Java Programming

Zheng-Liang Lu

Department of Computer Science & Information Engineering
National Taiwan University

Java 405
Spring 2024

```
class Lecture3 {

            "Flow Controls: Branching & Repetition"

}

// Keywords:
if, else, switch, case, break, default, yield, while, do, for,
continue
```

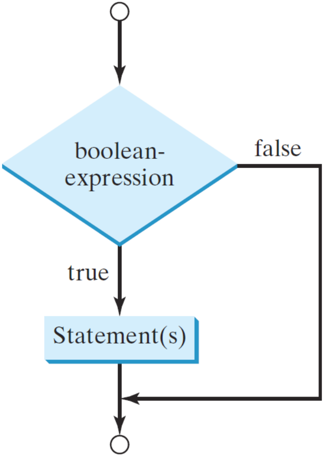
Flow Controls

- Most of statements are executed in sequential order.
- Programs can handle automatically with various situations when the branching (selection) rules are known.
- Moreover, programs may repeat some actions if necessary.
- For example, recall how to find the largest number in the list?

The if Branching Statement

```java
...
        if (/* Condition: a boolean expression */) {
            // Selection body: conditional statements.
        }
...
```

- If the condition is evaluated true, then the conditional statements will be executed once.
- If false, then the selection body will be ignored.
- Note that the braces can be omitted when the body contains only single statement.


boolean-
expression
false
true
Statement(s)

Example: Circle Area (Revisited)

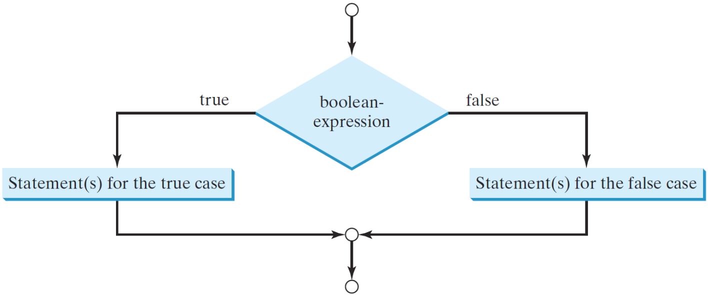
Write a program to receive a positive number as the circle radius and calculate its circle area.

```
...
        if (r > 0) {
            double A = r * r * 3.14;
            System.out.println(A);
        }
...
```

- What if the false case?

The if-else Statement

```
...
        if (/* Condition: a boolean expression */) {
            // Conditional statements for the true case.
        } else {
            // Conditional statement for the false case.
        }
...
```



Example: Circle Area (Revisited)

```java
...
        if (r > 0) {
            double A = r * r * 3.14;
            System.out.println(A);
        } else {
            System.out.println("Not a circle.");
        }
...
```

Nested Conditional Statements: Example

```java
...
        if (score >= 90)
            System.out.println("A");
        else {
            if (score >= 80)
                System.out.println("B");
            else {
                if (score >= 70)
                    System.out.println("C");
                else {
                    if (score >= 60)
                        System.out.println("D");
                    else
                        System.out.println("F");
                }
            }
        }
...
```

A Preferred Alternative: Multiple Branches

```java
...
        if (score >= 90)
            System.out.println("A");
        else if (score >= 80)
            System.out.println("B");
        else if (score >= 70)
            System.out.println("C");
        else if (score >= 60)
            System.out.println("D");
        else
            System.out.println("F");
...
```

- Avoid deep indentation to make your program easier to read!
- However, the order of conditions may be influential. (Why?)
- Furthermore, the runtime performance may degrade due to the order of conditions. (Why?)

Two Common Bugs

```
1 ...
2         if (r > 0);
3             double A = r * r * 3.14;
4             System.out.println(A);
5 ...
```

- Do not attach any semicolon to the condition (in Line 2).
  - If the parenthesis is followed by the semicolon in Line 2, Line 3 becomes unconditional and will be always executed.
- Multiple conditional statements should be enclosed by braces.

Example: Working with Uncertainty

> Write a program which (1) shows a math question, say sum of two random integers ranging from 0 to 9, (2) asks the user to answer the question, and then (3) judges this input.

- For example, the monitor displays "$2 + 5 =?$".
- If the user types 7, then the program reports "Correct."
- Otherwise, it reports "Wrong. The answer is 7."
- You can use **Math**.random() to generate random numbers.

