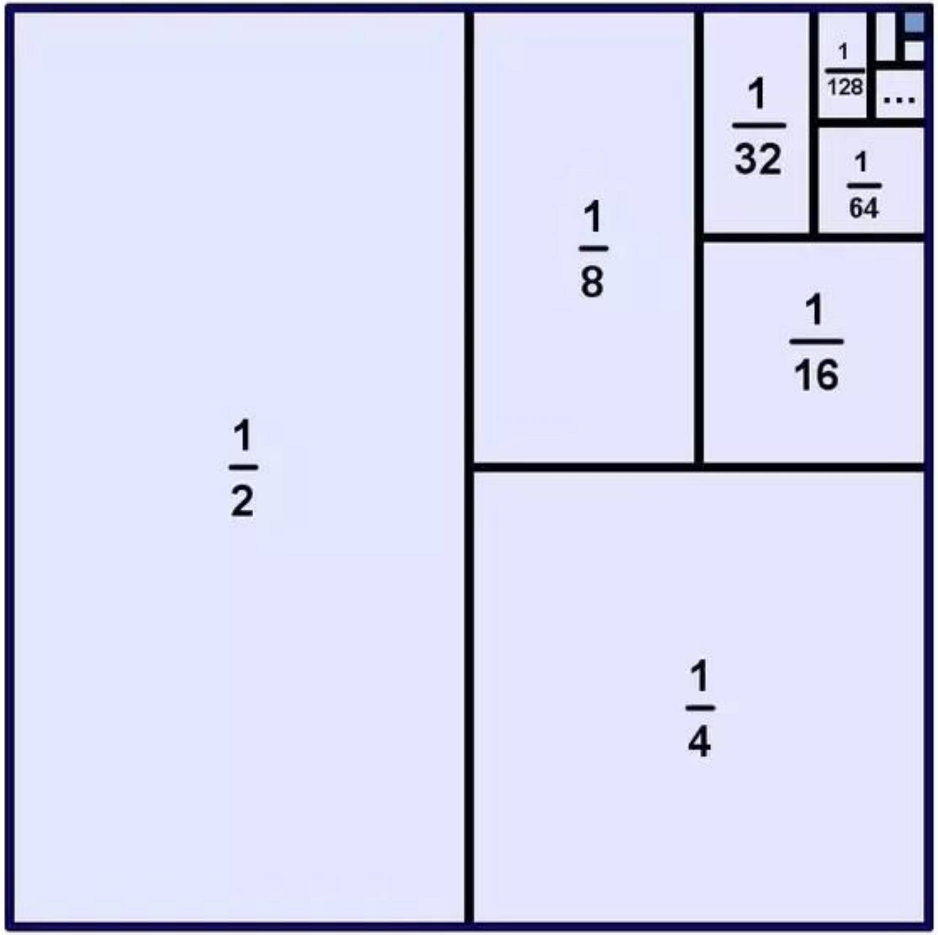
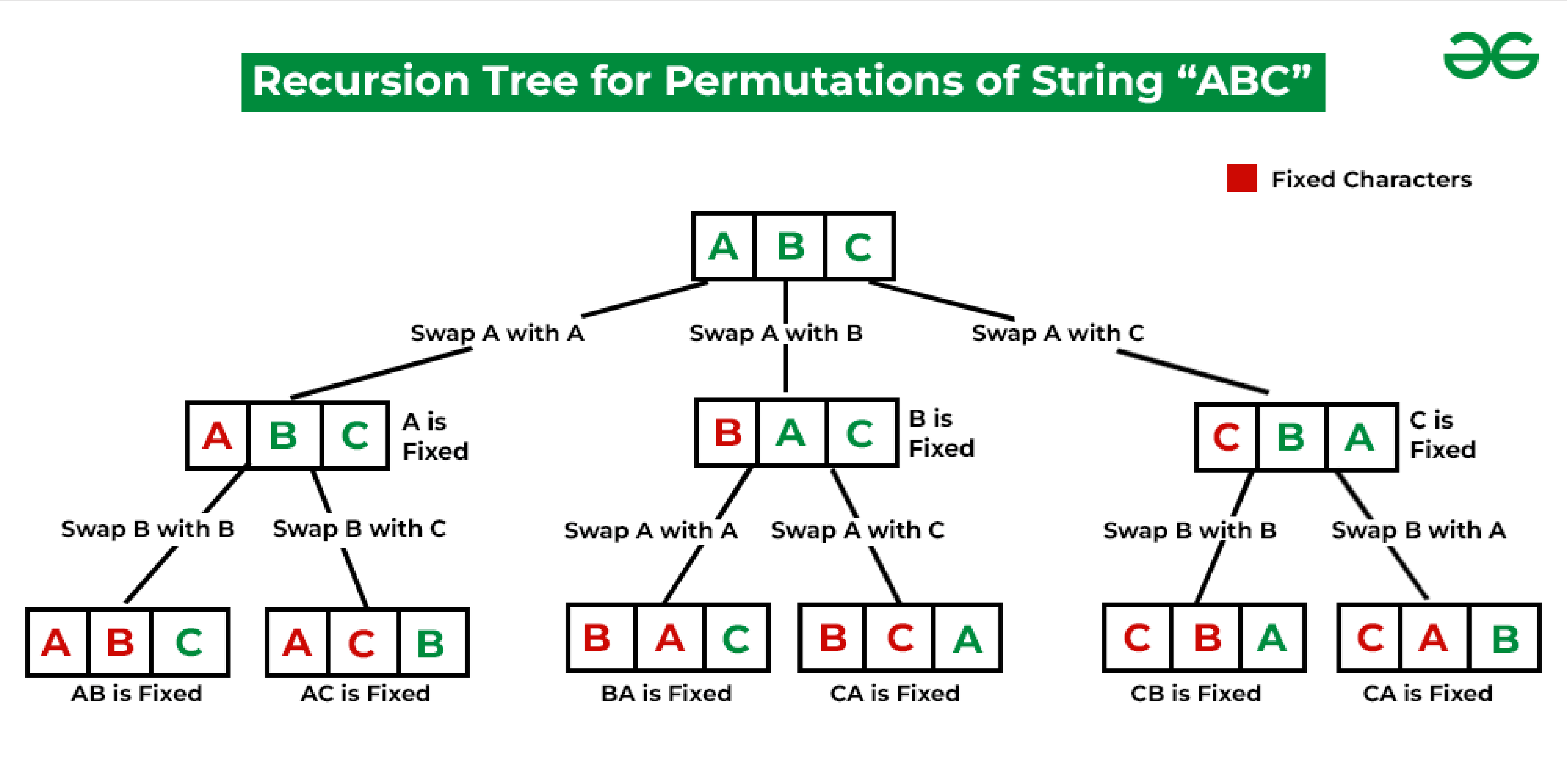
Math

