C# Programming

Zheng-Liang Lu

Department of Computer Science & Information Engineering
National Taiwan University

Online Course

```
class Lecture4
{

                "Flow Controls: Loops"

}

// Keywords:
while, do, for, break, continue
```

Essence of Loops

A loop can be used to repeat statements without writing the similar statements.

- For example, output "Hello, C#." for 100 times.

```
...
        Console.WriteLine("Hello, C#.");
        Console.WriteLine("Hello, C#.");
        .
        . // Copy and paste for 97 times.
        .
        Console.WriteLine("Hello, C#.");
...
```

```
...
        int cnt = 0;
        while (cnt < 100)
        {
            Console.WriteLine("Hello, C#.");
            cnt++;
        }
...
```

- This is a toy example to show the power of loops.
- In practice, any routine which repeats couples of times[1] can be done by folding them into a loop.

[1] I prefer to call these routines "patterns."

成也迴圈，敗也迴圈

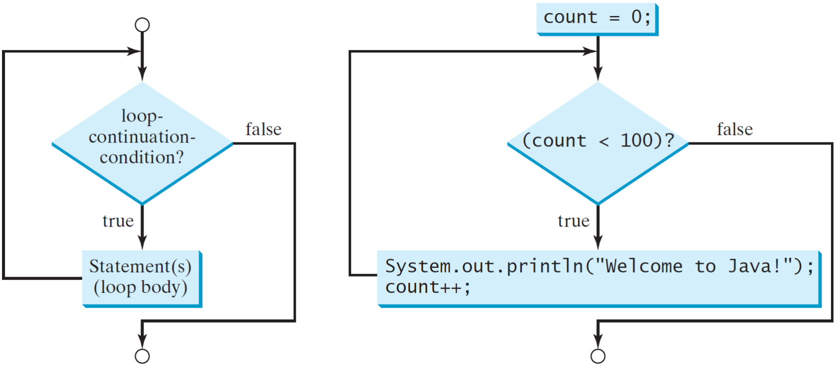
- Loops provide substantial computational power.
- Loops bring an efficient way of programming.
- Loops could consume a lot of time.
    - We will introduce the analysis of algorithms soon.

The while Loops

A while loop executes statements repeatedly while the condition is true.

```
...
        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 proceeds as soon as 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

- Write a program which sums up all integers from 1 to 100.
- In math,
$$\text{sum} = 1 + 2 + \cdots + 100.$$
- One could ask why not $(1 + 100) \times 100/2$?
- The above formula is suitable to only arithmetic series!
- We don't assume the data being an arithmetic series. (Why?)
- Instead, we rewrite the equation by decomposing it into several statements, shown in the next page.

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

- As you can see, there exist many similar statements to be wrapped by a loop!

- Using a while loop, the program can be rearranged as follows:

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

- You should guarantee that the loop will terminate as expected.
- In practice, the number of loop steps (iterations) is unknown until the input data is given.

Malfunctioned Loops

- It is easy to make an infinite loop.

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

- The common errors are as follows:
  - never start;
  - never stop;
  - not complete;
  - exceed the expected number of iterations;
  - (more and more.)

Example (Revisited)

- Write a program which allows the user to enter a new answer to the sum of two random integers repeatedly until correct.

```
...
        ...

        while (z != x + y)
        {
            Console.WriteLine("Try again?");
            z = int.Parse(Console.ReadLine());
        }
        Console.WriteLine("Correct.");