Digression: How to Generate Random Numbers?[1]

- **Math**.random() produces numbers between 0.0 and 1.0, exclusive.
- To generate integers ranging from 0 to 9, it is clear that

$$(\text{int})\ (\textbf{Math}.\text{random}() \times 10),$$

  because there are 10 possible states: 0, 1, 2, . . . , 9.
- In general, you could generate any integer between L and H by using

$$(\text{int})\ (\textbf{Math}.\text{random}() \times (H - L + 1)) + L.\ \text{(Why?)}$$

[1]See https://en.wikipedia.org/wiki/Pseudorandom_number_generator.

```java
...
        // (1) Generate two random integers.
        int x = (int) (Math.random() * 10);
        int y = (int) (Math.random() * 10);

        // (2) Display the math question.
        System.out.println(x + " + " + y + " =  ?");

        // (3) Ask the user to type his/her answer.
        Scanner input = new Scanner(System.in);
        int z = input.nextInt();
        input.close();

        // (4) Judge the input.
        if (z == x + y) {
            System.out.println("Correct.");
        } else {
            System.out.println("Wrong.");
            System.out.println("It is " + (x + y) + ".");
        }
...
```

- Extend this program for all arithmetic operators ($+ - \times \div$).

"Exploring the unknown requires tolerating uncertainty."

– Brian Greene

"I can live with doubt, and uncertainty, and not knowing. I think it is much more interesting to live not knowing than have answers which might be wrong."

– Richard Feynman

Exercise

First generate 3 random integers ranging from -50 to 50, inclusive. Then find the largest value of these integers.

- Recall the first algorithm example in our class.

```java
...
        int x = (int) (Math.random() * 101) - 50;
        int y = (int) (Math.random() * 101) - 50;
        int z = (int) (Math.random() * 101) - 50;

        int max = x;
        if (y > max) max = y;
        if (z > max) max = z;
        System.out.println("MAX = " + max);
...
```

- However, this program is limited by the number of data.
- To develop a reusable solution, we need arrays and loops.

The switch-case-break-default Statement

```java
1  ...
2          switch (target) {
3              case v1:
4                  // Conditional statements.
5                  break; // Leaving (jump to Line 16).
6              case v2:
7              .
8              .
9              .
10             case vk:
11                 // Conditional statements.
12                 break; // Leaving (jump to Line 16).
13             default:
14                 // Default statements.
15         }
16 ...
```

- The variable *target* must be a value of char, byte, short, int, or **String** type.
- The type of $v_1, \ldots$, and v_k must be identical to *target*.
- A break statement should be necessary to leave the construct; otherwise, there will be a fall-through behavior.
- The default case is used to perform default actions when none of cases matches *target*.
  - Like the else statements.

Example

```java
...
        String symbol = "XS";

        int size;
        switch (symbol) {
            case "L":
                size = 10;
                break;
            case "M":
                size = 5;
                break;
            case "XS":
            case "S": // "XS" and "S" share the same action.
                size = 1;
                break;
            default:
                size = 0;
        }

        System.out.println(size); // Output 1.
...
```

New Syntax (1/3): No More Breaks[2]

```java
...
        String symbol = "XS";

        int size;
        switch (symbol) {
            case "L"        -> size = 10;
            case "M"        -> size = 5;
            case "S", "XS" -> size = 1;
            default         -> size = 0;
        }

        System.out.println(size); // Output 1.
...
```

[2] Since JDK12.

New Syntax (2/3): Switch Expressions

```
1  ...
2          String symbol = "XS";
3  
4          int size = switch (symbol) {
5              case "L"         -> 10;
6              case "M"         -> 5;
7              case "S", "XS" -> 1;
8              default          -> 0;
9          };
10 
11         System.out.println(size); // Output 1.
12 ...
```

- Like all expressions, switch expressions evaluate to a single value and can be used in statements, say Line 4.

New Syntax (3/3): yield

```java
...
        String symbol = "XS";

        int size = switch (symbol) {
            case "L":
                yield 10;
            case "M":
                yield 5;
            case "S", "XS":
                yield 1;
            default:
                yield 0;
        };

        System.out.println(size); // Output 1.
...
```

Conditional Operator: Example

```java
...
        if (num1 > num2)
            max = num1;
        else
            max = num2;

        // The above statement is equivalent to the following:
        max = num1 > num2 ? num1 : num2;
...
```

- If num1 > num2, then execute $\boxed{\text{max} = \text{num1}}$; otherwise, $\boxed{\text{max} = \text{num2}}$.

