# Ch 3. Algorithms

## Algorithms

Sungwon Kang

**Acknowledgement**

- [Rosen 19] Kenneth H. Rosen, for Discrete Mathematics & Its Applications (8th Edition), Lecture slides
- [Hunter 11] David J. Hunter, Essentials of Discrete Mathematics, 2nd Edition, Jones & Bartlett Publishers, 2011, Lecture Slides

# Ch 3. Algorithms

**3.1 Algorithms**

3.2 The Growth of Functions

3.3 Complexity of Algorithms

# Algorithms

1. **Definition**
2. **Algorithm Specification Languages**
3. **Specifying the Goal of an Algorithm**
4. **Algorithm Specification Examples**

# 1. Definition

**Algorithm**: A step-by-step procedure for solving a problem or accomplishing some end especially by a computer

Also people say,

An *algorithm* is a list of instructions for doing something.

☛ Can be compared with a "cooking recipe"

# 2. Algorithm Specification Languages

- Can be expressed in many ways:

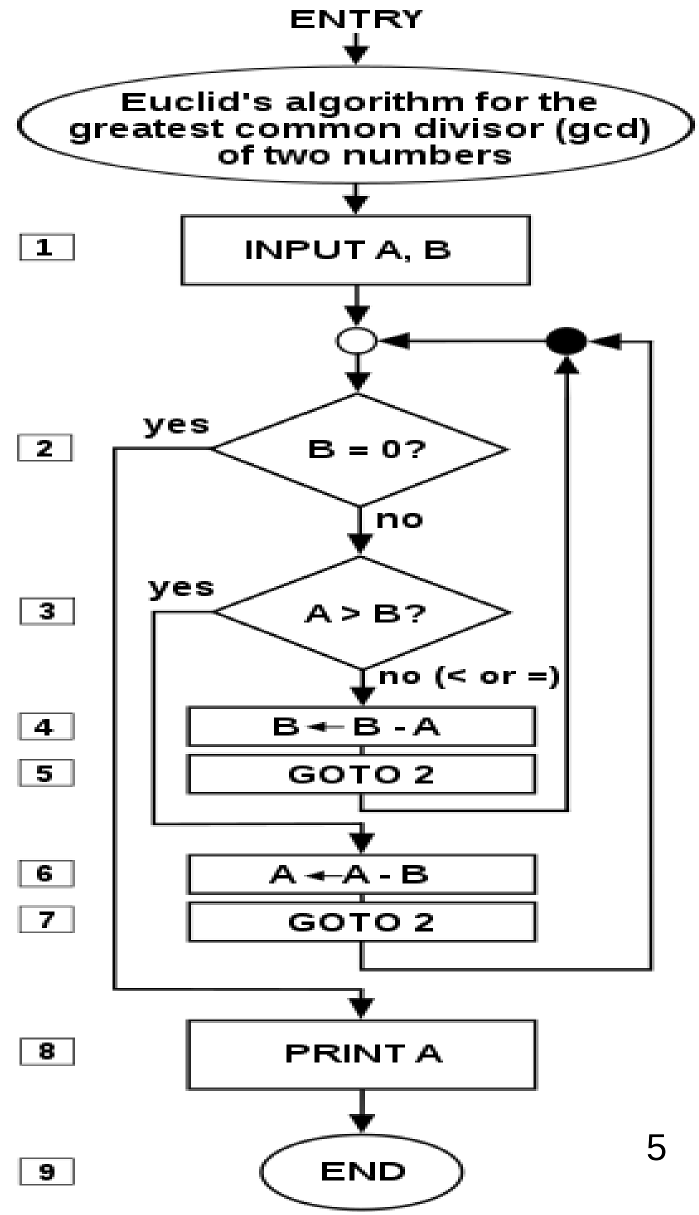
1) Natural languages

2) Computer Programming Languages

☛ 3) Flowchart

4) Pseudocode

Etc.




Flow chart of Euclid's algorithm for calculating the greatest common divisor of two numbers

Described using the following constructs:

(1) Sequencing ☚

(2) Branching

    (2A) if ... then ...

    (2B) if ... then ... else ...

(1) Looping

    (3A) Definite Loop: For-loop

    (3B) Indefinite Loop: While-loop

Described using the following constructs:

(1) Sequencing

(2) Branching ☚

(2A) if ... then ...

