



Competitive Programming

From Problem 2 Solution in $O(1)$

Elementary Math

Introduction

Mostafa Saad Ibrahim

PhD Student @ Simon Fraser University



Mathematics and CS

- Directly
 - Many CS components need math
 - Discrete Mathematics is critical area (E.g. Graph Theory)
 - Machine Learning needs Algebra, Calculus, Probability..
 - 3-D motion and graphics...Robot Trajectory...etc
- Indirectly
 - Builds logical/critical thinking skills
 - Thinking abstractly and concretely
 - Problem solving skills
- Commercial apps?
 - Most of them don't need math.

Mathematics and CP

- In competitive programming (CP), Math is an important topic.
- Many problems need **basic high school** skills
- Others may focus on **Discrete Mathematics**
 - Graph theory is the most important field
 - Number theory & Combinatorics are nice frequent topics
 - Little Geometry, Probability and Game Theory
- Problem Setters may avoid geometry and probability problems, due to output **precision** problems

#include<cmath> in C++

- C++ offers for us some ready-made functions
- Trigonometric, Hyperbolic, Exponential , Logarithmic, Power, Rounding, Remainder
- Please, play with Majority of these functions
 - **At least:** floor, ceil, round, fabs, sqrt, exp, log, log2, log10, pow, cos, cosh, acosh, isnan
 - We will explore some of them
- Other languages should also have similar functions

Machine Arithmetic

- $+$ $-$ $*$ $/$ are the usual arithmetic operations
- 32 bit numbers are suitable most of time
 - -2147483648 to 2147483647
 - Short fact: You have up to 2 billions
- 64 bit numbers can cover much wider range
 - 9,223,372,036,854,775,808 ($9 * 10^{18}$)
 - This is too big and fit **most** of time for your purpose
 - But slower (8 bytes vs 4), so use it only if needed
- Still we can face over/underflow
 - In intermediate computations or final results
- doubles range: $1.7E \pm 308$ (15 digits)

Real Numbers

- **Rational** Numbers: can be represented as **fraction**: $\frac{1}{6}$, $\frac{7}{2}$, $\frac{9}{3}$, $\frac{5}{1}$. **Irrational**: $\pi = 3.1415926$, $\sqrt{2}$
- **Decimal expansion** of fraction, to write it
 - $\frac{1}{16} = 0.0625$, $\frac{1}{2} = 0.5$
 - $\frac{1}{12} = 0.083333333333 \dots$ 3 repeats for ever
 - $\frac{5}{7} = 0.714285714285714285 \dots$ 714285 repeat forever
 - $\frac{1}{6} = 0.1(6)$, $\frac{1}{12} = 0.08(3)$. $\frac{5}{7} = 0.(71428)$, $\frac{1}{2} = 0.5(0)$
- How to know # of digits before cycle of n/d ?
 - [Programming](#)? mark reminders of long division
 - Mathematically? [See](#)

Double Comparison

■ Operations can result in double value

- Internal value can be shifted with +/- EPS
- EPS is a very small amount (e.g. $1e-10$)
- e.g. $x = 4.7$ may be internally 4.7000001 or 4.69999999
- so `if(x == 4.7)` fails! Although printing shows 4.7

■ Printing zero is tricky (-0.00 problem)

- Compare x first with zero. If zero, then $x = 0$

```
// return 0 for a==b, 1 for a > b, -1 for a < b
int comp_double(double a, double b)
{
    // if very small difference, then equal
    if (fabs(a-b) <= 1e-10)
        return 0;
    return a < b ? -1 : 1;
}
```

Big Integers

■ Sometimes computations are too big

- $1775!$ is 4999 digits

■ Java/C# implement BigInteger library to handle such big computations

■ In C++, you may implement it by yourself.