"We must all face the choice between what is right and what is easy."

– Prof. Albus Dumbledore,
Harry Potter and the Goblet of Fire, J.K. Rowling

"To be or not to be, that is the question."

– Prince Hamlet, *Hamlet*, William Shakespeare

Essence of Loops[3]

A loop is used to repeat statements.

- For example, output "Hello, Java." for 100 times.

```
...
        System.out.println("Hello, Java.");
        System.out.println("Hello, Java.");
        .
        . // Copy and paste for 97 times.
        .
        System.out.println("Hello, Java.");
...
```

[3] Try Celebrating 50 Years of Kids Coding.

```java
...
        int cnt = 0;
        while (cnt < 100) {
            System.out.println("Hello, Java.");
            cnt++;
        }
...
```

- This is a toy example to show the power of loops.
- In practice, any routine which repeats couples of times, so called patterns, can be done by wrapping them into a loop.

成也迴圈，敗也迴圈

- Loops provide substantial computational power.
- Loops bring an efficient way of programming.
- However, loops could consume a lot of time.[4]

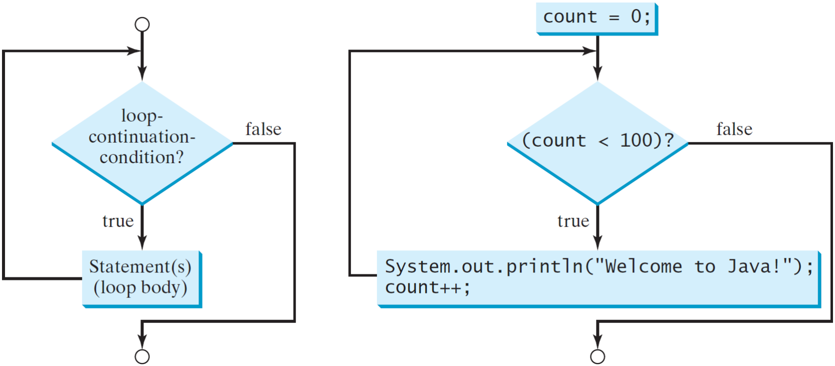
[4] You may check any algorithm textbook or course, say Algorithms Lab.

The while Loops

A while loop executes some statements repeatedly until the condition is false.

```
...
        while (/* Condition: a boolean expression */) {
            // Loop body.
        }
...
```

- If the condition is evaluated true, execute the loop body once and re-check the condition.
- The loop no longer continues when the condition is evaluated false.


loop-
continuation-
condition?
false
true
Statement(s)
(loop body)
count = 0;
(count < 100)?
false
true
System.out.println("Welcome to Java!");
count++;

Example: Summation

Write a program to sum up all integers from 1 to 100.

- In math,

$$\text{sum} = 1 + 2 + \cdots + 100.$$

- One may doubt why not $(1 + 100) \times 100/2$?
- The above formula is applicable to only arithmetic series!
- We don't assume the data being an arithmetic series. (Why?)
- To get a general solution, we decompose this summation into several statements, shown in the next page.

```java
...
        int sum = 0;
        sum = sum + 1;
        sum = sum + 2;
        .
        .
        .
        sum = sum + 100;
...
```

- As you can see, there exist many similar statements and we proceed to wrap them by using a while loop!

```java
...
        int sum = 0;
        int i = 1;
        while (i <= 100) {
            sum = sum + i;
            ++i;
        }
...
```

- Make sure that the loop terminates properly and outputs the correct result.
- In practice, the number of iterations often depends on the data size or the input parameter. (Why?)

Lurked Bugs: Malfunctioned Loops

- It is easy to make an infinite loop: always true.

```
...
        while (true);
...
```

- The common issues of writing loops are as follows:
  - loops never start;
  - loops never stop;
  - loops do not finish the expected iterations.

Example: Working with Uncertainty (Revisited)

Based on the previous program, allow the user to re-enter answers repeatedly until correct.

```
...
        ...

        while (z != x + y) {
            System.out.println("Try again?");
            z = input.nextInt();
        }
        System.out.println("Correct.");

        ...
...
```

Loop Design Strategy

- Identify the statements that need to be repeated.
- Wrap those statements by a loop.
- Set a proper continuation condition.

Indefinite Loops

Indefinite loops are the loops with unknown number of iterations.

- It is also called the sentinel-controlled loops, whose sentinel value is used to determine whether to execute the loop body.
- For example, the operating systems and the GUI apps.

