CS1131 Notes

Computational Thinking I

Chapters 2 thru 12

This is a compilation of concepts taught in CS1131.
Made by LQ

who is very stressed and tired and depressed right now

Links of All LQ Notes: LQ Notes Links Document

Topic
Topic 1: Algorithms
Topic 2: Number Bases
Topic 3: Turtle
Topic 4: For Loops and Variables
Topic 5: If Statements
Topic 6: Excel

Disclaimer

References from NUS High CS1131 Coursemology are made. Hence, please ***do not*** share this set of notes outside of your NUS High Y1 Schoolmates.

Take everything in this set of notes with a pinch of salt as there is neither enough time nor manpower to check through the notes fully. The order of topics in this set of notes is not completely accurate to how they were taught in Labs, but should contain similar content.

If you spot any mistakes, please do inform me if you can. Thanks :)

All the best for your exams!

~LQ (24 Sep)

Topic 1: Algorithms

In CS1131, 2 main types of algorithms are covered - Linear Search, and Binary Search.

1.1 What is an algorithm?

An algorithm is a **step by step list of instructions that, if followed exactly, will solve the problem under consideration**.

An algorithm should be **exact** (i.e. the result produced should perfectly solve the problem), **general** (i.e. the algorithm should not fail to function for any specific input), and should **terminate** (i.e. the algorithm will eventually finish running). However, an algorithm isn't necessarily fast.

1.2 Linear Search

A linear search is simply searching for the value **by going through each possible value one by one**. Consider the following problem:

Problem

A computer chooses a number at random, between 1 and 10, inclusive. You get to input a number, and the computer will tell you if the number is correct or wrong. How should you obtain the correct number?

Solution

In this case, we can perform Linear Search.

We first check if the number is 1. If the computer says that it's wrong, we continue by checking 2, then 3, and as such until we reach the correct number.

This is Linear Search.

1.3 Binary Search

Binary Search works on a problem where you know if your guess is too high, too low, or correct. We can **check the mid-value** of the possible range, and as such **reduce the range by half** accordingly. This sounds complicated at first, but it is better illustrated with an example and a diagram.

Consider the following problem:

Problem

A computer chooses a number at random, between 1 and 16, inclusive. You get to input a number, and the computer will tell you if the number is too high, too low, or correct. How should you obtain the correct number?

Solution

In this case, we can perform Binary Search.

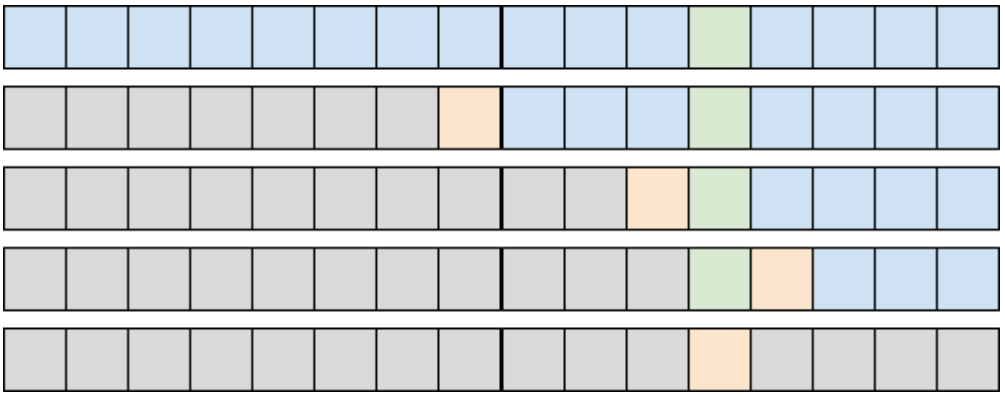
Let's assume the computer chose 13.

First, we check the mid-value of the range, 1 to 16. The mid-value is 8. The computer will say it's too low.

Now, the possible range is from 9 to 16. Hence, we check the new mid-value, 12. The computer then says it's still too low.

We then continue repeating this process, halving the possible range each step, until we reach the answer.

This is what it will look like, illustrated:



Blue refers to the possible values, green is the correct value, and orange is the value we're guessing currently.

Note that Linear Search could still be applied here, but it will take much longer, on average.

This is Binary Search.

