**Programming Labs**

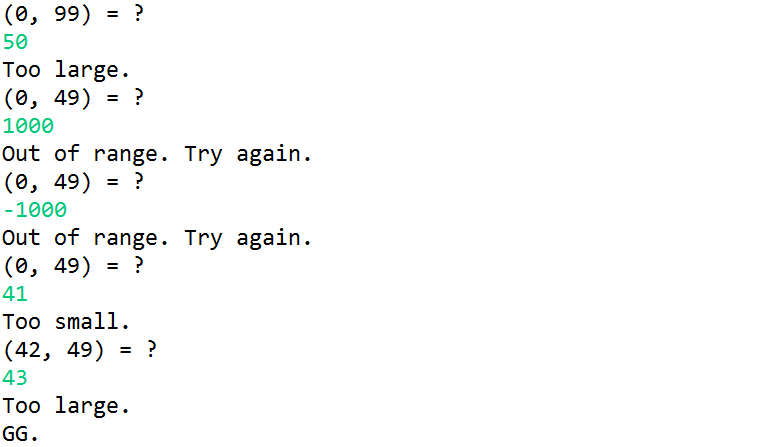
**Grading Policy** Since Java 310, I've canceled the final exam for your final grade. As the minimal requirement to acquire the course certificate, you need to finish all labs w/o OPTIONAL. I suggest that you could try optional labs as many as possible.

**Homework Submission** Send the source code (the file with .java extension) to my email shown in the beginning of this page. Remember to indicate this class number, your full Chinese name, and homework number in your email title. For example, [Java311] Homework 1 盧政良. Do not attach any class file because any executable file will be ruled out by Gmail.

* Lab 0

Install [JDK8](https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html) and [Eclipse](https://www.eclipse.org/downloads/) (or any IDE you have been familiar with, say [IntelliJ](https://www.jetbrains.com/idea/)) in your computer (desktop/laptop). You don't need to send me email for this lab unless you cannot complete the installation.

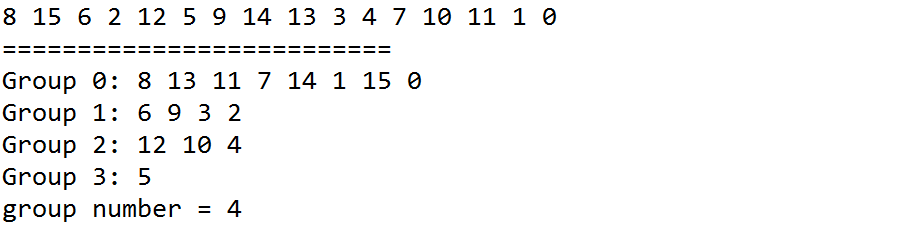
* Lab 1

Write a program for number-guessing game (you may refer to [Guess the Number](https://www.funbrain.com/games/guess-the-number) if you have never played this game). The program first generates a secret number ranging between 0 and 99, inclusive. Then the program asks the player to guess a number. If the input value is equal to the secret number, then the player wins. If not, then update the range depending on the input accordingly. (For example, assume that the secret number is 42. If the player types 50 for the first time, then the program shows (0, 49) on the screen.) When there is only one integer left, the player loses the game. Also, make sure that the player types a number in the feasible range; otherwise, ask the user to redo the input. The figure shown below illustrates the case you lose.  
  


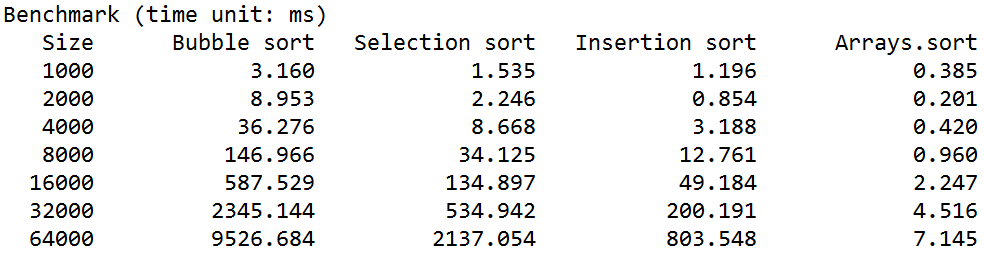
* + Lab 1-1 (optional)

Write an AI program which plays this game. For example, the naive strategy is to guess a number in random. Also, calculate the winning rate by simulating your AI program for 1e5 times.

* Lab 2

Read the problem statement in p.7 of the slides: [程式實作題](https://www.csie.ntu.edu.tw/~d00922011/cpp/309/%E5%AF%A6%E4%BD%9C%E9%A1%8C.pdf) for APCS. Let N be the number of students, denoted by 0, 1, 2, ..., N - 1. First, implement an algorithm which generates a sequence of 0, 1, 2, ..., N - 1 in random order. Note that all elements in sequence are distinct. Then write an algorithm to determine the number of groups among students and also report the members of each group. For example, N = 16.  
  