Example: Cashier

Write a program to (1) sum over positive integers from consecutive inputs until the first non-positive integer occurs and (2) output the total value.

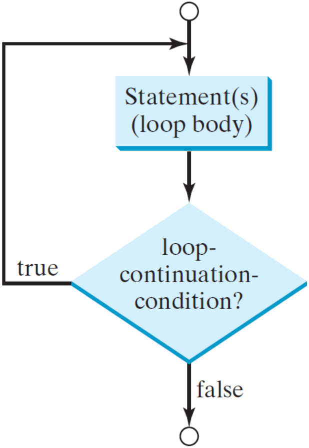
```java
...
        int total = 0, price = 0;
        Scanner input = new Scanner(System.in);

        System.out.println("Enter price?");
        price = input.nextInt();
        while (price > 0) {
            total += price;
            System.out.println("Enter price?");
            price = input.nextInt();
        }

        System.out.println("TOTAL = " + total);
        input.close();
...
```

The do-while Loops

A do-while loop is similar to a while loop except that it first executes the loop body and then checks the loop condition.

```
...
        do {
            // Loop body.
        } while (/* Condition: a boolean expression */);
...
```

- Do not miss a semicolon at the end of do-while loops.
- The do-while loops are also called the posttest loops, in contrast to the while loops, which are the pretest loops.


Statement(s)
(loop body)
loop-
continuation-
condition?
true
false

Example: Cashier (Revisited)

Write a program which sums over positive integers from consecutive inputs and then outputs the sum when the input is nonpositive.

```
...
        int total = 0, price = 0;
        Scanner input = new Scanner(System.in);

        do {
            total += price;
            System.out.println("Enter price?");
            price = input.nextInt();
        } while (price > 0);

        System.out.println("TOTAL = " + total);
        input.close();
...
```

The for Loops

A for loop uses an integer counter to control how many times the body is executed.

```
...
        for (initial-action; condition; increment) {
            // Loop body.
        }
...
```

- *initial-action*: declare and initialize a counter.
- *condition*: check if the loop continues.
- *increment*: how the counter changes after each iteration.

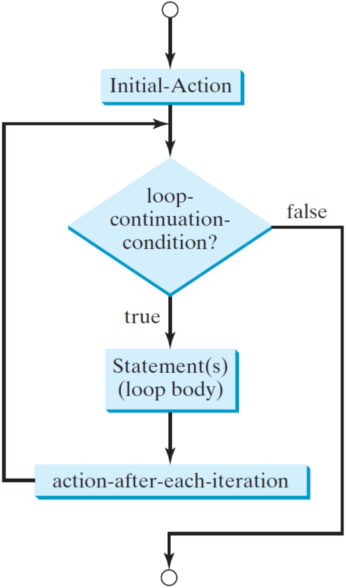
Example: Summation (Revisited)

Write a program to sum up the integers from 1 to 100.

```java
...
        int sum = 0;
        int i = 1;
        while (i <= 100) {
            sum = sum + i;
            ++i;
        }
...
```

```java
...
        int sum = 0;
        for (int i = 1; i <= 100; ++i)
            sum = sum + i;
...
```

- Note that the initial action `int i = 1` is executed only once.
- Make sure that you are clear with the execution flow of loops!


Initial-Action
loop-continuation-condition?
false
true
Statement(s)
(loop body)
action-after-each-iteration

Example: Even Numbers

Show all even integers from 1 to 100.

```java
...
        for (int i = 1; i <= 100; i++) { // Good?
            if (i % 2 == 0)
                System.out.println(i);
        }
...
```

```java
...
        for (int i = 2; i <= 100; i += 2) { // Which is better?
            System.out.println(i);
        }
...
```

Exercises

- Calculate the factorial of nonnegative integer N.[5]
  - For example, $10! = 3628800$.
- Calculate x^n with double value x and integer n.
  - For example, $2.0^{10} = 1024.0$.
- Calculate the following summation

$$p = 4 \times \sum_{i=0}^{10000} \frac{(-1)^i}{2i+1}.$$

  - The result is around 3.14.
  - Note that $p \to \pi$ as $N \to \infty$.

