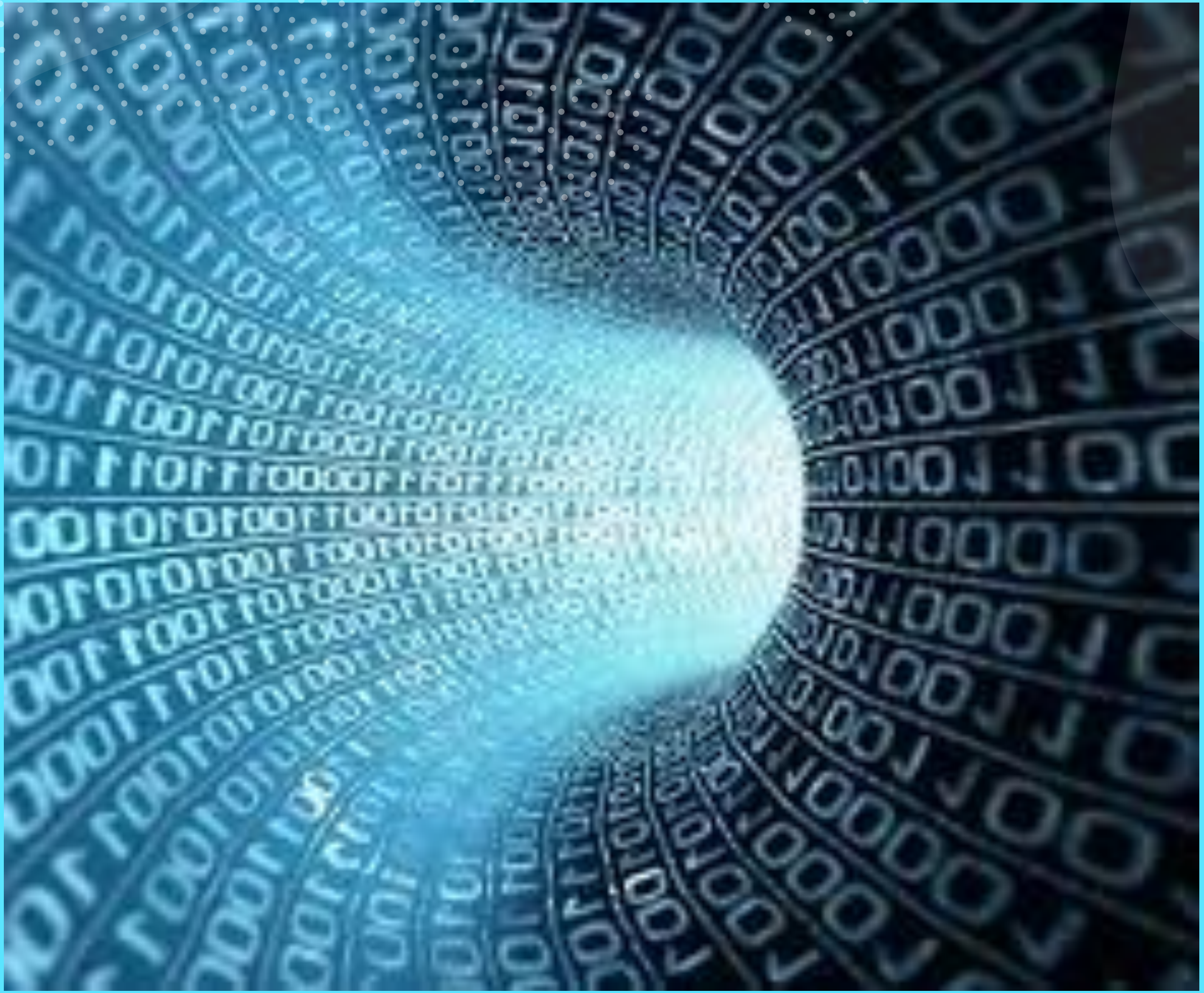
Algorithms

Lecture 1
Introduction to Algorithms

A. S. M. Sanwar Hosen

Email: sanwar@wsu.ac.kr

Date: 9th March 2023



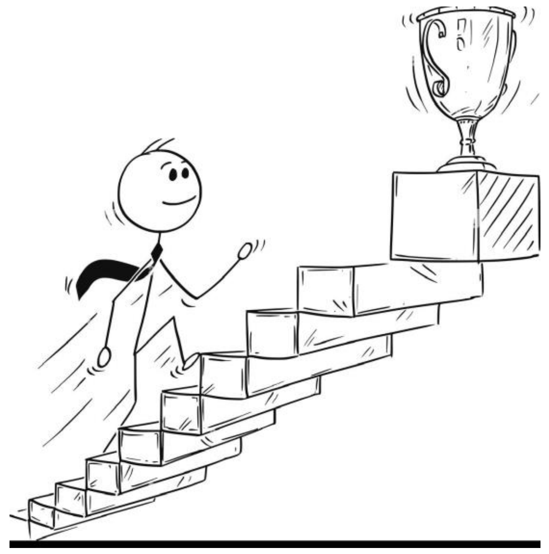
Introduction to Algorithms

- **What is an Algorithm?**

  An algorithm is a step-by-step process to solve a computing problem.

Process

```
Start
  Step 1
  Step 2
  Step 3
  Step 4
  Step …
End
```



- **What is a Process?**

  A process is a series of actions (instructions) to achieve a particular goal.

Introduction to Algorithms

- **Basic Components of an Algorithm:** The basic components of an algorithm are:
- ✓ Input and Output
- ✓ Arithmetic Calculation and Mathematical Functions
- ✓ Decision Making
- ✓ Repeated Calculation or Loops

Introduction to Algorithms

- **Input and Output:** Information provided to the program by a person, devices, environment, or another computer that are processed for the output. Output is the result produced by a computer after processing the given information. An example of input and output as below:

Example: Write a program to display or print a number on a computer screen.

```
Algorithm: Display a number on a computer screen
Start
  Step 1 (input): variable $x$, and assign a number to the variable   // i.e., $x$ = 5
  Step 2 (output): print($x$)                    // action or do
End
```

Introduction to Algorithms

❑ **Input and Output (cont..)**

Example: Write a program to display or print your name on a computer screen.

```
Algorithm: Print or display your name on a computer screen