        ...
...
```

Loop Design Strategy

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

Sentinel-Controlled Loops

> Another common technique for controlling a loop is to designate a special value when reading and processing a set of values.

- This special value, known as the sentinel value, signifies the end of the loop.
- For example, operating systems (OS) and GUI apps.

Example: Cashier Problem

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

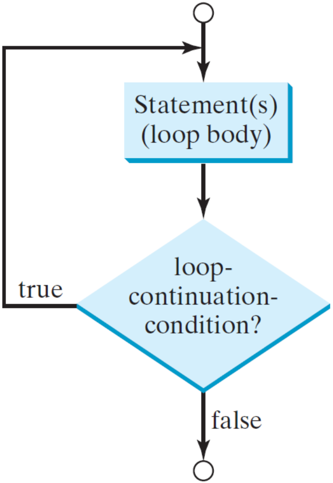
```
1  ...
2          int total = 0, price = 0;
3
4          Console.WriteLine("Enter price?");
5          price = int.Parse(Console.ReadLine());
6          while (price > 0)
7          {
8              total += price;
9              Console.WriteLine("Enter price?");
10             price = int.Parse(Console.ReadLine());
11             // These two lines above repeat Line 5 and 6?!
12         }
13
14         Console.WriteLine("Total = {0}", total);
15         input.close();
16 ...
```

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 post-test loops, in contrast to while loops, which are pre-test loops.


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

Example (Revisted)

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;

        do
        {
            total += price;
            Console.WriteLine("Enter price?");
            price = int.Parse(Console.ReadLine());
        }
        while (price > 0);

        Console.WriteLine("Total = {0}", total);
...
```

The for Loops

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

```
...
        for (init_action; condition; increment)
        {
            // Loop body.
        }
...
```

- *init_action*: declare and initialize a counter.
- *condition*: loop continuation.
- *increment*: how the counter changes after each iteration.

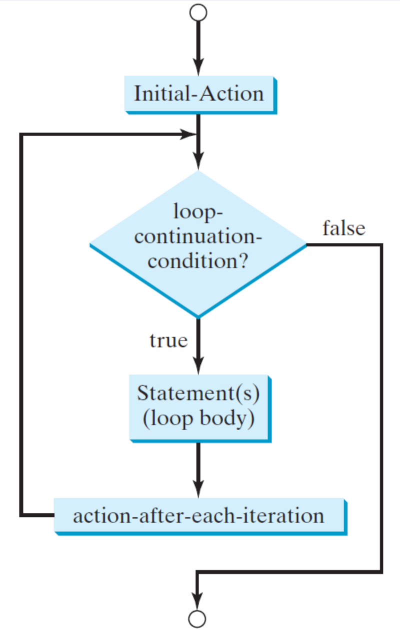
Example

Write a program which sums from 1 up to 100.

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

```
1 ...
2         int sum = 0;
3         for (int i = 1; i <= 100; ++i)
4         {
5             sum = sum + i;
6         }
7 ...
```

- Note that the first loop statement in Line 3 of the right listing is executed only once.
- Make sure that you are fully clear with the execution procedure of for loops!


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

Exercise

Write a program which displays all even numbers between 1 and 100.

- You may use the modular operator (%).

```
...
        for (int i = 1; i <= 100; i++) // Good?
        {
            if (i % 2 == 0)
                Console.WriteLine(i);
        }
...
```

- Also consider this alternative:

```
...
        for (int i = 2; i <= 100; i += 2) // Which is better?
        {
            Console.WriteLine(i);
        }
...
```

More Exercises

- Write a program to calculate the factorial of $N \geq 0$.[2]
  - For example, $10! = 3628800$.
- Write a program to calculate x^n, where x is a double value and n is an integer.
  - For example, $2.0^{10} = 1024.0$.
- Write a program to calculate

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

  - For example, the program outputs 3.141492 with $N = 10000$.
  - In math, $p \to \pi$ as $N \to \infty$.
  - Make friends with math.

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

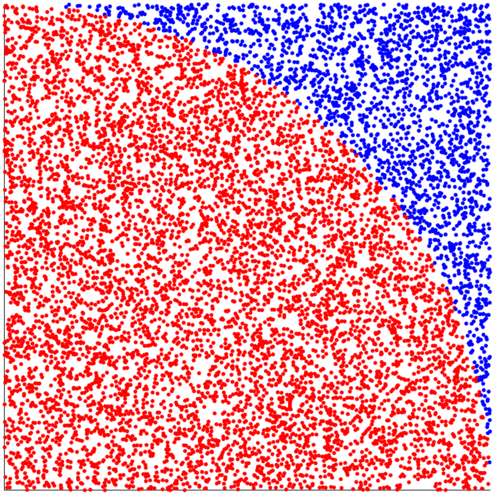
Numerical Example: Monte Carlo Simulation[3]

- Let n be the total number of sample points and m be the number of sample points falling in a quarter circle (shown in the next page).
- Write a program to estimate π by calculating

$$\hat{\pi} = 4 \times \frac{m}{n},$$

where $\hat{\pi} \to \pi$ as $n \to \infty$ by the law of large numbers (LLN).

