two sort algorithms
  - Insertion sort
    - ❑ takes  $c_1 n^2$  to sort  $n$  items
    - ❑ where  $c_1$  is a constant that does not depend on  $n$
    - ❑ it takes time roughly proportional to  $n^2$
  - Merge Sort
    - ❑ takes  $c_2 n \lg(n)$  to sort  $n$  items
    - ❑ where  $c_2$  is also a constant that does not depend on  $n$
    - ❑  $\lg(n)$  stands for  $\log_2(n)$
    - ❑ it takes time roughly proportional to  $n \lg(n)$
  - Insertion sort usually has a smaller constant factor than merge sort
    - ❑ so that,  $c_1 < c_2$
  - Merge sort is faster than insertion sort for large input sizes

55

## Algorithm Efficiency

- ❖ Consider now:
  - A **faster** computer **A** running **insertion sort** against
  - A **slower** computer **B** running **merge sort**
  - Both must sort an array of **one million** numbers
- ❖ Suppose
  - Computer **A** execute **one billion** ( $10^9$ ) instructions per second
  - Computer **B** execute **ten million** ( $10^7$ ) instructions per second
  - So computer **A** is **100** times faster than computer **B**
- ❖ Assume that
  - $c_1 = 2$                       and                       $c_2 = 50$

56

## Algorithm Efficiency

- ❖ To sort one million numbers
  - Computer A takes
 
$$\frac{2 \cdot (10^6)^2 \text{ instructions}}{10^9 \text{ instructions/second}}$$
 = 2000 seconds
  - Computer B takes
 
$$\frac{50 \cdot 10^6 \cdot \lg(10^6) \text{ instructions}}{10^7 \text{ instructions/second}}$$
 = 100 seconds
- ❖ By using algorithm whose running time grows more slowly, Computer B runs 20 times faster than Computer A
- ❖ For ten million numbers
  - Insertion sort takes = 2.3 days
  - Merge sort takes = 20 minutes

57