Prove the Inequality

$$\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots+\frac{1}{2^{n-1}}<1$$



$\frac{1}{2}$
$\frac{1}{8}$
$\frac{1}{32}$
$\frac{1}{128}$
...
$\frac{1}{64}$
$\frac{1}{16}$
$\frac{1}{4}$



Recursion Tree for Permutations of String "ABC"
Fixed Characters
A B C
Swap A with A
Swap A with B
Swap A with C
A B C
A is Fixed
B A C
B is Fixed
C B A
C is Fixed
Swap B with B
Swap B with C
Swap A with A
Swap A with C
Swap B with B
Swap B with A
A B C
AB is Fixed
A C B
AC is Fixed
B A C
BA is Fixed
B C A
CA is Fixed
C B A
CB is Fixed
C A B
CA is Fixed

Factorial

$$n! = n * (n-1) * (n-2) * (n-3) * \cdots * 3 * 2 * 1$$

0! = 1

1! = 1

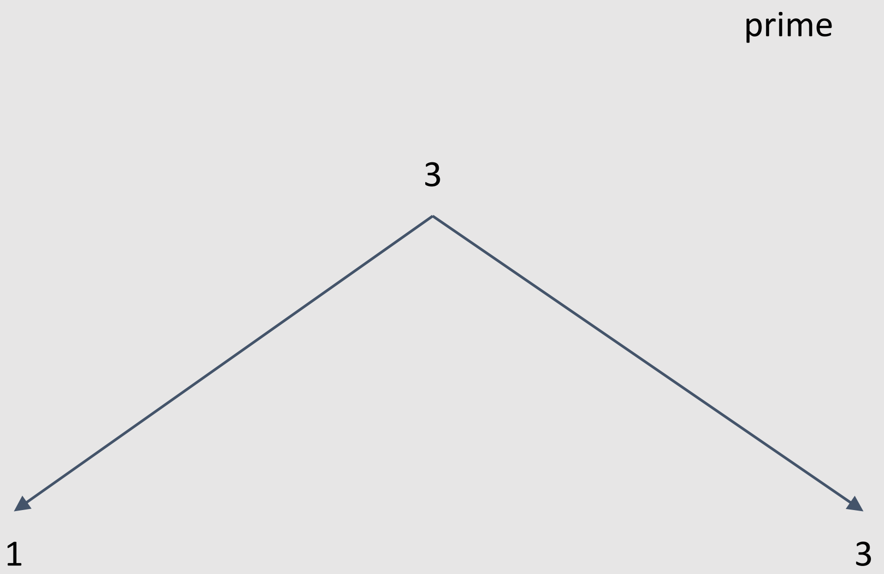
2! = 2 x 1 = 2

3! = 3 x 2 x 1 = 6

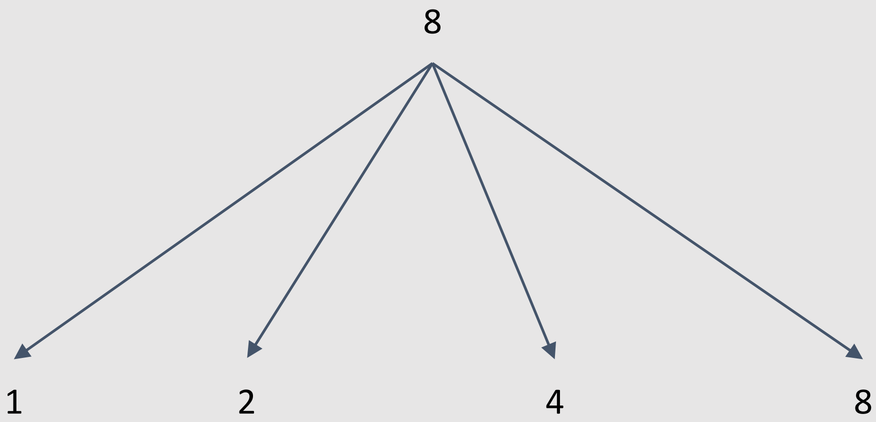
4! = 4 x 3 x 2 x 1 = 24

PRIME NUMBERS

2, 3, 5, 7, 11, 13, 17, 19, 23,
29, 31, 37, 41, 43, 47, 53, 59,
61, 67, 71, 73, 79, 83, 89, 97


prime
3
1
3

composite



```python
amount_divisors = 0

number = int(input())

for i in range(1, number + 1):
    if number % i == 0:
        amount_divisors += 1

if amount_divisors == 2:
    print("prime")
else:
    print("composite")
```