(2B) if ... then ... else ...

(1) Looping

(3A) Definite Loop: For-loop

(3B) Indefinite Loop: While-loop

Described using the following constructs:

(1) Sequencing

(2) Branching

(2A) if ... then ...

(2B) if ... then ... else ...

(1) Looping ☚

(3A) Definite Loop: For-loop

(3B) Indefinite Loop: While-loop

What about recursion?

☛ Recursion is essentially an indefinite loop !

# Specifying Algorithm in Pseudocode

**Pseudocode**: a convenient way of specifying algorithms or designs of computer programs

.

**Example**

*Start Program*
*Enter two numbers, A, B*
*Add the numbers together*
*Print Sum*
*End Program*

or

*Enter two numbers, A, B*
*Add the numbers together*
*Print Sum*

How can we specify more complicated algorithms in Pseudocode?

# Pseudocode

## Assignment statement

$$x \leftarrow y$$

## If...then statement

$$\texttt{if } \langle \text{condition} \rangle \texttt{ then } \langle \text{statement} \rangle$$

```
print "Old value of x:" x
if x > 5 then x ← x + 3
print "New value of x:" x
```

An example algorithm described using “if-then”

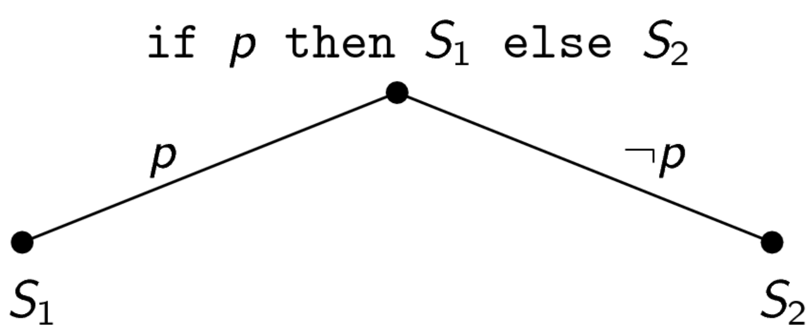
## If. . .then. . .else statements

Let $p$ be a logical statement that is either true or false. Then the `if...then...else` statement

$\texttt{if}\ p\ \texttt{then}$
$\quad\texttt{statement}_1$
$\texttt{else}$
$\quad\texttt{statement}_2$

Typical syntax for the "if-then-else" construct

will execute either $\texttt{statement}_1$ or $\texttt{statement}_2$, depending on whether $p$ is true or false, respectively.



# while-loop and for-loop

**While-loop**

Let $P(x)$ be a predicate statement. Then the *while-loop*

```
while P(x) do
      statement(x)
```

Typical syntax for "while-loop"

will continue to execute `statement`$(x)$ as long as $P(x)$ remains true.

**For-loop**

```
For x:= 1 to n do
    statement(x)
```

Typical syntax for "for-loop"

will repeat execution of statement(x) n times.

## Precondition and postcondition

A rigorous way of specifying **the goal or the requirements of an algorithm**:
*input condition* and *output condition*.

**Definition**

Let $A$ be an algorithm. A *precondition* of $A$ is a statement about state of the algorithm variables before $A$ executes. A *postcondition* of $A$ is a statement about the algorithm variables after execution.

# Example

## Sorting Algorithm

Preconditions: The elements of the array $x_1, x_2, x_3, \ldots, x_n$ can be compared by $\leq$. In addition, $n \geq 2$.

Postcondition: $x_1 \leq x_2 \leq x_3 \leq \cdots \leq x_n$.

# Example: precondition and postcondition

## Bubblesort

```
for i ∈ {1, 2, ..., n − 1} do
    for j ∈ {1, 2, ... n − i} do
        if x_j > x_{j+1} then swap x_j and x_{j+1}
```

Preconditions: The elements of the array $x_1, x_2, x_3, \ldots, x_n$ can be compared by $\leq$. In addition, $n \geq 2$.

Postcondition: $x_1 \leq x_2 \leq x_3 \leq \cdots \leq x_n$.

# 4. Algorithm Specification Examples

## Algorithm 1: Sequential search

**Goal**: Given a list of elements

<x1, x2, ... , xn>

and a target element t, find t in the list.