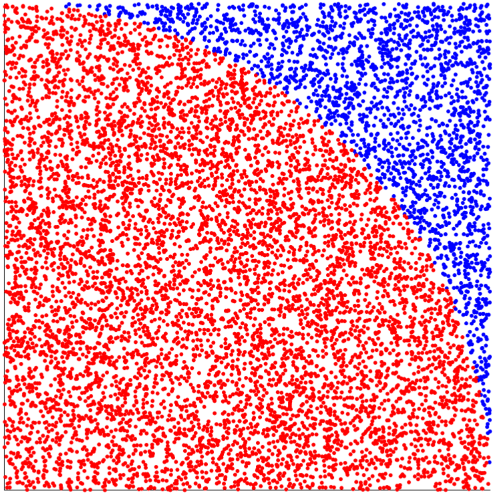
[5] See https://en.wikipedia.org/wiki/Factorial.

Numerical Example: Monte Carlo Simulation[6]

- Write a program to estimate π.
- Let N be the total number of points and M be the number of points falling in a quarter circle, illustrated in the next page.
- The algorithm states as follows:
  - For each round, draw a point by invoking **Math**.random() twice and check if the point falls in the quarter circle.
  - If so, then do $\boxed{\text{M++}}$; otherwise, ignore it.
  - Repeat the previous two steps for N rounds.
- Hence we can calculate the estimate

$$\hat{\pi} = 4 \times \frac{M}{N}.$$

[6]See https://en.wikipedia.org/wiki/Monte_Carlo_method.



```java
...
        int N = 100000;
        int M = 0;

        for (int i = 1; i <= N; i++) {

            double x = Math.random();
            double y = Math.random();

            if (x * x + y * y < 1) M++;

        }

        System.out.println("pi ~ " + 4.0 * M / N);
        // Why 4.0 but not 4?
...
```

- Note that $\hat{\pi} \to \pi$ as $N \to \infty$ by the law of large numbers (LLN).[7]
- This algorithm is one example of Monte Carlo simulation.[8]

[7] See https://en.wikipedia.org/wiki/Law_of_large_numbers.

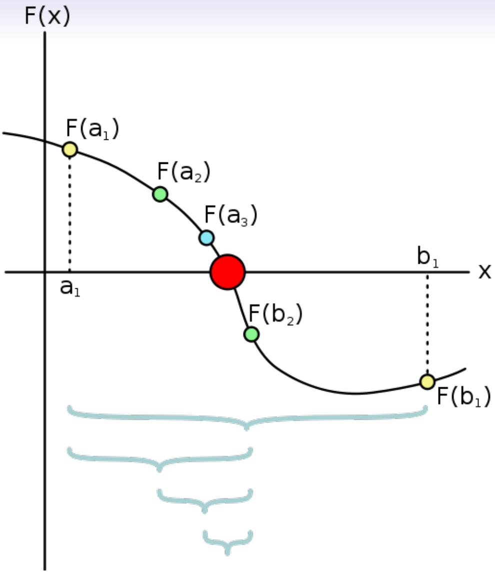
[8] See https://en.wikipedia.org/wiki/Monte_Carlo_method.

Numerical Example: Root Finding

- Consider to find the root for the polynomial $x^3 - x - 2$.
- Choose $a = 1$ and $b = 2$ as initial guess.[9]
- By the bisection method[10], divide the search interval into two sub-intervals, and decide which sub-interval is the next search interval.
- The algorithm will stop to output the approximate root when it meets the preset error tolerance, say $\varepsilon = 10^{-9}$. (Why?)
- This strikes a balance between efficiency and accuracy.

[9] For most of numerical algorithms, say Newton's method, we need an initial guess to start the root-finding procedure. Even more, the result is severely sensitive to an initial guess.

[10] It is also called the binary search. See Bisection Method.



https://en.wikipedia.org/wiki/Bisection_method#/media/File:Bisection_method.svg

```java
...
        double a = 1, b = 2, c = 0, eps = 1e-9;

        while (b - a > eps) {

            c = (a + b) / 2; // Find the middle point.

            double fa = a * a * a - a - 2;
            double fc = c * c * c - c - 2;

            if (fa * fc < 0) {
                b = c;
            } else {
                a = c;
            }

        }

        System.out.println("Root = " + c);
        double residual = c * c * c - c - 2;
        System.out.println("Residual = " + residual);
...
```

Jump Statements: Example

The statement break and continue are often used to provide additional controls in repetition structures.

```java
for (int i = 1; i <= 5; ++i) {

    if (i == 3) {
        break;
        // Early termination.
    }

    System.out.println(i);
}
// Output: 1 2
```

```java
for (int i = 1; i <= 5; ++i) {

    if (i == 3) {
        continue;
        // Skip this round.
    }

    System.out.println(i);
}
// Output: 1 2 4 5
```

Example: Primality Test[11]

Write a program to check if the input integer is a prime number.