- Exercise: Try to represent/add/multiply big numbers
- I wrote this [code](#) when was young. Use freely (not in TC)
- Main idea: Think in number as **reversed array**

■ In competitions, nowadays problem setters avoid such problems most of time

Big Integers: Factorial

- Create a big array. Initialize it to 1
- Think in it as **reversed** array. Arr[0] first digit
 - e.g. 120 represented as 021 (arr[0] = 0)
- From $i = 2$ to N
 - Multiply i in every cell
 - For every cell, if its value > 9 handle its carry
 - $v \Rightarrow (v \% 10, v / 10)$
 - For last cell, check if it has carry (typically will have), and put it in next cell, **AS LONG AS** there is a carry
- See code example in previous page link

Big Integers: Factorial

1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0

Initialize $1 = 1!$

Multiply $2 = 2!$

Multiply $3 = 3!$

Multiply 4

Remember...Keep only a digit in cell...move carry to next cell

4	2	0	0	0	0	0	0	0	0
20	10	0	0	0	0	0	0	0	0
0	12	0	0	0	0	0	0	0	0
0	2	1	0	0	0	0	0	0	0

4! But REVERSED

Multiply 5

Move Carry 1st cell

Move Carry 2nd cell = 5!

Be careful from last cell...we may keep shift carry to right many times when N is large

Rounding Values

- Rounding is replacing value with another approximate value...many types and styles
- In C++, we have 4 rounding functions
 - **round**: nearest integer to x [halfway cases away from 0]
 - **floor**: round down
 - **ceil**: round up
 - **trunc**: rounds toward zero (remove fraction)
- In integers, x/y is floor of results
 - $\text{ceil}(x, y) = (x+y-1)/y$
 - $\text{round}(x, y) = (x+y/2)/y$ [if $x > 0$] and $(x-y/2)/y$ [$x < 0$]

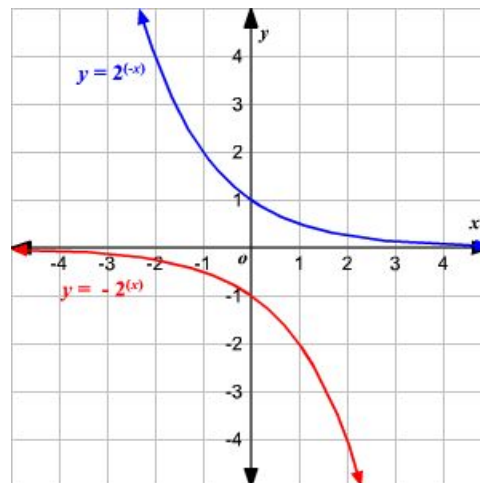
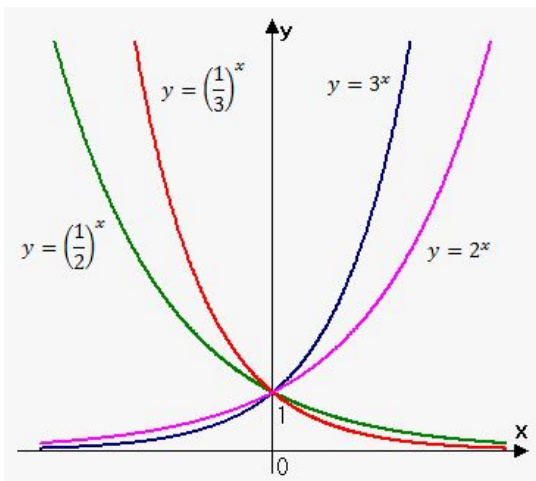
Rounding Values: Examples

- Be careful from -ve and 0.5
- | value | round | floor | ceil | trunc |
|-------|-------|-------|------|-------|
| 2.3 | 2.0 | 2.0 | 3.0 | 2.0 |
| 3.8 | 4.0 | 3.0 | 4.0 | 3.0 |
| 5.5 | 6.0 | 5.0 | 6.0 | 5.0 |
| -2.3 | -2.0 | -3.0 | -2.0 | -2.0 |
| -3.8 | -4.0 | -4.0 | -3.0 | -3.0 |
| -5.5 | -6.0 | -6.0 | -5.0 | -5.0 |
- $\text{round}(x) == x < 0 ? \text{ceil}(x-0.5) : \text{floor}(x+0.5);$
 - To round to multiple of a specified amount
 - $\text{round}(x, m) = \text{round}(x / m) * m$
 - $\text{round}(48.2 \text{ seconds}, 15) = 45 \text{ seconds}$
 - $\text{round}(2.1784 \text{ dollars}, 0.01 (1 \text{ cent})) = 2.18 \text{ dollars}$

