Running Time Analysis.

Time analysis consists of ~~reasing~~ reasoning about an algorithm's speed.

Main Concerns:

* How much longer does the method take when the input gets larger?
* Which of several different methods is fastest?

Understanding through an example

"The stair-counting problem"

While standing on the top of a tower, two persons are involved to count the stairs to the bottom.

Method # 1: "walk down and keep a tally"

While holding a pen and paper, each time you take a step down, you make a mark on the sheet of paper.
When you reach the bottom, you run back up, and tell the other person
"There are this many steps".

Compiled by M. Azam Ali

Method #2

"walk down, but let your friend keep the tally".

You stop after one step, lay your hat on the step and run back to your friend and make a mark on the sheet of paper.

Then you run back to your hat, pick it up, take one more step and lay the hat down on the Second step. Then run back to your friend:
"Make another mark on the sheet of paper".

This continues until your hat reaches the bottom and speed back up the steps one more time.
"Make the one last mark on the paper".

At this point, you grab the piece of paper and say
"There are this many steps".

Method # 3 : In the third method, you don't walk down the stairs at all. Instead you spot a sign by the staircase saying ... ② ③

> There are
> 2689
> steps in this ~~staircase~~ way

You take the pen and paper from your friend and write
"There are 2689 steps".

The first issue is to ~~decide exactly~~ calculate
"how much time is spent carrying out each method".

This can be done with the help of a stopwatch but there are some drawbacks to measuring actual time.

- Actual time may vary from person to person.
- " " " " if the ~~f:~~ same person executes the method under different physical conditions.

So, Instead of measuring the actual elapsed time during each method,

"we count certain operations that occur while carrying out the methods".

Two types of operations can be counted in the above example:

1. Each time you walk up or down one step, that is one operation.
2. Each time you or your friend marks a symbol on the paper, that is also one operation.

To measure the efficiency of each method, we will ask the question.

"How many operations are needed for each of the three methods?"

In the First method: you take 2689 steps down, another 2689 steps up and mark 2689 marks for a total of 8067 operations.

In the Second method:

You mark 2689 marks

You start by going down one step and
back up one step
Then down two
and up two
Then down three
and up three and so forth.

The total # of operations taken is

5

Downward steps	= 1+2+3+4+.....+2689	= 3,616,705
Upward steps	= " "	= 3,616,705
Marks made		= 2,689
Total operations		= 7,236,099

The Third Method is the quickest of all

Only four marks are made on the paper. (i.e four digits) 2689

So the total # of operations = 4.

Operation: There is no precise definition of what constitues an operation, although an operation should satisfy your intuition of a small step.

When the method's time depends on the size of the input, then the time can be given as an expression, where part of the expression is the input's size. The time expressions for the above three methods

Method 1 $\longrightarrow$ $3n$

" 2 $\longrightarrow$ $n + 2(1+2+3+\cdots+n) = n^2+2n$

" 3 $\longrightarrow$ The # of digits in the number $= [\log_{10} n] + 1$

The expression on the right give the # of operations performed by each method when the stairway has n steps.

The expression on the right side of method 2 needs to be simplified.

$$1+2+3+\cdots\cdots+n-1+n$$

<u>Trick</u>: "Compute twice the amount of the expression and then divide the result by 2".

$$\begin{array}{r} 1 + 2 + \cdots\cdots + n \\ + \quad n + n-1 + \cdots\cdot + 2 + 1 \\ \hline (n+1) + (n+1) + \cdots\cdot + (n+1) + (n+1) \end{array}$$

$$= n(n+1)$$

OR

$$2(1+2+3+\cdots\cdots+n) = n(n+1)$$

$$1+2+3+\cdots\cdots+n = \frac{n(n+1)}{2}$$

<u>The Simplification for method 2 is</u>

Number of operations for method 2

$$= n + 2(1+2+\cdots\cdots+n)$$

$$= n + 2\left[\frac{n(n+1)}{2}\right]$$

$$= n + n(n+1)$$

$$= \cancel{n} + n^2 + n$$

$$= \boxed{n^2 + 2n}$$

So method 2 requires n^2+2n operations.

Simplification of Method 3

"The number of digits in a number n, when written in base 10 notation, is approximately equal to another mathematical quantity known as the <u>base 10 logarithm</u> of n.

$$\log_{10} n$$

The base 10 logarithm does not always give a whole number. e.g. $\log_{10}(2689) = 3.43$ rather than 4.

The exact # of digits in a positive number n is obtained by rounding $\log_{10} n$ downward to the next whole number and then adding 1.

i.e.

$$[\log_{10} n] + 1$$

3.43 rounded downward => 3

$3 + 1 = 4$

The final table of # of operations

Method 1 ⟶ $3n$

Method 2 ⟶ $n^2 + 2n$

Method 3 ⟶ $[\log_{10} n] + 1$

Big-O notation.

The time analysis we gave for the three stair-counting methods were very precise. They computed the exact # of operations for each method. But such precision is sometimes not needed.

"Often it is enough to know in a rough manner, how the number of operations is affected by the input size".

Suppose that we apply our various stair counting methods to a tower with 10 times as many ~~as~~ steps as that tower.

If n is the # of steps for that tower (i.e. the tower with 2689 steps) then this taller tower will have $10n$ steps.

The # of operations needed for Method 1 on the taller tower increases tenfold. (from $3n$ to $3 \times (10n) = 30n$.

The time for Method 2 increases 100-fold (from n^2 to about $(10n)^2 = 100n^2$

and Method 3 increases by only one operation. (from # of digits in to # of digits in $10n$)

or " 2689 to 26890.

4 digits to 5 digits.

Representing the information in Big-Oh notation.

3 common examples of the big-oh notation

① **Quadratic time:** $O(n^2)$

If the largest term in a formula is no more than a constant times n^2, then the algorithm is said to be

"big-O of n^2" written $O(n^2)$

and the algorithm is called quadratic.

In a quadratic algorithm, doubling the input size makes the # of operations increase by approximately four fold (or less).

e.g. Method 2 n^2+2n operations

For a 100 step Tower $(100)^2+2(100) = 10,200$ operations

Double the size to 200 steps $(200)^2+2(200) = 40,400$

$10,200 \times 4 = 40800$

approximately fourfold.

② Linear Time $O(n)$

If the largest term in the formula is a constant times n, then the algorithm is said to be "big-Oh of n" written $O(n)$ and the algorithm is called Linear.

10

"In a linear algorithm, doubling the input size makes the time increase by approximately twofold (or less)".

eg. $3n$

For a 100 step tower $3(100) = 300$

Double the size to 200 $3(200) = \underline{600}$

Twice of 300.

Another example

If formula is $3n+7$

$3(100)+7 = 307$

$3(200)+7 = \underline{607}$ Approximately double of 307 but less than exact double of 307 i.e $\underline{614}$

3) Logarithmic time:

If the largest term in the formula is a constant times a logarithm of n, then the algorithm is

"big-oh of the ~~a~~logarithm of n"

written as $O(\log n)$

and the algorithm is called Logarithmic.

"In a logarithmic algorithm, doubling the input size will make the time increase by no more than a fixed number of new operations, such as one more operation, two more operations — or in general by c more operations, where c is a fixed constant.

In Method 3, if size is doubled from 500 to 1000 requires only one extra operation.

Using Big-O notation

Method 1 $\rightarrow 3n \rightarrow O(n)$
Method 2 $\rightarrow n^2+2n \rightarrow O(n^2)$
Method 3 $\rightarrow [\log_{10} n]+1 \rightarrow O(\log n)$