Start
  Step 1 (input): variable named ‘my_name’ and assign your name to it   // i.e., my_name = ‘Jenifer’
  Step 2 (output): print(my_name)        // action or do
End
```

Introduction to Algorithms

- **Arithmetic Calculation and Mathematical Functions:** It provides the steps of process of arithmetic or mathematical operations/functions. Examples of arithmetic operators and functions are $+, -, *, \sin(x), \log(x)$ and so on.

Example: Write a program that gives the output of the following function $y = 2x + c$, where y is the output, x is a variable, and c is a constant. Given, x = 5 and c = 3.

```
Algorithm: Find the output of the function $y = 2x + c$
Start
  Step 1 (input): variables $x$ and $c$, and assign the values to them   // $x = 5$, $c$ =3
  Step 2 (statement): $y = 2 * x + c$    // action or do, * and + are arithmetic operators
  Step 3 (output): return $y$
End
```

Introduction to Algorithms

❑ Arithmetic Calculation and Mathematical Functions (cont..)

Example: Write a program that gives the output of the following function $y = \log(x) + c$, where y is the output, x is a variable, and c is a constant. Given, $x = z^2$, $z = 5$, and $c = 5$.

```
Algorithm: Find the output of the function $y = \log(x) + c$
Start
  Step 1 (input): variables named $x$, c and $z$, and assign the values to $z$ and $c$    // $z$ = 5, $c$ = 3
  Step 2 (statement): $x = z^2$                  // action or do; $z^2 = z * z$
  Step 3 (statement): $y = \log(x) + c$      // action or do
  Step 4 (output): return $y$
End
```