Topic 2: Number Bases

Bases? Like… Acids and Bases?
No, not that. Number Bases are essentially different ways to represent a number.

2.1 Decimal (Base-10)

The base system we use on a day-to-day basis is called the **decimal** number system. Since deci means 10, it means that the decimal system is made of 10 digits (i.e. from 0 to 9). We also call the decimal number system the **base-10** number system.

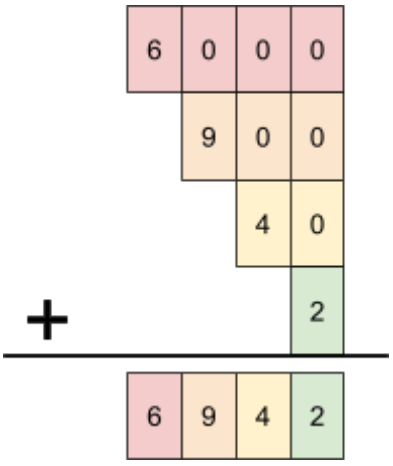
Hence, we denote numbers written in base-10 with a subscript "10" when dealing with multiple types of bases, to avoid confusion. (e.g. 63_{10})

2.2 Binary (Base-2)

The Binary number system, or base-2 number system, only has 2 digits, 0 and 1. This is the system computers use to store data, and is what allows most, if not all electronics to work.

To figure out what binary numbers mean, we first need to know how decimal numbers work.

As shown below, we can break down a number like 6942 into a single digit, multiplied by a power of 10.



This is how binary numbers work, too, but instead of powers of 10, it uses powers of 2 (Hence, base-2).

In decimal, 1000 represents 10^3, since there are 3 trailing zeros. However, in binary, 1000 will represent 2^3, or 8.

In this similar fashion, we can construct the following table:

Binary	Decimal

1	1
10	2
100	4
1000	8
10000	16
100000	32
1000000	64
10^n	2^n

Now, let's try translating 101_2 into decimal.

Note: we denote binary numbers with a subscript "2", like 101_2, to avoid confusion.

Solution:

$$\begin{aligned} 101_2 &= 100_2 + 1_2 \\ &= 4_{10} + 1_{10} \\ &= 5_{10} \end{aligned}$$

To convert a decimal number into binary, we do the same thing, but in reverse. We break down the number into its powers of 2, Take the example 69_{10}.

Solution:

$$\begin{aligned} 69_{10} &= 64_{10} + 4_{10} + 1_{10} \\ &= 1000000_2 + 100_2 + 1_2 \\ &= 1000101_2 \end{aligned}$$

2.3 Octal (Base-8)

The octal base system, or base-8 system, consists of the digits 0 to 7. It functions on a similar logic to how base-2 and base-10 systems work.

Recall how in base-2, each new place value is a new power of 2. In base-8, it would be a new power of 8 instead.

Octal	Decimal

1	1
10	8
100	8^2
1000	8^3
10^n	8^n

It is worth noting that if you have something like 30_8, it simply means $3_8 \times 10_8$. (which is equivalent to $3_{10} \times 10_8$)

Let's try translating 514_8 into decimal.

Solution:

$$\begin{aligned} 514_8 &= 5_8 \times 100_8 + 1_8 \times 10_8 + 4_8 \times 1_8 \\ &= 5_{10} \times 64_{10} + 1_{10} \times 8_{10} + 4_{10} \times 1_{10} \\ &= 320_{10} + 8_{10} + 4_{10} \\ &= 332_{10} \end{aligned}$$

2.3 Hexadecimal (Base-16)

The hexadecimal base system, or base-16 system, consists of the digits 0 to 9, as well as the letters A to F. It functions on a similar logic to how base-2 and base-10 systems work.

Hexadecimal	Decimal
A	10
B	11
C	12
D	13
E	14
F	15

Again, recall how in base-2, each new place value is a new power of 2. In base-16, it would be a new power of 16 instead. Everything else follows the same logic.

Topic 3: Turtle

(hmph mewos are better than turtles)

Turtle is a commonly used (and very useful) python **library**. It contains code written by someone else that you can use to draw certain graphics.

3.1 Basic Turtle Functions

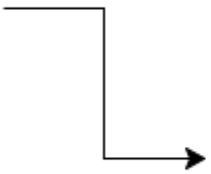
Here are a few basic turtle functions.