#stdin #stdout 0.03s 9624KB

stdin

7

stdout

prime

```python
amount_divisors = 0

number = int(input())

for i in range(1, number + 1):
    if number % i == 0:
        amount_divisors += 1

if amount_divisors == 2:
    print("prime")
else:
    print("composite")
```

#stdin #stdout 0.03s 9684KB

stdin

8

stdout

composite

```cpp
#include <iostream>

using namespace std;

int main() {
    int amount_divisors = 0;
    int number;
    cin >> number;
    for (int i = 1; i <= number; i++) {
        if (number % i == 0) {
            amount_divisors += 1;
        }
    }

    if (amount_divisors == 2) {
        cout << "prime";
    }
    else {
        cout << "composite";
    }
}
```

#stdin #stdout 0.01s 5308KB

stdin

7

stdout

prime

```cpp
#include <iostream>

using namespace std;

int main() {
    int amount_divisors = 0;
    int number;
    cin >> number;
    for (int i = 1; i <= number; i++) {
        if (number % i == 0) {
            amount_divisors += 1;
        }
    }

    if (amount_divisors == 2) {
        cout << "prime";
    }
    else {
        cout << "composite";
    }
}
```

#stdin #stdout 0.01s 5400KB

stdin

8

stdout

composite

Asymptotic

- We need to check each number from 1 to n
- $O(n)$

How to do faster?

N

<= √N

check all numbers

> √N

do nothing

Why we can ignore “> √N”

- $X > \sqrt{N}$, $N / X = Y$
- $N / X < N / \sqrt{N} \rightarrow N / X < \sqrt{N} \rightarrow Y < \sqrt{N}$
- So if we have divisor > $\sqrt{N}$, we also will have divisor < $\sqrt{N}$

Example

- We need to check 10
- $5 > \sqrt{10}$, $10 / 5 = 2$
- $10 / 5 < 10 / \sqrt{10} \rightarrow 10 / 5 < \sqrt{10} \rightarrow 2 < \sqrt{10}$

```python
amount_divisors = 0

number = int(input())
i = 1

while i * i <= number:
    if number % i == 0:
        amount_divisors += 1
    i += 1

if number != 1 and amount_divisors == 1:
    print("prime")
else:
    print("composite")
```

#stdin #stdout 0.03s 9848KB

stdin

8

stdout

composite

```python
amount_divisors = 0

number = int(input())
i = 1

while i * i <= number:
    if number % i == 0:
        amount_divisors += 1
    i += 1

if number != 1 and amount_divisors == 1:
    print("prime")
else:
    print("composite")
```

#stdin #stdout 0.03s 9700KB

stdin

7

stdout

prime

```cpp
#include <iostream>

using namespace std;

int main() {
    int amount_divisors = 0;
    int number;
    cin >> number;
    for (int i = 1; i * i <= number; i++) {
        if (number % i == 0) {
            amount_divisors += 1;
        }
    }

    if (number != 1 && amount_divisors == 1) {
        cout << "prime";
    }
    else {
        cout << "composite";
    }
}
```

#stdin #stdout 0.01s 5360KB

stdin

8

stdout

composite

```cpp
#include <iostream>

using namespace std;

int main() {
    int amount_divisors = 0;
    int number;
    cin >> number;
    for (int i = 1; i * i <= number; i++) {
        if (number % i == 0) {
            amount_divisors += 1;
        }
    }

    if (number != 1 && amount_divisors == 1) {
        cout << "prime";
    }
    else {
        cout << "composite";
    }
}
```

#stdin #stdout 0.01s 5304KB

stdin

7

stdout

prime

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Prime numbers

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Prime numbers

```cpp
#include <iostream>

using namespace std;
const int MAX_NUM = 100;

int main() {
    bool table[MAX_NUM];
    for (int i = 0; i < MAX_NUM; i++) {
        table[i] = true;
    }
    table[0] = table[1] = false;
    for (long long i = 2; i < MAX_NUM; i++) {
        if (table[i]) {
            for (long long j = i * i; j < MAX_NUM; j += i) {
                table[j] = false;
            }
            cout << i << " ";
        }
    }
}
```