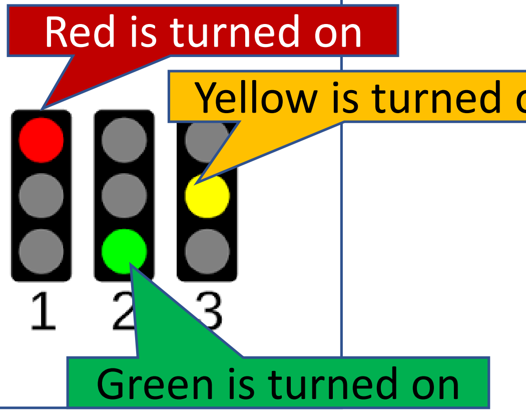
Introduction to Algorithms

- **Decision Making:** It decides the direction and flow of a next step based on various conditions. There are various statements used in programing such as if-else or nested if.

Example: Write a program that gives the direction of driving based on traffic lights (read, yellow and green are turned on and off) on a road.

Algorithm: Direction of driving based on traffic lights (turned on or turned off) on a road

```
Start
  Step 1 (input): variable named ‘light’ and assign the value to it    // light = read, green or yellow
  Step 2 (condition): if light is read then
              Step 2.1 (output): stop driving                      // action or do
  Step 3 (condition): if light is green then
              Step 3.1 (output): start driving and keep driving    // action or do
  Step 4 (condition): if light is yellow then
              Step 4.1 (output): stop driving when it is safe to   // action or do
End
```



Introduction to Algorithms

- **Decision Making**

Example: Write a program that generates a random number between 0 to 100, and adds with 100 if the random number is equal to 10.

Algorithm: Add 100 with a random number between 0 to 100, if the random number is equal to 10

```
Start
  Step 1 (input): variables named $x$ and $y = 0$, generate a random number and assign it to $x$  // $x$ = random(0,100)
  Step 2 (condition): if $x$ is equal to 10 then
        Step 2.1 (statement): $y = x + 100$        // action or do
  Step 3 (output): return $y$
End
```

Introduction to Algorithms

❑ Decision Making

Example: Write a program that generates a random number between 0 to 100, and does the following operations:

1) If the random number is greater than 20, adds the number with 100

2) If the random number is less than 20, adds the number with 200

3) If the random number is equal to 20, adds the number with 150

```
Algorithm: Add 100 with a random number between 0 to 100
Start
  Step 1 (input): variables named $x$ and $y = 0$, generate a random number and assign to $x$  // $x$ = random(0,100)
  Step 2 (condition): if $x$ is greater than 20 then
          Step 2.1 (statement): $y = x + 100$        // action or do
  Step 3: (condition): else if $x$ is less than 20 then
          Step 3.1 (statement): $y = x + 200$        // action or do
  Step 4: (condition): else if $x$ is equal to 20 then
          Step 4.1 (statement): $y = x + 150$        // action or do
  Step 5 (output): return $y$
End
```

Introduction to Algorithms

- **Repeated Calculation or Loops:** It refers to a sequence of instructions or code being repeated until a specific result is achieved. In programing, the repetition of instructions are conducted by ‘for loops’, ‘while loops’, and recursion.

✓ **For Loop:** ‘for’ component in “for loop” refers to do something (actions) for a certain number of times.

Example: Write a program that displays or prints your name 10 times on a computer screen.

```
Algorithm: Display your name 10 times on a computer screen
Start
   Step 1 (input): variables named ‘my_name’, and assign your name to it  // my_name = ‘Jenifer’
                   , and ‘count’ and assign an initial value 0 to it  // count = 0
   Step 2 (loop and number of times): for count in range(0, 9)
        Step 2.1  (output): print(my_name)            //action or do
        Step 2.2 (statement): count = count + 1        //action or do (increment by 1)
End
```

Introduction to Algorithms

❑ **Repeated Calculation or Loops (cont..)**

✓ **While Loop:** It allows to repeatedly execute a block of code as long as a specified condition is true.