Exponential function

- If we have 2 jackets, 2 jeans & 2 shoes, I can have $2 \times 2 \times 2 = 2^3 = 8$ different clothing styles
- Exponential function: $y = b^x$
- b (base), x (exponent): real, integer, +ve, -ve
- Popular values: 2, 10, e [e is **Euler's** number ≈ 2.7]
 - a 64 bit integer stores 2^{64} numbers ... a very **big** number
- $2^{-x} = (\frac{1}{2})^x = 0.5^x$ and $2^x = (\frac{1}{2})^{-x} = 0.5^{-x}$
- In C++: $\text{pow}(2, 3) = 2^3 = 8$
- Exponential indicates growing **fast**

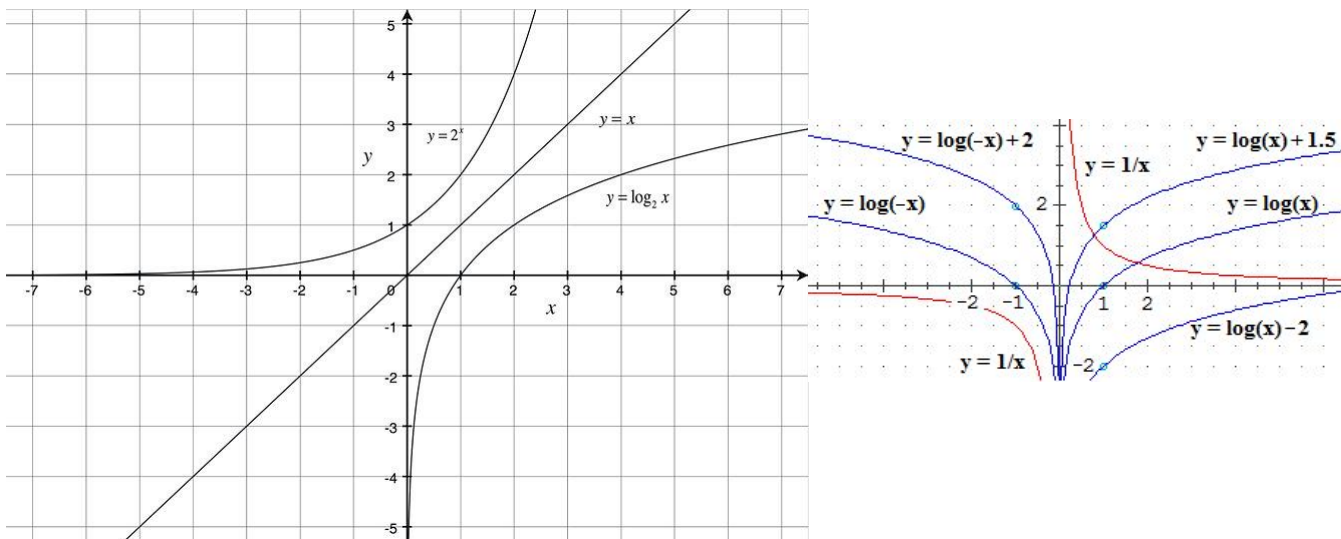
Exponential function: Graphs



Logarithm

- It is the **inverse** operation to **exponentiation**
- $y = b^x \implies \log_b y = x$
 - $\log_{10} 1000 = \text{how many 10 multiplications} = 1000? 3$
 - $\log_2 16 = \text{how many 2 multiplications} = 16? 4$
 - $\log_{10} 0.001 = \text{how many 10 divisions} = 1/1000? -3$
- $b=[10, e, 2] \implies (\text{common, natural, binary}) \log$
 - Math notations: $\lg(x)$, $\ln(x)$, $\text{lb}(x)$
 - In c++: $\log_{10}(x)$, $\log(x)$, $\log_2(x)$
- \log is **strictly** increasing for $b > 1$
- \log is strictly decreasing for $0 < b < 1$

Logarithm: Graphs



\log is **strictly** increasing function. Strictly [nothing equal], increasing, go up.