#stdin #stdout 0.01s 5520KB

stdin

```
10
```

stdout

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 8
```

```python
table = [True] * 100
table[0] = table[1] = False
for i in range(2, 100):
    if (table[i]):
        for j in range(i * i, 100, i):
            table[j] = False
        print(i)
```

#stdin #stdout 0.03s 9516KB

stdin

Standard input is empty

stdout

2
3
5
7
11
13
17
19
23

Asymptotic

- to erase 2 - n / 2, to erase 3 - n / 3, to erase 5 - n / 5 and so on
- asymptotic - $O(\Sigma(n / i))$, where i is prime
- $\Sigma(n / i)$, where i is prime $<= \Sigma(n / i) <= O(n\log n)$
- Moreover, it is even possible to prove that such an implementation works for $O(n\log(\log(n)))$, but it is rather complicated and we will not do it for now

THE ASSOCIATIVE PROPERTY OF MULTIPLICATION EXPLAINED!

$$(ab)c = a(bc)$$

$$(8 \times 4) \times 2 = 8 \times (4 \times 2)$$

Commutative Property of Multiplication

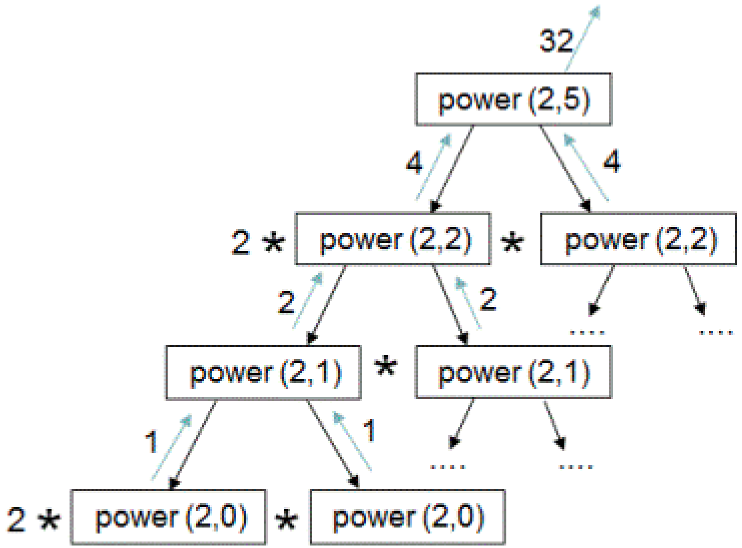
Switching the order of
the multiplicand (the first factor)
and the multiplier (the second factor)
does not change the product.

$$4 \cdot 5 = 20$$
$$5 \cdot 4 = 20$$

The generic formula $a \cdot (b) = b \cdot (a)$

Associative

- $A^4 = A * A * A * A = (A * A) * (A * A) = A^2 * A^2 = (A^2)^2$
- $A^n = A * A * \ldots * A * A = (A * \ldots * A) * (A * \ldots * A) = A^{(n/2)} * A^{(n / 2)} = (A^{(n / 2)})^2$



partial recursion tree for calculating 2^5

```python
def bin_pow(a, n):
    if n == 1:
        return a
    if n == 0:
        return 1
    res = bin_pow(a, n // 2)
    return res * res * bin_pow(a, n % 2)

a = 2
n = 10
print(bin_pow(a, n))
```

#stdin #stdout 0.03s 9508KB

stdin

```
2 10
```

stdout

```
1024
```

```cpp
#include <iostream>

using namespace std;
const int MOD = 1e9 + 7;

int bin_pow(int a, int n) {
    if (n == 1) {
        return a;
    }
    if (n == 0) {
        return 1;
    }
    int res = bin_pow(a, n / 2);
    return res * res * bin_pow(a, n % 2);
}

int main() {
    int a, n;
    cin >> a >> n;
    cout << bin_pow(a, n);
}
```

#stdin #stdout 0.01s 5472KB

stdin

```
2 10
```

stdout

```
1024
```

Modulo

- Let’s imagine we work only with integers [0, MOD)
- $X < Y \nRightarrow X \bmod MOD < Y \bmod MOD$
- $X > Y \nRightarrow X \bmod MOD > Y \bmod MOD$
- $X \neq Y \nRightarrow X \bmod MOD \neq Y \bmod MOD$
- $X = Y \Rightarrow X \bmod MOD = Y \bmod MOD$

Modulo

- MOD = 10
- 9 < 11, BUT (9 % 10) > (1 % 10), 9 > 1
- 9 ≠ 19, BUT (9 % 10) = (19 % 10), 9 = 9
- 11 > 9, BUT (11 % 10) < (9 % 10), 1 < 9
- X = Y, (A + MOD * C1) % MOD = (A + MOD * C1) % MOD, A = A, example: 13 = 13, 3 = 3

Modulo

- (A + B) % MOD = (A % MOD + B % MOD) % MOD
- (A - B) % MOD = (A % MOD - B % MOD) % MOD + MOD
- (A * B) % MOD = ((A % MOD) * (B % MOD)) % MOD
- (A / B) % MOD = ?????

Modulo

- MOD = 10
- (11 + 19) % 10 = 30 % 10 = 0, (11 % 10 + 19 % 10) % 10 = (1 + 9) % 10 = 0
- (11 - 19) % 10 = -8 % 10 = 2, (11 % 10 - 19 % 10) % 10 + 10 = (1 - 9) % 10 + 10 = 2
- (11 * 19) % 10 = 209 % 10 = 9, ((11 % 10) * (19 % 10)) % 10 = (1 * 9) % 10 = 9

Modulo

- (A / B) % MOD = ((A % MOD) / (B % MOD)) % MOD
- (100 / 2) % 50 = 2, ((100 % 50) / 2) = 0
- (100 / 50) % 50 = 2, ((100 % 50) / (50 % 50)) = 0 / 0 = ?

How does division work?

- (A / B) = A * (1 / B) = A * B^(-1)
- (4 / 2) = 4 * (1 / 2) = 4 * 0.5 = 2
- What is B^(-1)?
- B^(-1) * B = 1
- 2 * 0.5 = 1
- B^(-1) for modular

What if p is prime?

$$a^{p-1} \equiv 1 \pmod{p}$$

$$20^{7-1} \equiv 1 \pmod{7}$$

$$20^{6} \equiv 1 \pmod{7}$$

$$64,000,000 \equiv 1 \pmod{7}$$

$$\frac{64,000,000 - 1}{7} \equiv 9,142,857$$

$X = 1 \cdot 2 \cdot \cdots \cdot (p-1)$
$Y = a \cdot (2a) \cdot \cdots \cdot (p-1)a$
$1, 2, 3, \ldots, (p-1)$ - all unique and no zero
$(a)\%p, (2a)\%p, (3a)\%p, \ldots, ((p-1)a)\%p$ - all unique and no zero(because p is prime)

Example?

- $p = 7, a = 3$
- $1 * 2 * 3 * 4 * 5 * 6$
- $(3 * 6 * 9 * 12 * 15 * 18) \bmod p = (3 * 6 * 2 * 5 * 1 * 4) \bmod p$

$X\%p = Y\%p$, because $X\%p = Y\%p = 1 \cdot 2 \cdot \cdots \cdot (p-1)$
$(p-1)! \equiv a^{p-1} \cdot (p-1)!$, $(p-1)! \neq 0 \Rightarrow 1 \equiv a^{p-1}$