Example: Prints the numbers from 1 to 5.

```
Algorithm: Prints the numbers from 1 to 5
Start
  Step 1 (input): variables named ‘$x$’ and ‘$count$’, and assign an initial value to them   // x = 5, count = 1
  Step 2 (while loop and number of times): while count is less than and equal to $x$ do
      Step 2.1  (statement): print($count$)           //action or do
      Step 2.2 (statement): $count = count + 1$       //action or do (increment by 1)
End
```

Introduction to Algorithms

❑ **Repeated Calculation or Loops (cont..)**

✓ **Recursion:** It involves a function calling itself, directly or indirectly.

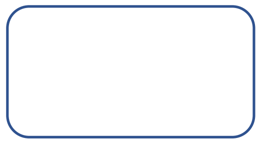
Example: Calculates the factorial of a number $n = 5$.

```
Algorithm: Factorial of a number
Start
  Step 1 (input): variable named '$n$' and assign an initial value to it   // $n$ = 5
  Step 2 (define the function of the factorial): def factorial($n$)
      Step 2.1 (condition): if $n$ is equal to 0 then
          Step 2.1.1 (output): return 1
      Step 2.2 (condition): else $n$ is not equal to 1 then
          Step 2.2.1 (output):  return $n$*factorial ($n - 1$)
End
```

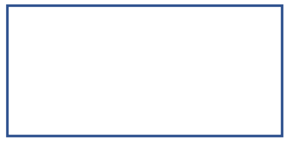
Introduction to Algorithms

- **Flowchart Diagram:** It represents a workflow or step-by-step process to solving a task. In a flowchart diagram, there are several symbols used to refer the state of the solution.

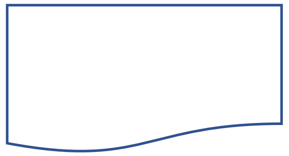
  Example: Flowchart symbols are as follows:



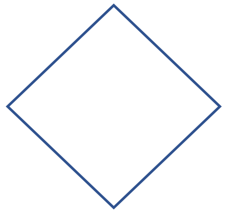
Terminator:
Refers to the starting or ending point of the system



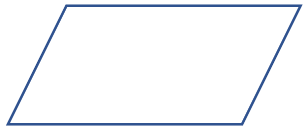
Process:
Indicates some operations



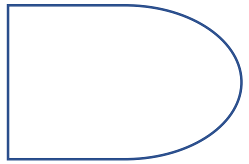
Document:
Indicates document or report



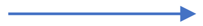
Decision:
Represents a decision or branching point



Data:
Represents information entering or leaving the system



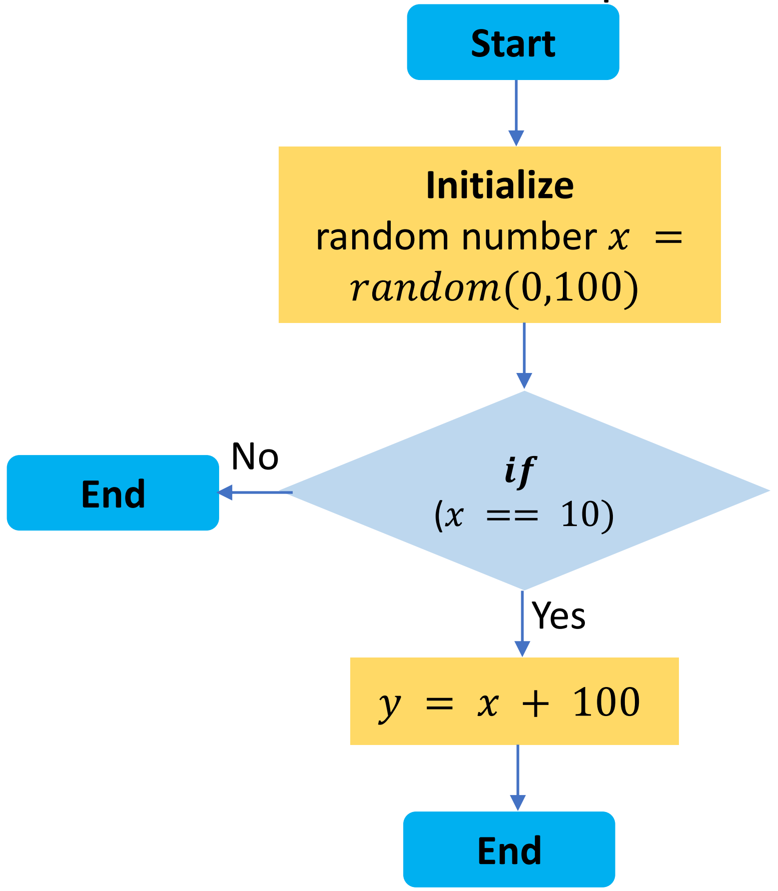
Delay or Bottleneck:
Identifies a delay or a bottleneck