- Let x be any integer larger than 2.
- Then x is a prime number if x has no positive divisors other than 1 and itself.
- It is straightforward to divide x by all integers from 2 to $x - 1$.
- To speed up, divide x by only integers smaller than $\sqrt{x}$ instead of x. (Why?)

[11] See https://en.wikipedia.org/wiki/Primality_test.

```java
...
        Scanner input = new Scanner(System.in);
        System.out.println("Enter x > 2?");
        int x = input.nextInt();
        boolean isPrime = true;
        input.close();

        for (int y = 2; y <= Math.sqrt(x); y++) {
            if (x % y == 0) {
                isPrime = false;
                break;
            }
        }

        if (isPrime) {
            System.out.println("Prime");
        } else {
            System.out.println("Composite");
        }
...
```

Example: Cashier (Revisited)

```java
...
        while (true) {

            System.out.println("Enter price?");
            price = input.nextInt();
            if (price <= 0) break; // Stop criteria.
            total += price;

        }
        System.out.println("Total = " + total);
...
```

Remarks

- The while loops are equivalent to the for loops.
- You can always rewrite the for loops by the while loops, and versa.
- In practice, you could use a for loop when the number of repetitions is known.
- Otherwise, a while loop is preferred.

One More Example: Compounding

> Write a program to determine the holding years for an investment doubling its value.

- Let *balance* be the current amount, *goal* be the goal of this investment, and r be the annual interest rate (%).
- The compounding formula is represented in recursive form:

$$balance = balance \times (1 + r \,/\, 100.0).$$

- Output the holding years with the final balance.

```java
...
        int r = 18; // In percentage.
        int balance = 100;
        int goal = 200;

        int years = 0;
        while (balance < goal) {
            balance *= (1 + r / 100.0);
            years++;
        }

        System.out.println("Holding years = " + years);
        System.out.println("Balance = " + balance);
...
```

- If the interests are paid monthly, how many months you may hold to reach the goal?

```java
...
        int years = 0; // Should be declared here; scope issue.
        for (; balance < goal; years++) {
            balance *= (1 + r / 100.0);
        }
...
```

```java
...
        int years = 1; // Why?
        for (; ; years++) {
            balance *= (1 + r / 100.0);
            if (balance >= goal) break;
        }
...
```

- Leaving the condition blank assumes true.

Nested Loops: Example

Write a program to print the 9×9 multiplication table.

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

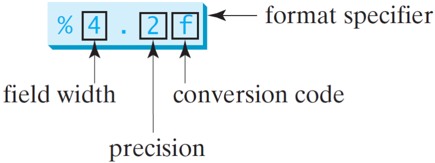
```java
...
    public static void main(String[] args) {

        for (int i = 1; i <= 9; ++i) {

            // In row i, output each i * j.
            for (int j = 1; j <= 9; ++j) {
                System.out.printf("%3d", i * j);
            }
            System.out.println();

        }

    }
...
```

- For each i, the inner loop goes from $j = 1$ to $j = 9$.
- As an analog, i acts like the hour hand of the clock, while j acts like the minute hand of the clock.

Digression: Output Format

- Use **System**.out.printf() to display formatted outputs.
- For example,

```
...
        System.out.printf("Pi = %4.2f", 3.1415926);
        // Output 3.14.
...
```



- Without specifying the width, only 6 digits after the decimal point are displayed.

Format specifier	Corresponding type	Example
%b	boolean	true, false
%c	char	a
%d	int	123
%f	float, double	3.141592
%e	float, double	6.626070e−34
%s	String	NTU

- By default, the output is right justified.
- If a value requires more spaces than the specified width, then the width is automatically increased.
- You may try various parameters such as the plus sign (+), the minus sign (-), and 0 in the middle of format specifiers.
    - Say %+8.2f, %−8.2f, and %08.2f.

Formatted Output with Multiple Items

items

```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```

display count is 5 and amount is 45.560000

- All items must match the format specifiers in order, in number, and in exact type.

Exercise: Triangles

```
*                 * * * * *                 *       * * * * *
* *               * * * *                 * *         * * * *
* * *             * * *                 * * *           * * *
* * * *           * *                 * * * *             * *
* * * * *         *                 * * * * *               *
  Case (a)          Case (b)          Case (c)          Case (d)
```

```java
...