[3] See https://en.wikipedia.org/wiki/Monte_Carlo_method. Also read https://medium.com/@jonathan_hui/monte-carlo-tree-search-mcts-in-alphago-zero-8a403588276a.



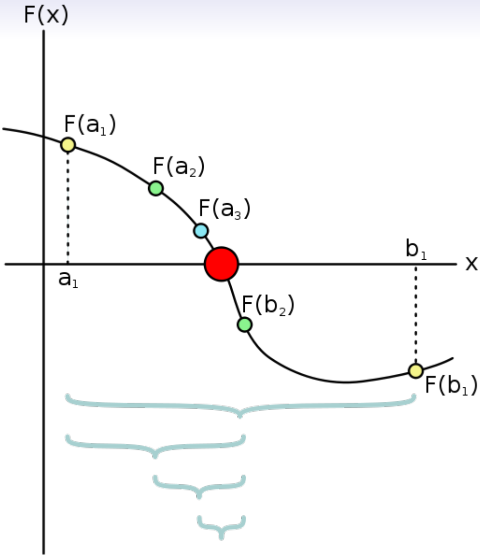
```
public class MonteCarloDemo
{
    static void Main(string[] args)
    {
        Random rng = new Random();
        int N = 100000;
        int m = 0;

        for (int i = 1; i <= N; i++)
        {
            double x = rng.NextDouble(); // Ranging from 0 to 1.
            double y = rng.NextDouble();

            if (x * x + y * y < 1) m++;
        }

        Console.WriteLine("pi = {0}", 4.0 * m / N);
        // Why 4.0 but not 4?
    }
}
```

Numerical Example: Bisection Method for Root-Finding[5]

- Consider the polynomial $x^3 - x - 2$.
- Now we proceed to find the root x' such that $x'^3 - x' - 2 = 0$.
- First choose $a = 1$ and $b = 2$ as an initial guess.[4]
- By using the bisection method, repeatedly divide the search interval into two sub-intervals, and decide which sub-interval is the next search interval.
- Due to finite precision of floats, we terminate the algorithm earlier by setting an error tolerance, say $\varepsilon = 1e - 9$, to strike a balance between efficiency and accuracy.

[4] 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.

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



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

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

        while (b - a > eps)
        {
            c = a + (b - a) / 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;
        }

        Console.WriteLine("Root = {0}", c);
        double residual = c * c * c - c - 2;
        Console.WriteLine("Residual = {0}", residual);
...
```

Jump Statements

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

- The loop is terminated right after a break statement is executed.
- The loop skips this iteration right after a continue statement is executed.
- In practice, jump statements should be conditioned.

Example: Primality Test[6]

Write a program which determines if the input integer is a prime number.

- Let $x > 1$ be any natural number.
- 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 natural numbers smaller than x.
- For speedup, you can divide x by only numbers smaller than $\sqrt{x}$. (Why?)

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

```
...
        Console.WriteLine("Enter x > 2?");
        int x = int.Parse(Console.ReadLine());
        bool isPrime = true;

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

        if (isPrime)
            Console.WriteLine("Prime");
        else
            Console.WriteLine("Composite");
...
```

Another Example: Cashier Problem (Revisited)

- Redo the cashier problem by using an infinite loop with a break statement.