Flow:
Represents the flow of the sequence and direction of a process

❑ Flowchart Diagram (cont..)

Example: Provides a flowchart diagram of a program that generates a random number between 0 to 100 and adds with 100 if the random number is equal to 10.



Introduction to Algorithms

- **Pseudocode:** Pseudocode is a plain language description of the steps in an algorithm or other system.

Example: Write a pseudocode of a program that generates a random number between 0 to 100 and adds with 100 if the random number is equal to 10.

```
Algorithm: Add 100 with a random number between 0 to 100,
           if the random number is equal to 10
Start
  initialize $y = 0$
  generate $x = random(0,100)$
    if $x == 10$ then
      $y = x + 100$
    return $y$
End
```

Introduction to Algorithms

- ❑ **Algorithm Analysis:** A process of analyzing the performance of an algorithm with respect to its time complexity, space complexity, and other performance metrices. There are two different methods for analyzing an algorithm as follows:

- ✓ **Theoretical Analysis:** It involves using mathematical tools to derive bounds on the worst-case, best-case, or average case running time of an algorithm.

- ✓ **Empirical Analysis:** Empirical analysis involves running the algorithm on different inputs and measuring its running time and memory usage.

Introduction to Algorithms

❑ Factors of an Algorithm Analysis

✓ **Time Complexity:** It is a measure of the amount of time it takes to run the algorithm as a function of the size of the input. The time complexity of an algorithm is often expressed by big O notation.
For example: The time complexity of an algorithm $O(n)$ will take at most linear time as the input size increases, while an algorithm with $O(n^2)$ will take at most quadratic time as the input size increases.

✓ **Cases of the Time Complexity:** There are three cases of time complexity of an algorithm as follows:
- Worst-case: It is the maximum amount of time that the algorithm takes to solve a problem given any input of size n.
- Best-case: It is the minimum amount of time that the algorithm takes to solve a problem given any input of size n.
- Average-case: It is the expected amount of time that the algorithm takes to solve the problem, given a random input of size n.

Introduction to Algorithms

❑ Factors of an Algorithm Analysis (cont..)

✓ **Space Complexity:** It is a measure of the amount of memory space required by the algorithm to solve a problem as a function of the size of the input. It describes how much memory the algorithm needs to store its variables, data structures, and other objects as it runs.

For example: The space complexity of an algorithm $O(n)$ will take at most linear space as the input size increases, while an algorithm with $O(n^2)$ will take at most quadratic space as the input size increases.

✓ **Cases of the Space Complexity:** There are three cases of space complexity of an algorithm as follows:
- Worst-case: This is the maximum amount of memory space required by the algorithm to solve a problem given any input of size n.
- Best-case: This is the minimum amount of memory space required by the algorithm to solve a problem given any input of size n.
- Average-case: It is the expected amount of memory space required by the algorithm to solve a problem given any input of size n.

Lecture Review

❑ This lecture discussed about:

✓ Algorithm

- Components of an Algorithm

✓ Flowchart Diagram

✓ Pseudocode

✓ Algorithm Analysis

- Time Complexity (Worst-case, Best-case, and Average-case)
- Space Complexity (Worst-case, Best-case, and Average-case)