Before coding, we should always write `import turtle` to tell python that we are, in fact, using the turtle library.

To make a turtle move forward, we use `turtle.forward()`. We put the number of pixels we want the turtle to move forward inside the brackets. So, if you want the turtle to move 30 pixels, you write `turtle.forward(30)`.

To make a turtle turn left, we write `turtle.left(45)`, again with the angle to turn inside the brackets. In the example above, the turtle will turn left by 45 degrees. Similarly, `turtle.right()` is used in the same way.

Let's try to determine the code used to generate this shape.



The turtle moved forward by 50 pixels, then turned right by 90 degrees, then forward by 75 pixels, and left by 90 degrees, and then forward by 50 pixels again.

We can now represent this in code:

```
turtle.forward(50)
turtle.right(90)
turtle.forward(75)
turtle.left(90)
turtle.forward(50)
```

There are also other functions, as such:

`turtle.stamp()` This function does not require any input, and just stamps the turtle's shape onto the screen.
`turtle.goto(x, y)` This function sends the turtle to (x, y). Note that the center of the screen is (0, 0), and as you move left, the x value decreases. As you move right, the x value increases. As you move up, the y value increases. Lastly, as you move down, the y value decreases.
`turtle.shape("shape")` This function changes the turtle's shape. "shape" can be replaced with "square", "triangle", "circle", or even just "turtle". Note that the double-quotation marks are required.
`turtle.color("color")` This function changes the turtle's color. In a question, a color will be given, which should look something like "#00ff00". Simply copy that into the brackets and note that the double-quotation marks are required.
`turtle.penup()` This function lifts up the turtle's pen, which means it won't leave a trail. No inputs required.
`turtle.pendown()`

This function puts down the turtle's pen, which means it will continue to leave a trail. No inputs required.

Topic 4: For Loops

Hey time check it's 11pm I've been writing for 2 hours

For Loops are essentially what you use in python to repeat a certain action multiple times.
It is very useful for shortening code and making our lives easier, as coders, in general.

4.1 Using For Loops to Draw a Triangle

For example, if we wanted to draw a triangle in turtle, without for loops, we would have to write:

```
turtle.forward(50)
turtle.left(120)
turtle.forward(50)
turtle.left(120)
turtle.forward(50)
turtle.left(120)
```

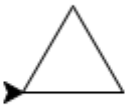
But that's quite repetitive and boring, isn't it? Instead, we can just write:

```
for count in range(3):
     turtle.forward(50)
     turtle.left(120)
```

The number, 3, in the brackets of range can just be thought as "repeat 3 times". We will look more into detail later in this topic.

Note that the name "count" can be replaced by any other name. Typically, we would use a short name like "n" or "i" for convenience, but you could use any name you want.

Both of these codes represent the same thing, but the second one is shorter and hence it is often preferred.



4.2 Generalisation

In general, if we want to draw a n-sided polygon, we use the following code:

```
n = int(input("Enter number of sides: "))
```

```
length = int(input("Enter length: "))

for count in range(n):
     turtle.forward(length)
     turtle.left(360 / n)
```

The first line takes an input from the user for the number of sides and converts it into an integer. Similarly, the second line does the same, for the length of each side.

Then, on line 4, we use a for loop to repeat the preceding action n times, once for each side of the polygon.

On line 5, we move the turtle forward by `length` pixels, then on line 6, we turn it by `360 / n` degrees *(this is actually one exterior angle of a n-sided polygon, mathematically)*

4.3 Variable

A variable is like a box that can store data.
We can define a variable `x` as such:

```
x = 10
```

This assigns the value, 10, to the variable, x, thereby defining it.

We can modify this value by reassigning a value, like:

```
x = 11
```

What if we want to add 1 to x, even if you don't know what x is? Well, simple! We can simply access the variable while re-defining, as such:

```
x = x + 1
```

Yes, this makes no mathematical sense, but this essentially means "take the value of x, add 1, then assign that to x".

Question

Consider the code below. What will be its output?

```
x = 10
x = x + 5
print(x)
```

Solution

Well, we start off with x being 10. Then, in line 2, we take x to be itself plus 5, which is 15. Thus, the output will be 15.