```
...
        while (true)
        {
            Console.WriteLine("Enter price?");
            price = int.Parse(Console.ReadLine());
            if (price <= 0) // Stop criteria.
                break;      // Leave the while loop.
            total += price;
        }
        Console.WriteLine("Total = {0}", total);
...
```

Equivalence: while and for Loops

If the number of repetitions is known in advance a for loop may be used; otherwise, a while loop is preferred.

- One can always transform for loops to while loops, and versa.

Example: Compounding

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

- Let *balance* be the initial amount, *goal* be the goal amount, and *r* be the annual interest rate (%).
- We may use the compounding formula

$$balance = balance \times (1 + r \,/\, 100).$$

- Then output the number of holding years with the final balance.

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

        int years = 0;
        while (balance < goal)
        {
            balance *= (1 + r / 100);
            years++;
        }

        Console.WriteLine("Holding years = {0}", years);
        Console.WriteLine("Balance = {0, 5:F2}", balance);
...
```

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

```
...
        int years = 0; // Should be declared here; scope issue.
        for (; balance < goal; years++)
            balance *= (1 + r / 100);
...
```

```
...
        int years = 1; // Why?
        for (; ; years++)
        {
            balance *= (1 + r / 100);
            if (balance > goal) break;
        }
...
```

- Leaving the condition (the middle statement) blank assumes true.

Nested Loops by Example

Write a program to show a 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

```
...
    static void Main(string[] args)
    {
        for (int i = 1; i <= 9; ++i)
        {
            // In row i, output each j.
            for (int j = 1; j <= 9; ++j)
                Console.Write("{0, 3}", i * j); // Not WriteLine!
            Console.WriteLine();
        }
    }
...
```

- 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.

Example: Triangles

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

```
...
        // Case (a)
        for (int i = 1; i <= 5; i++)
        {
            for (int j = 1; j <= i; j++)
                Console.Write("*");
            Console.WriteLine();
        }

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

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

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

...
```

Exercise: Pythagorean Triples[7]

- Let $a < b < c \leq 20$ be three distinct positive integers.
- Write a program to find all triples satisfied with $a^2 + b^2 = c^2$.

```
...
        for (int a = 1; a <= 20; a++)
            for (int b = a + 1; b <= 20; b++)
                for (int c = b + 1; c <= 20; c++)
                    if (a * a + b * b == c * c)
                        Console.WriteLine("{0} {1} {2}", a, b, c);
...
```

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

Analysis of Algorithms

- There may exist various algorithms for the same problem.
- We then compare these algorithms by measuring their efficiency.
- To do so, we estimate the growth rate of running time in function of input size n.
- We proceed to introduce the notion of time complexity.[8]
- Similar to time complexity, we later turn to the notion of space complexity.

[8] Also see https://en.wikipedia.org/wiki/Time_complexity.

Example: SUM

```
...
        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 positive number.
- Recall that all declarations are finished in compile time.
- Hence we don't count them in the calculation.
- The number of total operations is $4n + 3$.

Exercise: TRIANGLE