        // Case (a)
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.printf("*");
            }
            System.out.println();
        }

        // Case (b)
        // Your work here.

        // Case (c)
        // Your work here.

        // Case (d)
        // Your work here.

...
```

Analysis of Algorithms

- A problem may be solved by various algorithms.
- We compare these algorithms by measuring their efficiency.
- Adopting a theoretical approach, we identify the growth rate of running time in function of input size n.
- This introduces the notion of time complexity.[12]
- Let's analyze the following two examples.

[12] See https://en.wikipedia.org/wiki/Time_complexity. Similar to time complexity, we later turn to the notion of space complexity.

Example 1: SUM

```java
...
        int sum = 0, i = 1; // Assign          -> 2.
        while (i <= n) {    // Compare         -> n + 1.
            sum = sum + i;  // Add and assign -> 2n.
            ++i;            // Increase by 1  -> n.
        }
...
```

- Let n be any nonnegative number.
- Then count the number of all runtime operations.
- Note that we ignore declarations in the calculation. (Why?)
- In this case, the total number of operations is $4n + 3$.

Example 2: TRIANGLE

```
*
* *
* * *
* * * *
* * * * *
⋮
```

```java
...
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++)
                System.out.printf("*");
            System.out.println();
        }
...
```

- We estimate the time cost by counting the total number of asterisks:

$$1 + 2 + \cdots + n = \frac{(1+n) \times n}{2}.$$

Big O Notation[13]

- Let $f(n)$ be the time cost of your algorithm, and $g(n)$ be some simple function.
- We define

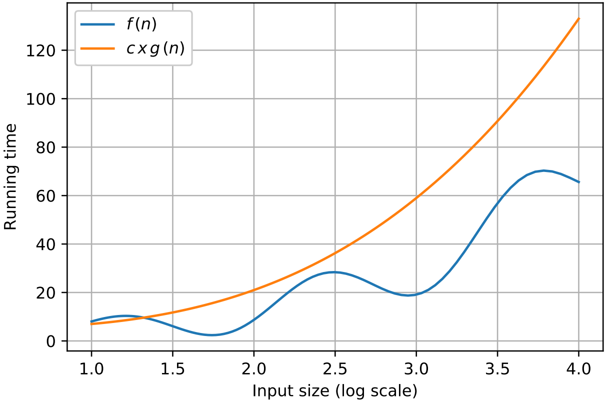
$$f(n) = O(g(n)) \text{ as } n \to \infty$$

provided that there is a constant $c > 0$ and some n_0 such that

$$f(n) \leq c \times g(n), \quad \forall n \geq n_0.$$

- No clue? See the illustration shown in the next page.

[13] See https://en.wikipedia.org/wiki/Big_O_notation. You can also check the other 4 symbols (o, Θ, Ω, and ω) in any algorithm textbook.



- Clearly, $g(n)$ is the asymptotic upper bound of $f(n)$.[14]
- In other words, Big O implies the worst case of the algorithm.
- We then classify the algorithms in Big O sense.

[14] See https://en.wikipedia.org/wiki/Big_O_notation#Infinite_asymptotics.

Discussions (1/4)

- Assume that the algorithm takes $8n^2 - 3n + 4$ steps.
- When n becomes large enough, the leading term dominates the whole behavior of the polynomial.
- So we simply focus on the leading term.
- It is easy to find a constant, say $c = 9$, so that $9n^2 \geq 8n^2$ holds.
- We then conclude that

$$8n^2 - 3n + 4 = O(n^2).$$

- It could say that the algorithm runs in $O(n^2)$ time.

Discussions (2/4)

- It is clear that SUM runs in $O(n)$ time and TRIANGLE runs in $O(n^2)$ time. (Why?)
- As a thumb rule, k-level loops run in $O(n^k)$ time.
- Determine the time complexity for the loop shown below.