```cpp
#include <iostream>

using namespace std;
const long long a = 10000, b = 5000, m = 1e9 + 7;

long long bin_pow(long long a, long long n) {
    if (n == 1) {
        return a;
    }
    if (n == 0) {
        return 1;
    }
    long long res = bin_pow(a, n / 2);
    return ((res * res) % m * bin_pow(a, n % 2)) % m;
}

int main() {
    long long reverse = bin_pow(b, m - 2);
    cout << reverse << " " << (reverse * b) % m << " " << (a * reverse) % m;
}
```

#stdin #stdout 0.01s 5508KB

stdin

```
Standard input is empty
```

stdout

```
571400004 1 2
```

```python
a, b, m = 10000, 5000, int(1e9 + 7)

def bin_pow(a, n):
    if n == 1:
        return a
    if n == 0:
        return 1
    res = bin_pow(a, n // 2)
    return ((res * res) % m * bin_pow(a, n % 2)) % m

reverse = bin_pow(b, m - 2)
print(reverse, (reverse * b) % m, (reverse * a) % m)
```

Успешно #stdin #stdout 0.03s 9520KB

stdin

```
2 10
```

stdout

```
571400004 1 2
```

Harmonic

$$\sum_{i=1}^{n} \frac{1}{i} = \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n} = O(log(n))$$

Let's round $\frac{1}{i}$ to $\frac{1}{2^j}$, where $2^j \geq i$

$$\sum_{i=1}^{n} \frac{1}{i} < \frac{1}{1} + \frac{1}{2} + \frac{1}{2} + \frac{1}{4} \leq O(log(n))$$

$$\sum_{i=1}^{n} \frac{1}{i} \leq O(log(n))$$

Let's round $\frac{1}{i}$ to $\frac{1}{2^j}$, where $2^j \leq i$

$$\sum_{i=1}^{n} \frac{1}{i} \geq O(log(n))$$

Factorization

- 52 = 2 * 2 * 13
- 36 = 2 * 2 * 3 * 3
- 44 = 2 * 2 * 11

GCD

- $52 = 2 * 2 * 13$
- $36 = 2 * 2 * 3 * 3$
- $GCD(52, 36) = COMMON(2 * 2 * 13, 2 * 2 * 3 * 3) = 2 * 2$
- $GCD(52, 36) = 2 * 2 * GCD(9, 13) = 4$

LCM

- $52 = 2 * 2 * 13$
- $36 = 2 * 2 * 3 * 3$
- $LCM(52, 36) = COMMON(2 * 2 * 13, 2 * 2 * 3 * 3) * UNIQUE(2 * 2 * 13, 2 * 2 * 3 * 3) = 2 * 2 * 3 * 3 * 13 = 468$
- $LCM(52, 36) = 2 * 2 * LCM(9, 13) = 468$

LCM and GCD

- 52 = 2 * 2 * 13, 36 = 2 * 2 * 3 * 3
- GCD(52, 36) = 4
- LCM(A, B) = A * B / GCD(A, B) = 52 * 36 / 4 = 468
- LCM(A, B) = COMMON * UNIQUE, A * B = COMMON * COMMON * UNIQUE, GCD(A, B) = COMMON
- LCM(A, B) = COMMON * COMMON * UNIQUE / COMMON = COMMON * UNIQUE

GCD(A, B)

- $GCD(A, B) = GCD(A - B, B)$, if $A > B$
- $GCD(A, B) = X$
- $GCD(A - B, B) = GCD(A - X * C, B) = GCD(A, B)$
- $GCD(52, 36) = GCD(16, 36) = GCD(16, 4) = GCD(0, 4) = 4$
- $GCD(A, B) = GCD(B, A \text{ MOD } B) = GCD(B, A - B * C2) =$
  $GCD(B \text{ MOD } A, A) = GCD(B - A * C1, A)$

```python
def gcd(a, b):
    if b == 0:
        return a
    return gcd(b, a % b)

a = int(input())
b = int(input())
print(gcd(a, b))
```

#stdin #stdout 0.04s 9680KB

stdin

```
52
36
```

stdout

```
4
```

```
#include <iostream>

using namespace std;