Keyword: permutation index.

* Lab 3

First, implement three sorting algorithms (bubble sort, insertion sort, and selection sort). You then compare the efficiency of these three sorting algorithms together with Array.sort (check its API) by running Monte Carlo simulation for various sizes and benchmarking the time cost between algorithms (doubling hypothesis), shown as below.  
  
You may use System.currentTimeMillis() or System.nanoTime() to make timestamps (remember to use long variables). Note that the actual numbers in the table may differ from mine due to various hardware and software environments.

* + Lab 3-1 (optional)

Implement Shell's sorting algorithm which runs in O(n (log n)^2) time. You may refer to [shell sort](https://en.wikipedia.org/wiki/Shellsort) in Wikipedia.

* + Lab 3-2 (optional)

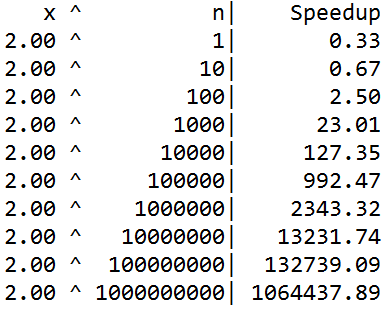
Implement the merge sort algorithm which runs in O(n log n) time. You may refer to [merge sort](https://en.wikipedia.org/wiki/Merge_sort) in Wikipedia.

* Lab 4

Let x be any real number and n be any positive integer. Write a program which calculates x ^ n by recursion. For example, 2 ^ 10 = 1024. Try to make your program run in O(log n) time. Note that you are not allowed to use Math.pow and any loop in your solution.  
Keyword: exponentiation by squaring.

* + Lab 4-1 (optional)

Compare your solution to the naive method (running in O(n) time). Report the speedup.



* + Lab 4-2 (optional)

Modify your solution for negative n.

* + Lab 4-3 (optional)

Convert the recursion version into the loop version.

* + Lab 4-4 (optional)

Can we do better? Introduce bitwise operators (&) and shift operators (>>) to your solution.

* + Lab 4-5 (optional)

Let M be any positive int value. Write a function to calculate x ^ n mod M for n = 10000000. You may use the polynomial remainder theorem (see [here](https://en.wikipedia.org/wiki/Polynomial_remainder_theorem)).

* Lab 5

Rewrite the program in Lab 1 in object-oriented programming. You need to analyze the functionalities in Lab 1. You may define any number of classes if necessary. For example, you can consider two objects in this system: one for the game procedure, the other for the players (in order to change the strategies in the future). In this way, you should define a protocol so that the game object has minimal dependency on the player. If so, you can easily develop various strategies to play this game and see which strategy is more promising.  
Keyword: OOP, class inheritance, method overriding, subtype polymorphism, dependency injection.

* + Lab 5-1 (optional)

Try various strategies for this game. For example, the naive strategy is to guess a number in random, and the binary-search strategy is to return the number in the middle of the feasible range. Then compare the winning rates by simulating these strategies for 1e5 times.

* Lab 6 (optional)

The following problems interest me.

* + Lab 6-1 (optional)

Write a program which estimates the Euler constant e ~ 2.7183 by Monte Carlo simulation. The procedure is as follows: (1) Set N = 1e5 and M = 0; (2) for each iteration, find the minimal number k of random numbers which follows a standard uniform distribution (simply use Math.random() in this case) so that the sum of these random numbers is greater than 1; (3) do M += k; (4) then the estimator of e can be done by M / N.  
Keyword: Monte Carlo, Euler constant.

* + Lab 6-2 (optional)

Write a program which calculates the price of European call options. The input is as follows: (1) S (spot price), (2) X (strike price), (3) T (time to maturity, year), (4) v (annual volatility), (5) r (annual interest rate). The price of European call options can be done as follows: (1) Set N = 1e5 and c = 0; (2) for each iteration, calculate the asset price S' by S' = S \* exp((r - 0.5 \* v ^ 2) \* T + v \* sqrt(T) \* RANDN), where RANDN is a random variable following a standard normal distribution (use the method nextGaussian() of Random objects; see [Random](https://docs.oracle.com/javase/8/docs/api/java/util/Random.html) from Oracle); (3) if S' - X > 0, do c += S' - X; (4) after N iterations, do c /= N. Finally, output the option price c. For example, the European call price is 14.2xxx for the case of S = 100, X = 100, T = 1, v = 0.3, r = 0.05.  
Keyword: Monte Carlo, option price, Black-Scholes model, geometric Brownian motion, financial engineering.