---

# Sequential search: pre/post

**Goal**: Given a list of elements
<x1, x2, ... , xn>
and a target element t, find t in the list.

---

Preconditions: $\{x_1, x_2, \ldots, x_n\} \subseteq U$ with $n \geq 1$
$t \in U$

Postconditions: $t = x_i$
$i \in \{1, 2, \ldots, n+1\}$
$(i = n+1) \Rightarrow (t \notin \{x_1, x_2, \ldots, x_n\})$

**Now let's specify the algorithm for sequential search !**

**Goal**: Given a list of elements
<x1, x2, ... , xn>
and a target element t, find t in the list.

---

$i \leftarrow 1$
$x_{n+1} \leftarrow t$ // $x_{n+1}$ is called the sentinel
`while` $t \neq x_i$ `do`
    $i \leftarrow i + 1$
`if` $i = n + 1$ `then`
    `print` Element $t$ was not found.
`else`
    `print` Element $t$ was found in location $i$.

# Algorithm 2: Binary Search (iterative)

Preconditions: $X = \{x_1, x_2, \ldots x_n\} \subseteq U$, with $n \geq 1$,
$x_1 < x_2 < \cdots < x_n$, and $t \in U$.

Postcondition: $(t \notin \{x_1, x_2, \ldots, x_n\}) \vee (x_l = t)$

Note that for binary search the input list must be sorted.

This is a subject of the "Data Structures" course.
So we will not go into the details of the algorithm too deeply.

Preconditions: $X = \{x_1, x_2, \ldots x_n\} \subseteq U$, with $n \geq 1$,
$x_1 < x_2 < \cdots < x_n$, and $t \in U$.

Postcondition: $(t \notin \{x_1, x_2, \ldots, x_n\}) \vee (x_l = t)$

$l \leftarrow 1, \; r \leftarrow n$<br>
**while** $l < r$ **do**<br>
    $i \leftarrow \lfloor (l + r)/2 \rfloor$<br>
    **if** $t > x_i$ **then**<br>
        $l \leftarrow i + 1$<br>
    **else**<br>
        $r \leftarrow i$<br>
**if** $t = x_l$ **then**<br>
    **print** Element $t$ was found in location $l$.<br>
**else**<br>
    **print** Element $t$ was not found.

# Trace of binary search (iterative)

Data:

| $n$ | $t$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|---|---|
| 5 | 12 | 3 | 6 | 9 | 12 | 15 |

Trace:

| $l$ | $r$ | $i$ | test | |
|---|---|---|---|---|
| 1 | 5 | | $1 \overset{?}{<} 5$ | 1 |
| | | $\lfloor \frac{1+5}{2} \rfloor = 3$ | $12 \overset{?}{>} 9$ | 2 |
| $3 + 1 = 4$ | | | $4 \overset{?}{<} 5$ | 3 |
| | | $\lfloor \frac{4+5}{2} \rfloor = 4$ | $12 \overset{?}{>} 12$ | 4 |
| | 4 | | $4 \overset{?}{<} 4$ | 5 |

# Algorithm 3 : Binary Search (recursive)

```
function BinSearch(t ∈ U,
                   X = {x_1, x_2, ... x_n} ⊆ U,
                   l, r ∈ {1, 2, ..., n})
    i ← ⌊(l + r)/2⌋
    if t = x_i then
        return true
    else
      ⌜ if (t < x_i) ∧ (l < i) then
            return BinSearch(t, X, l, i − 1)
        else
          ⌜ if (t > x_i) ∧ (i < r) then
                return BinSearch(t, X, i + 1, r)
            else
      ⌞   ⌞     return false
```

Data: $X = \{3, 6, 9, 12, 15, 18, 21, 24, 27, 30\}$.

Top-down evaluation:

$$\begin{aligned}
\texttt{BinSearch}(21, X, 1, 10) &= \texttt{BinSearch}(21, X, 6, 10) \\
&= \texttt{BinSearch}(21, X, 6, 7) \\
&= \texttt{BinSearch}(21, X, 7, 7) \\
&= \texttt{true}
\end{aligned}$$

## Comparison of Recursive Algorithm and Iterative Algorithm

| | **Recursive Algorithm** | **Iterative Algorithm** |
|---|---|---|
| Algorithm design | Easier | Harder |
| Correctness proof | Easier | Harder |
| Performance | Not Good | Good |

☛ **Solution**: Design a recursive algorithm first and transform it into an iterative algorithm