int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}

int main () {
    int a, b;
    cin >> a >> b;
    cout << gcd(a, b);
}
```

#stdin #stdout 0.01s 5504KB

stdin

52
36

stdout

4

Facts about GCD

- GCD(a, b) = GCD(b, a)

1) GCD(a, b) <= GCD(b, a), because GCD(a, b) = x, a = x * c1, b = x * c2 ⇒ GCD(b, a) = GCD(x * c2, x * c1) >= x
2) GCD(b, a) <= GCD(a, b), because GCD(b, a) = x, b = x * c1, a = x * c2 ⇒ GCD(a, b) = GCD(x * c2, x * c1) >= x

GCD(a, b) <= GCD(b, a) <= GCD(a, b) ⇒ GCD(a, b) = GCD(b, a)

Facts about GCD

- $GCD(a, GCD(b, c)) = GCD(GCD(a, b), c)$

1) $GCD(a, GCD(b, c)) \le GCD(GCD(a, b), c)$, because $GCD(b, c) = x$, $GCD(a, x) = y \Rightarrow a \bmod y = b \bmod y = c \bmod y = 0 \Rightarrow GCD(a, b) \ge y$, $GCD(y, c) = y$
2) $GCD(GCD(a, b), c) \le GCD(a, GCD(b, c))$, because $GCD(a, b) = x$, $GCD(x, c) = y \Rightarrow a \bmod y = b \bmod y = c \bmod y = 0 \Rightarrow GCD(b, c) \ge y$, $GCD(a, y) = y$

Facts about GCD

- Asymptotic - O(log(n))
- One of the worst cases of Euclid's Algorithm is Fibonacci numbers
- fib_n = fib_(n - 1) + fib_(n - 2), so fib_n % fib_(n - 1) = fib_(n - 2)
- GCD(fib_n, fib_(n - 1)) = GCD(fib_n % fib_(n - 1), fib_(n - 1)) = GCD(fib_(n - 2), fib_(n - 1))

Asymptotic

- GCD(a, b) = GCD(b, a % b)
- if a > b then (a % b) * 2 < a
- (a % b) = c, a = b * k_1 + c; c < b, k_1 >= 1
- a > c * k_1 + c >= c * 2

$GCD(a, b, c) = GCD(GCD(a, b), c)$

- We need to prove $GCD(a, b, c) = GCD(GCD(a, b), c)$
- $GCD(a, b, c) = d \Rightarrow GCD(a, b) = d * k\_1, c = d * k\_2$
- We need to prove $GCD(k\_1, k\_2) = 1$
- $GCD(k\_1, k\_2) = k\_3 \Rightarrow a \bmod (k\_3 * d) = b \bmod (k\_3 * d) = c \bmod (k\_3 * d) = 0 \Rightarrow GCD(a, b, c) = k\_3 * d$
- Contradiction

Facts about GCD

- $x = GCD(c, d)$
- $GCD(a, b, x) = GCD(GCD(a, b), x) \Rightarrow GCD(a, b, GCD(c, d)) = GCD(GCD(a, b), GCD(c, d))$
- $GCD(a, b, GCD(c, d)) = GCD(a, GCD(b, c, d)) = GCD(a, GCD(b, GCD(c, d)))$

Binary GCD

- $GCD(A, B) = 2 * GCD(A / 2, B / 2)$, if $(A \bmod 2 = 0)$ and $(B \bmod 2 = 0)$
- $GCD(A, B) = GCD(A / 2, B)$, if $(A \bmod 2 = 0)$ or $(B \bmod 2 = 0)$
- $GCD(A, B) = GCD(A - B, B)$, if $A > B$
- $GCD(A, B) = GCD(A, B - A)$, if $A <= B$

```python
def gcd(a, b):
    if a == 0 or b == 0:
        return max(a, b)
    if b % 2 == 0 and a % 2 == 0:
        return 2 * gcd(a // 2, b // 2)
    if b % 2 == 0:
        return gcd(a, b // 2)
    if a % 2 == 0:
        return gcd(a // 2, b)
    if a > b:
        return gcd(a - b, b);
    return gcd(a, b - a);

a = 52
b = 36
print(gcd(a, b))
```

#stdin #stdout 0.04s 9444KB

stdin

```
Standard input is empty
```

stdout

```
4
```

```cpp
#include <iostream>

using namespace std;

int gcd(int a, int b) {
    if (a == 0 || b == 0) {
        return max(a, b);
    }
    if (b % 2 == 0 && a % 2 == 0) {
        return 2 * gcd(a / 2, b / 2);
    }
    if (b % 2 == 0) {
        return gcd(a, b / 2);
    }
    if (a % 2 == 0) {
        return gcd(a / 2, b);
    }
    if (a > b) {
        return gcd(a - b, b);
    }
    return gcd(a, b - a);
}