```
...
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= i; j++)
                Console.Write("*");
            Console.WriteLine();
        }
...
```

- I think, before counting, it may be $cn^2 + \cdots$ with some c.
- What is the number of operations? (Try.)

Big-O Notation[9]

- We define

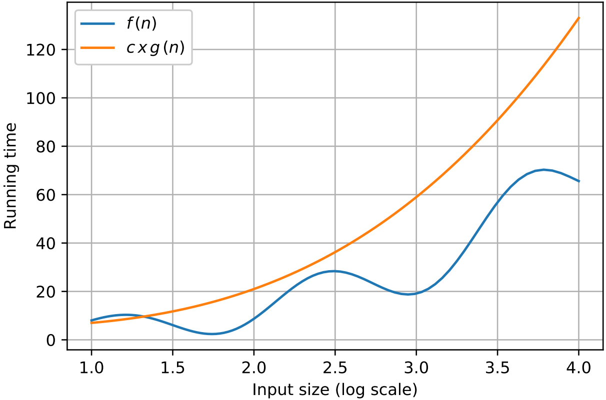
$$f(n) \in O(g(n)) \text{ as } n \to \infty$$

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

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

- Note that $f(n) \in O(g(n))$ is equivalent to say that $f(n)$ is one instance of $O(g(n))$.

[9] See `https://en.wikipedia.org/wiki/Big_O_notation`.



- $f(n) \in O(g(n))$ indicates the asymptotic upper bound of $f(n)$.
- In other words, big-O describes the worst case of this algorithm.

Discussions (1/3)

- For example, consider $8n^2 - 3n + 4$.
- For n large enough, ignore the last two terms. (Why?)
- It is easy to find a constant $c > 0$, say $c = 9$.
- So we have $8n^2 - 3n + 4 \in O(n^2)$.
- A shortcut to identify the order of time complexity is as follows:
  - Keep the leading term only.
  - Drop the coefficient.
- See? $8n^2 - 3n + 4 \in O(n^2)$.

Discussions (2/3)

- Can you determine the order of time complexity for the previous two examples?
  - SUM: $O(n)$.
  - TRIANGLE: $O(n^2)$.
- As a thumb rule, k-level loops run in $O(n^k)$ time.

Which Algorithm 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 (3/3)

- It is worth to note that

$$8n^2 - 3n + 4 \notin O(n),$$

and

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

- However, we should say that $8n^2 - 3n + 4 \in O(n^2)$ when it comes to classification of algorithms. (Why?)

Orders of Growth Rates

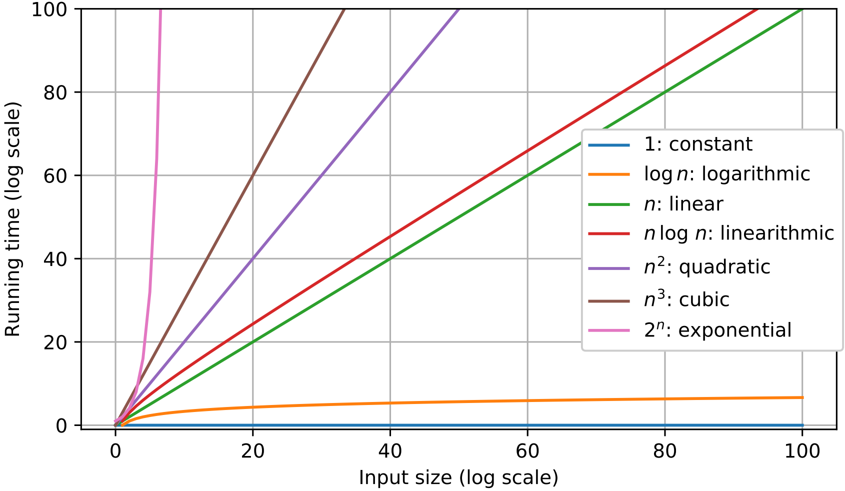


Table of Big-O

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 run in $O(1)$ time. (Why?)
- However, not every single statement runs in $O(1)$ time.
  - For example, calling **Array**.Sort() does not mean that sorting is cheap.
- Some algorithms also run in $O(1)$ time, for example, the arithmetic formulas. (Why?)
- However, there is no free lunch.
- A trade-off between generality and efficiency should be made to strike a balance.

Exponential-Time Algorithms & Computability

- We are actually overwhelmed by lots of intractable problems.
  - For example, the travelling salesman problem (TSP).[10]
- Playing game well is even hard.[11]
  - Check out AlphaGo and AlphaStar.[12]
- Moreover, there exist problems which cannot be solved by computers.
  - Turing (1936) proved the first unsolvable problem, called the halting problem.[13]

---

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

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

[12] See https://en.wikipedia.org/wiki/AlphaGo and https://deepmind.com/blog/article/AlphaStar-Grandmaster-level-in-StarCraft-II-using-multi-agent-reinforc

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

Logarithmic-Time Algorithms

- We have learned one of logarithmic-time algorithms. (Which?)

Outstanding Theoretical Problem[15]

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

- 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 solved in exponential time."
    - For example, id verification is easier than hacking an account.
- One could say that $\mathbb{P}$ is easier than $\mathbb{NP}$.
- $\mathbb{P} \stackrel{?}{=} \mathbb{NP}$ is asking if $\mathbb{NP}$ is solved by $\mathbb{P}$.
- We don't have any rigorous proof yet.
- It is also one of the Millennium Prize Problems.[14]

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

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