4.4 Operations in Python

Operation	Name	What does it do?
+	Addition	a + b means “add a and b”
-	Subtraction	a - b means “subtract b from a”
*	Multiplication	a * b means “multiply a and b”
/	Division	a / b means “divide a by b”
//	Integer Division	a // b means “divide a by b, then round down to the nearest integer”
%	Modulo	a % b means “find the remainder of a divided by b”
**	Exponentiation	a ** b means “raise a to the b-th power”

Question

Consider the following code. What is the value of x?

```
x = (5 % 3) ** (7 // 2)
```

Solution

We first evaluate 5 % 3, which is 2, since 5 leaves a remainder of 2 when divided by 3.

Then 7 // 2 is 7/2 rounded down, which is 3.5 rounded down, or 3.

Hence, x is equal to 2 to the power of 3, or **8**.

4.4 More In-depth Use of For Loops

There are 3 types of uses we need to know in CS1131 of for loops.

4.4.1 Range with 1 input

If you see something like:

```
for n in range(10):
      ...
```

This will cause n to loop through the values from 0 to 10 minus 1, or 9.

So, if you print out each value of n, as such:

```
for n in range(10):
      print(n)
```

The code will print:

```
0
1
2
3
4
5
6
7
8
9
```

In short, the final value of n is one less than the number put into the brackets.

4.4.2 Range with 2 inputs

We can fit more than one value into range, to better describe a certain range. For example, let's say you want n to loop through the values 1 to 11.

This can be achieved with:

```
for n in range(1, 12):
      ...
```

In this case, the first number represents the starting number of the for loop. Then, the second number is the stopping number of the for loop. However, recall that the final value of n isn't actually the stopping number, but instead the stopping number minus 1.

Hence, if we want the values of n from 1 to 11, the stopping number must be 11 + 1 = 12.

4.4.2 Range with 3 inputs

Ok, now what if we want to print all multiples of 3 between 3 and 27, inclusive?

This can be achieved with:

```
for n in range(3, 28, 3):
      print(n)
```

In this case, the first number represents the starting number of the for loop. Then, the second number is the stopping number of the for loop. However, recall again that the final value of n isn't actually the stopping number, but instead the stopping number minus 1.

Then, the third value is called the **step**. It represents the number added to n everytime the loop repeats. Hence, the code above prints:

```
3
6
9
12
15
18
21
24
27
```

As you can see, the value goes from 3 to 3 + 3 = 6, then to 6 + 3 = 9 and so on. Hence, the step between each number is 3.

Topic 5: If Statements

Woo it's 11:37 pm as of writing ._. i'm dying sorry for trash quality

If statements are a way to check if a certain condition is met. For example, let's take a modified version of an example in Topic 1.

Problem

A computer chooses a number at random, stored in a variable called `ans`. You get to input a number, x, and the computer will tell you if the number is too high, too low, or correct. How should you code the computer to do so?

Solution

```
x = int(input("Enter x: "))
if x < ans:
     print("Too low!")
if x == ans:
     print("Correct!")
if x > ans:
     print("Too high!")
```

The code is pretty self-explanatory. However, I want to point out that the usage of a colon is **required**, or else an error will be thrown.

Also, note the equality and inequality signs, as explained below.

5.1 Equality and Inequality signs

Here is a table of equality and inequality signs you can use in an if statement.

Sign	Meaning
>	Larger than
<	Less than
>=	Larger than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

5.2 If-Else

An else clause can be added after an if statement, and that else clause will run if and only if the condition in the if statement fails.

```
if 1 == 0:
      print(“WTF???”)
else:
      print(“Ok yea makes sense”)
```

In the code above, we check if 1 is equal to 0, which is false. Hence the else statement runs, and the code prints “Ok yea makes sense”.

Topic 6: Random

AHHHHH ITS ALMOST 12

Random is a python library, which, as shown by its name, produces random numbers. To use the library, we first write "`import random`" at the start of the code.

There are 2 main functions in random you should know.

6.1 random.randrange

`random.randrange(a, b)` takes 2 inputs, a and b. It then produces a random number between a and b, inclusive of a, but exclusive of b, just like the range in a for loop, hence the name rand**range**.