int main () {
    int a, b;
    cin >> a >> b;
    cout << gcd(a, b);
}
```

#stdin #stdout 0.01s 5432KB

stdin

52
36

stdout

4

Asymptotic

O(log(a) + log(b))

- (a or b) - even ⇒ a / 2 or b / 2
- (a and b) - odd ⇒ (a -= b or b -= a) and (a / 2 or b / 2)

every 2 operations we divide number

$$P(a_1, \ldots, a_n, x_1, \ldots, x_m) = 0$$

$ax + by = c$	This is a linear Diophantine equation.
$w^3 + x^3 = y^3 + z^3$	The smallest nontrivial solution in positive integers is $12^3 + 1^3 = 9^3 + 10^3 = 1729$. It was famously given as an evident property of 1729, a taxicab number (also named Hardy–Ramanujan number) by Ramanujan to Hardy while meeting in 1917.[1] There are infinitely many nontrivial solutions.[2]
$x^n + y^n = z^n$	For $\mathrm{n} = 2$ there are infinitely many solutions (x, y, z): the Pythagorean triples. For larger integer values of n, Fermat's Last Theorem (initially claimed in 1637 by Fermat and proved by Andrew Wiles in 1995[3]) states there are no positive integer solutions (x, y, z).
$x^2 - ny^2 = \pm 1$	This is Pell's equation, which is named after the English mathematician John Pell. It was studied by Brahmagupta in the 7th century, as well as by Fermat in the 17th century.
$\frac{4}{n} = \frac{1}{x} + \frac{1}{y} + \frac{1}{z}$	The Erdős–Straus conjecture states that, for every positive integer $n \geq 2$, there exists a solution in x, y, and z, all as positive integers. Although not usually stated in polynomial form, this example is equivalent to the polynomial equation $4xyz = yzn + xzn + xyn = n(yz + xz + xy)$.
$x^4 + y^4 + z^4 = w^4$	Conjectured incorrectly by Euler to have no nontrivial solutions. Proved by Elkies to have infinitely many nontrivial solutions, with a computer search by Frye determining the smallest nontrivial solution, $95800^4 + 217519^4 + 414560^4 = 422481^4$.[4][5]

$ax + by \neq gcd(a, b) * k$

- $ax + by = c$, $c \neq gcd * k_1$
- $(ax + by) = gcd * k_2$
- no solution

$ax + by = gcd(a, b)$

- $ax + by = gcd(a, b)$
- if we have $(a, b) = (gcd(a, b), 0)$, then answer is $(x, y) = (1, 0)$, because $gcd(a, b) * x = gcd(a, b)$
- if we have $(a, b) = (0, gcd(a, b))$, then answer is $(x, y) = (0, 1)$, because $gcd(a, b) * y = gcd(a, b)$

$ax + by = \gcd(a, b)$

- $ax + by = \gcd(a, b)$
- $(b \% a) * x_1 + a * y_1 = \gcd(a, b)$
- $(b - \lfloor b/a \rfloor * a) * x_1 + a * y_1 = \gcd(a, b)$
- $b * x_1 + a * (y1 - \lfloor b/a \rfloor * x_1) = \gcd(a, b)$

$ax + by = \gcd(a, b)$

- we have $b * x_1 + a * (y1 - \lfloor b/a \rfloor * x_1) = \gcd(a, b)$
- we had $ax + by = \gcd(a, b)$
- $y = x_1$, $x = (y1 - \lfloor b/a \rfloor * x_1)$

Example

- $36 * x + 52 * y = 4$
- $(52 \% 36) * x\_1 + 36 * y\_1 = 4$

  $16 * x + 36 * y = 4$
- $(36 \% 16) * x\_1 + 16 * y\_1 = 4$

  $4 * x + 16 * y = 4$
- $(16 \% 4) * x\_1 + 4 * y\_1 = 4$

  $0 * x + 4 * y = 4$

Example

- $0 * x + 4 * y = 4 \Rightarrow (x, y) = (0, 1)$

To check $0 * 0 + 4 * 1 = 4$

- $4 * x + 16 * y = 4 \Rightarrow y = x\_1 = 0, x = (y\_1 - \lfloor b/a \rfloor * x\_1) = 1 - 4 * 0 = 1$

To check $4 * 1 + 0 * 0 = 4$

Example

- $16 * x_1 + 36 * y_1 = 4 \Rightarrow y = x_1 = 1, x = (y_1 - \lfloor b/a \rfloor * x_1) = 0 - 2 * 1 = -2$

To check $16 * -2 + 36 * 1 = 4$

- $52 * x_1 + 36 * y_1 = 4 \Rightarrow y = x_1 = -2, x = (y_1 - \lfloor b/a \rfloor * x_1) = 1 + 1 * 2 = 3$

To check $52 * -2 + 36 * 3 = 4$

```python
class Erat:
    def __init__(self, gcd, x, y):
        self.gcd = gcd
        self.x = x
        self.y = y

    def __iter__(self):
        return iter((self.gcd, self.x, self.y))

def gcd(a, b):
    if a == 0:
        return Erat(b, 0, 1)
    (gcd_res, x_1, y_1) = gcd(b % a, a)
    x, y = y_1 - (b // a * x_1), x_1
    return Erat(gcd_res, x, y)

a = int(input())
b = int(input())
(gcd_res, x, y) = gcd(a, b)
print(gcd_res, x, y)
# a * x + b * y = gcd_res
# -2 * 52 + 3 * 36 = 4
```

#stdin #stdout 0.03s 9880KB

stdin

```
52
36
```

stdout

```
4 -2 3
```

```cpp
#include <iostream>

using namespace std;

struct Erat {
    int gcd, x, y;
    Erat(int gcd, int x, int y) {
        this->gcd = gcd;
        this->x = x;
        this->y = y;
    }
};

Erat gcd(int a, int b) {
    if (a == 0) {
        return Erat(b, 0, 1);
    }
    Erat sol = gcd(b % a, a);
    return Erat(sol.gcd, sol.y - (b / a * sol.x), sol.x);
}