1 2 5 5 7 9 [Increasing]

1 2 5 6 7 9 [strictly Increasing]

9 7 5 5 2 1 [decreasing]

9 7 6 5 2 1 [strictly decreasing]

Logarithm Operations

	Formula	Example
product	$\log_b(xy) = \log_b(x) + \log_b(y)$	$\log_3(243) = \log_3(9 \cdot 27) = \log_3(9) + \log_3(27) = 2 + 3 = 5$
quotient	$\log_b\left(\frac{x}{y}\right) = \log_b(x) - \log_b(y)$	$\log_2(16) = \log_2\left(\frac{64}{4}\right) = \log_2(64) - \log_2(4) = 6 - 2 = 4$
power	$\log_b(x^p) = p \log_b(x)$	$\log_2(64) = \log_2(2^6) = 6 \log_2(2) = 6$
root	$\log_b \sqrt[p]{x} = \frac{\log_b(x)}{p}$	$\log_{10} \sqrt{1000} = \frac{1}{2} \log_{10} 1000 = \frac{3}{2} = 1.5$

$$\log_b(x) = \frac{\log_k(x)}{\log_k(b)}. \quad \log_b(x) = \frac{\log_{10}(x)}{\log_{10}(b)} = \frac{\log_e(x)}{\log_e(b)}. \quad b = x^{\frac{1}{\log_b(x)}}.$$

- $\log_b b = 1, \log_b 1 = 0, \log_b 0 = -\infty, \log_1 x = \text{undefined}$
- $b^{\log_b(x)} = x \Rightarrow \text{take } \log_b \text{ for the equation to get } x$
- $xb^y \Rightarrow b^{\log_b(x) + y}$

Logarithm and # of digits

- $\log_{10}(10x) = \log_{10}(10) + \log_{10}(x) = 1 + \log_{10}(x).$
- base 10 can be used to know # of digits
 - # digits = $1 + \text{floor}(\log_{10}(x))$
 - $\log_{10}(1000) = 3 \Rightarrow 4 \text{ digits}$
 - $\log_{10}(1430) = 3.15 \Rightarrow 4 \text{ digits}$
 - $\log_{10}(9999) = 3.99 \Rightarrow 4 \text{ digits}$
 - $\log_{10}(10000) = 4 \Rightarrow 4 \text{ digits}$
 - So from 1000 to 10000-1, we have $\log_{10}(x) = 3.\text{xyz}..$
- Generally, # of digits in base b.
 - $\log_2(16) = 4 \Rightarrow 5 \text{ bits. [16 in base 10 = 10000 in base 2]}$
- Homework: # of digits of factorial n?

Math Materials

- Knowledge of interest [here](#)
- Mathematics part in
 - Programming Challenge
 - Competitive Programming
 - Algorithms Books (e.g. CLR)
- Other Math Books
 - Concrete Mathematics
 - [Discrete Mathematics](#) and Its Applications
 - Mathematics for Computer Science (2013)
- Solving...Solving...Solving
 - if can't solve..see editorial/solution...take notes

Time To Solve

- There are some ad-hoc problems in the video
- There also some problems on Big Integers
 - This type of problems usually don't appear nowadays
 - Why?: problem setters avoid languages advantages
- Warmup by solving some of them :)
- Also read this [cheat sheet](#)

تم بحمد الله

علمكم الله ما ينفعكم

ونفَعكم بما تعلمتم

وزادكم علماً