```
...
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++) {
                for (int k = 1; k <= 5; k++) {
                    // Loop body.
                }
            }
        }
        // This algorithm runs in O( ? ) time.
...
```

Discussions (3/4): Which Will You Choose?

Benchmark

Size	$O(n)$	$O(n^2)$	$O(n^3)$
1	c_1	c_2	c_3
10	$10c_1$	$100c_2$	$1000c_3$
100	$100c_1$	$10000c_2$	$1000000c_3$

- In theory, the smaller the order, the faster the algorithm.

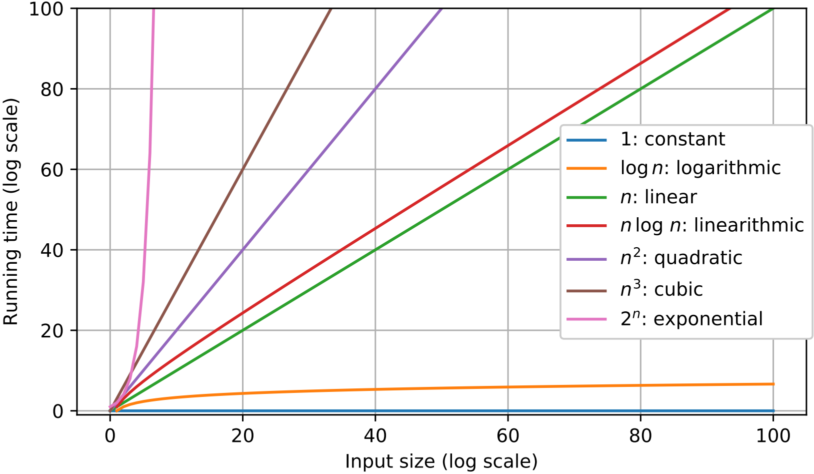
Discussions (4/4)

- It is worth to note that

$$8n^2 - 3n + 4 \neq O(n), \text{ and } 8n^2 - 3n + 4 = O(n^3). \text{ (Why?)}$$

- We would say that $8n^2 - 3n + 4 = O(n^2)$ for complexity analysis. (Why?)

Orders of Growth Rates



Big O Table

Growth order	Description	Example
$O(1)$	independent of n	x = y + z
$O(\log n)$	divide in half	binary search
$O(n)$	one loop	find maximum
$O(n \log n)$	divide and conquer	merge sort
$O(n^2)$	double loop	check all pairs
$O(n^3)$	triple loop	check all triples
$O(2^n)$	exhaustive search	check all subsets

Constant-Time Algorithms

- Basic instructions (e.g. $+$) run in $O(1)$ time. (Why?)
- Some algorithms indeed run in $O(1)$ time, for example, the arithmetic formulas. (Why?)
- However, there is no free lunch. (Why?)
- We should strike a balance by making a trade-off between generality and efficiency.
    - To reuse the program, it must be a general solution whose assumption should be little and weak.
    - To speed up the program, it could be optimized for the desire cases (so making assumptions).

- In addition, a program without writing explicit loops may not run in $O(1)$ time.
- For example, calling **Arrays**.sort() still takes more than $O(1)$ time to finish the sorting task.
- All in all, the time complexity is about the effort spent on the task but not how many time you sacrifice.

Exponential-Time Algorithms & Computability

- We, in fact, are overwhelmed by lots of intractable problems.
  - For example, the travelling salesman problem (TSP).[15]
  - Playing game well is hard.[16]
- Even worse, Turing (1936) proved the first undecidable (unsolvable) problem, called the halting problem.[17]
- You can find any textbook for theory of computation or computational complexity for further details.

[15] See https://en.wikipedia.org/wiki/Travelling_salesman_problem.

[16] See https://en.wikipedia.org/wiki/Game_complexity. Check out AlphaGo.

[17] See https://en.wikipedia.org/wiki/Halting_problem.

Logarithmic-Time Algorithms

- We have met one of logarithmic-time algorithms. (Which?)
- In conclusion, the log-time algorithms run much faster than the linear-time algorithms.
- However, the log-time algorithms require one assumption: ordered sequence.
- You will learn this kind of algorithms in any course about algorithms and data structures.

Outstanding Theoretical Problem[19]

$$\mathbb{P} \stackrel{?}{=} \mathbb{NP}$$

- In layman's term, $\mathbb{P}$ is the problem set of "being solved and verified in polynomial time."
- $\mathbb{NP}$ is the problem set of "being verified in polynomial time but perhaps being solved in exponential time."
  - For example, id verification is easier than hacking an account.
- One could say that $\mathbb{P}$ is easier than $\mathbb{NP}$.
- $\mathbb{P} \stackrel{?}{=} \mathbb{NP}$ asks if $\mathbb{NP}$ is solved by $\mathbb{P}$.
- It is still an open issue and also one of the Millennium Prize Problems.[18]

[18] See https://en.wikipedia.org/wiki/Millennium_Prize_Problems.

[19] See https://en.wikipedia.org/wiki/P_versus_NP_problem.