6.2 random.random

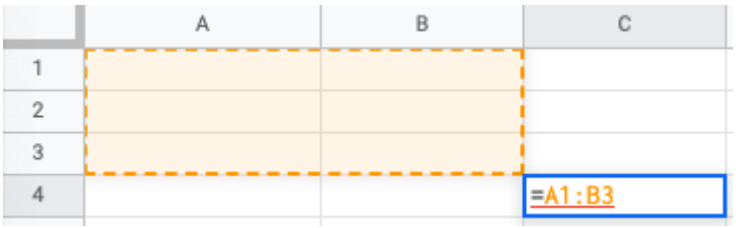
`random.random()` does not require any input, and will produce a random decimal number between 0 and 1.

Topic 7: Excel

7.1 Definitions

7.1.1 Range

A **range** in excel is like a **rectangle of cells**. A1:B3 refers to the collection of cells in the rectangle with opposite corners A1, B3; So A1:B3 refers to A1, A2, A3, B1, B2, B3.



7.1.2 Condition

A **condition** in excel can mean one of two things:

1. An inequality: using the symbols (>, <, >= or <=) like in "10 >= 5".
2. A template: using wildcards (*, ?) like in "m?w*" (See 7.2.1 - * and ?)

7.2 Wildcards and Absolute Reference

7.2.1 * and ?

The wildcard * represents **any number of characters** (can be 0 characters too).
The wildcard ? represents **exactly 1 character** (can be a space).

Wildcards can be **nullified** by using a tilde (~), if you want to search for those literal characters. For example, if you want to use FIND on a literal question mark, you have to write FIND("~?", …

7.2.2 Absolute Reference

Absolute Reference is used when you don't want something to be affected by autofill. For example, in the function VLOOKUP.

We can "lock" the cell in place by writing it like $A$1 (or $A1, or A$1, depending on situation) instead of just A1. Note that each dollar sign locks the respective row/column from autofill.

7.3 Excel Functions

Here are the common tested excel functions.

LEFT(*cell*, *num*): Gets the leftmost *num* characters in the cell *cell*. LEFT(“abcdef”, 4) returns “abcd”.
RIGHT(*cell*, *num*): Gets the rightmost *num* characters in the cell *cell*. RIGHT(“abcdef”, 4) returns “cdef”.
MID(*cell*, *start*, *num*): Gets the first *num* characters starting from the *start*-th character in the cell *cell*. MID(“abcdef”, 2, 3) returns “bcd”.
UPPER(*text*): Returns all letters in *text* in uppercase. UPPER("a bAnaNA") returns "A BANANA"
LOWER(*text*): Returns all letters in *text* in lowercase. LOWER("a bAnaNA") returns "a banana"
PROPER(*text*): For every word in *text*, return the first letter in uppercase and the rest in lowercase. PROPER("a baNaNA") returns "A Banana"
LEN(*text*): Returns the length of *text*. LEN("banana man!") returns 11
FIND(*findtext*, *text*, [*start*]): Returns the position of the first occurrence of *text* in cell *cell* starting from *start*-th character. *start* is optional, so if left blank, the function returns the first occurrence. FIND("g", "mingyang") returns 4, since "g" is the fourth character FIND("g", "mingyang", 4) returns 4, since the first "g" starting from the fourth character is the fourth character itself. FIND("g", "mingyang", 5) returns 8, since the first "g" starting from the fifth character is the eighth character.
LARGE(*range*, *num*) returns the *num*-th largest number in the range *range*.

If the range A1:A5 corresponds to the values 1, 3, 5, 7, 9,

LARGE(A1:A5, 2) returns 7, as 7 is the second largest number.

SMALL(*range*, *num*) returns the *num*-th smallest number in the range *range*.

If the range A1:A5 corresponds to the values 1, 3, 5, 7, 9,

SMALL(A1:A5, 2) returns 3, as 3 is the second smallest number.

MAX(*range*) returns the largest number in the range *range*.

If the range A1:A5 corresponds to the values 1, 3, 5, 7, 9,

MAX(A1:A5) returns 9, as 9 is the largest number.

MIN(*range*) returns the smallest number in the range *range*.

If the range A1:A5 corresponds to the values 1, 3, 5, 7, 9,

MIN(A1:A5) returns 1, as 1 is the smallest number.

AVERAGE(*range*) returns the average of the numbers in the range *range*.

If the range A1:A5 corresponds to the values 1, 3, 5, 7, 9,

AVERAGE(A1:A5) returns 5, as 5 is the average of 1, 3, 5, 7 and 9.

MEDIAN(*range*) returns the median of the numbers in the range *range*.

The median of a set of numbers is obtained in the following way:

1. Arrange the numbers in ascending/descending order
2. If the number of items is odd, take the middle number as the median.
   If the number of items is even, take the average of the two middle numbers as the median.

Take the example MEDIAN(1, 2, 8, 7, 2, 9).

We rearrange the numbers in ascending order, like 1, 2, 2, 7, 8, 9.

Since there is an even number of items, we take the two values in the middle, 2 and 7, and we take the average. Hence MEDIAN(1, 2, 8, 7, 2, 9) is 4.5.

MODE(*range*) returns the mode of the numbers in the range *range*.

The mode of a set of numbers is just the number(s) that occur most frequently.
If multiple modes exist, the first mode is returned by excel.

E.g. MODE(3, 5, 7, 2, 5, 2, 2, 5) returns 5 because 5 appeared the most, and appeared earliest.

IF(*condition*, *value1*, *value2*) returns *value1* if *condition* is true, else it returns *value2*.
condition can be statements like "5 >= C1" or "6 < 10". Refer to 7.1.2 - Condition.

IF("6 < 9", 3, 4) returns 3, since the condition 6 < 9 is true.

COUNT(*range*) counts the number of **non-empty** cells in *range*.

If the range A1:A5 corresponds to the values 1, 3, 5, 7, 9,

COUNT(A1:A5) returns 5, as all 5 cells are non-empty.

COUNTIF(*range*, *condition*) returns the number of cells in *range* that satisfy the condition *condition*.

If the range A1:A5 corresponds to the values 1, 8, 5, 2, 9,

COUNTIF(A1:A5, “>3”) returns 3, as 8, 5, and 9 satisfy this condition.

COUNTIFS(*range1*, *condition1*, [*range2*], [*condition2*], ...) returns the number of cells in *range1* that satisfy the condition *condition1*, and the corresponding cell in *range2* satisfy *condition2*, and so on.

There must be at least 1 range and 1 condition, but for each range, there must be an accompanying condition, even if the same range is used.

If the range A1:A5 corresponds to the values 1, 8, 5, 2, 9,

COUNTIFS(A1:A5, “>3”, A1:A5, “<7”) returns 1, as only 5 lies between the values 3 and 7.

SUM(*range*) returns the sum of all numbers in the range *range*.

SUM(1, 4, 2, 3) returns 10.

SUMIF(*range*, *condition*) returns the sum of all numbers in the range *range* that satisfy *condition*.

If the range A1:A5 corresponds to the values 1, 8, 5, 2, 9,

SUMIF(A1:A5, ">2") returns 22, as that's the sum of 5, 8 and 9, which all satisfy the condition.

SUMIFS(*sumrange*, *conditionrange1*, *condition1*, *]conditionrange2]*, *[condition2]*, ...) sums all numbers in *sumrange* where the corresponding number in *conditionrange1* satisfies *condition1*, **AND** the corresponding number in *conditionrange2* satisfies *condition2* **AND** so on...

VLOOKUP(*key*, *tablerange*, *column*, FALSE) finds the corresponding value in *tablerange* for the value *key*. The number *column* is the column number that the values sit in the table *tablerange*.

Reference Table	
101	BEST CLASS
102	ALSO GOOD
103	WTF
104	YES GOOD
105	IT EXISTS?
106	YEYY ACCELL

For example, take the example above. The table is situated A2:B7.

VLOOKUP("101", $A$2:$B$7, 2, FALSE) returns "BEST CLASS".

Note we always use "FALSE" and 2 refers to the **second column** in the table. Also, the dollar signs (See 7.2.2 - Absolute Reference) are necessary in this case, to make sure this table stays in place when autofill is applied.

HLOOKUP(*key*, *tablerange*, *row*, FALSE) finds the corresponding value in *tablerange* for the value *key*. The number *row* is the row number that the values sit in the table *tablerange*.

References

NUS High Coursemology, CS1131, Lab 3, Q2
NUS High Coursemology, CS1131, Lab 3, Q3
LQ's exhausted brain

Author's Note

Yea these notes were kinda useless innit… Why did I spend 3 hours on this its now 12.01am

Good luck on your Lab Tests <3

~A very dead LQ (25 Sep now)