int main() {
    int a, b;
    cin >> a >> b;
    Erat sol = gcd(a, b);
    cout << sol.gcd << " " << sol.x << " " << sol.y << "\n";
}
// a * x + b * y = gcd_res
// -2 * 52 + 3 * 36 = 4
```

#stdin #stdout 0.01s 5316KB

stdin

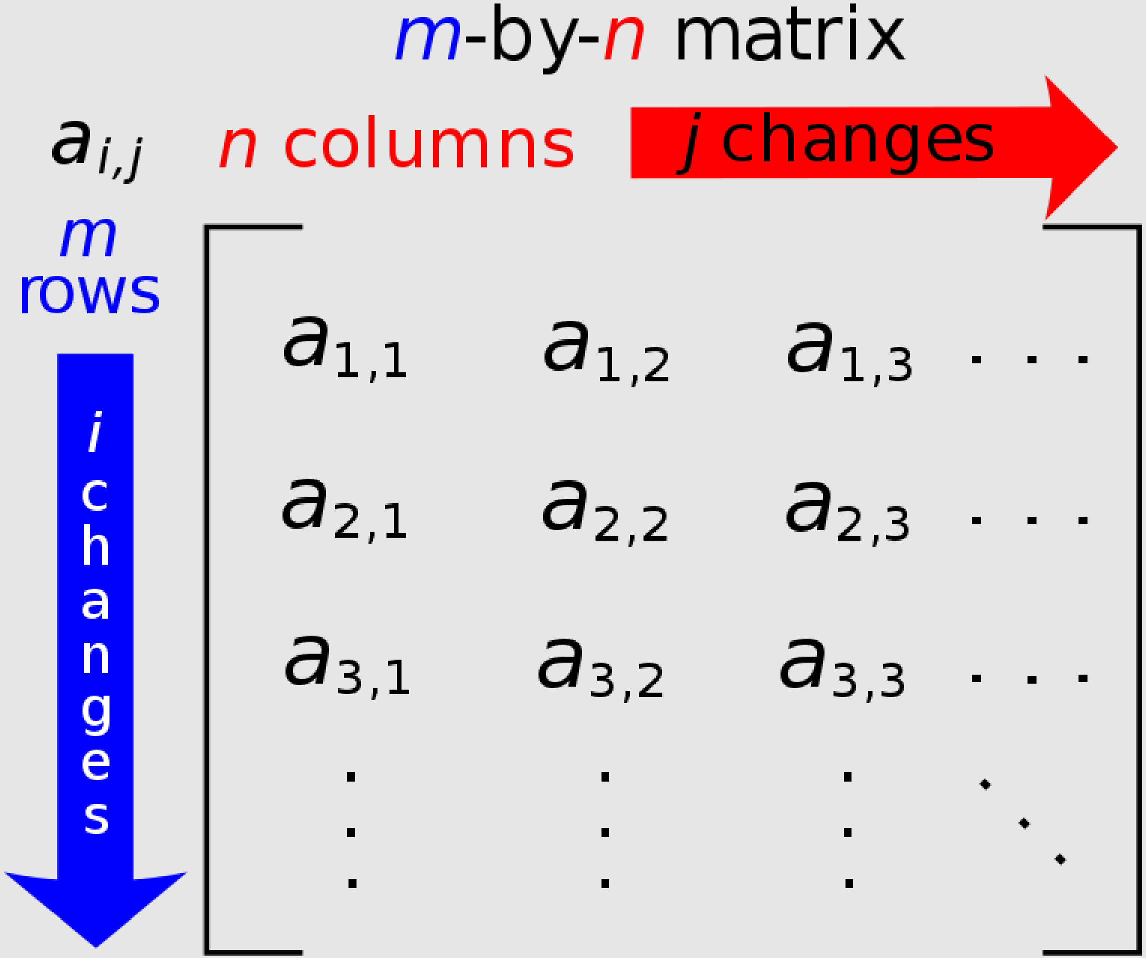
```
52
36
```

stdout

```
4 -2 3
```

How does division work?

- $A * (B^{-1}) = 1 \pmod{M}$
- $A * (B^{-1}) + M * K = 1$
- So we can divide numbers if $\gcd(A, M) = 1$


m-by-n matrix
a i,j
n columns
j changes
m rows
i changes
a 1,1 a 1,2 a 1,3
a 2,1 a 2,2 a 2,3
a 3,1 a 3,2 a 3,3

$$A + C = \begin{bmatrix} -5 & 2 & 0 \\ 7 & -3 & 4 \\ -1 & 3 & 2 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 8 \\ 6 & -14 & 2 \\ 9 & 5 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} (-5)+(0) & (2)+(-1) & (0)+(8) \\ (7)+(6) & (-3)+(-14) & (4)+(2) \\ (-1)+(9) & (3)+(5) & (2)+(1) \end{bmatrix}$$

$$A + C = \begin{bmatrix} -5 & 1 & 8 \\ 13 & -17 & 6 \\ 8 & 8 & 3 \end{bmatrix}$$

$$3 \cdot 0 + 1 \cdot 2 + 0 \cdot 0 = 2$$

$$\begin{bmatrix} 3 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 4 \\ 2 & 2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & \\ & \end{bmatrix}$$

$$2 \times 3 \qquad 3 \times 2 \qquad 2 \times 2$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \mathrm{x} \begin{bmatrix} 10 & 11 \\ 20 & 21 \\ 30 & 31 \end{bmatrix}$$

$$= \begin{bmatrix} 1\mathrm{x}10 + 2\mathrm{x}20 + 3\mathrm{x}30 & 1\mathrm{x}11 + 2\mathrm{x}21 + 3\mathrm{x}31 \\ 4\mathrm{x}10 + 5\mathrm{x}20 + 6\mathrm{x}30 & 4\mathrm{x}11 + 5\mathrm{x}21 + 6\mathrm{x}31 \end{bmatrix}$$

$$= \begin{bmatrix} 10+40+90 & 11+42+93 \\ 40+100+180 & 44+105+186 \end{bmatrix} = \begin{bmatrix} 140 & 146 \\ 320 & 335 \end{bmatrix}$$

$$A * (B * C) = (A * B) * C$$

matrix multiplication

associativity

$0 \quad 1 \quad 0 \quad -1 \quad 0 \quad 3 \quad 5$

e.g. $A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$, $B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

then $AB = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$

$BA = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$

$$\mathsf{A}^n = \underbrace{\mathsf{A} \cdots \mathsf{A}}_{n}.$$

```
class Matrix{
private:
    vector<vector<int> > elems;
public:

    int Rows() {
        return elems.size();
    }

    int Columns() {
        return elems[0].size();
    }

    Matrix(int n, int m) {
        elems.resize(n, vector<int>(m, 0));
    }

    int& operator ()(int i, int j) {
        return this->elems[i][j];
    }

    Matrix operator*(Matrix x) {
        Matrix c(this->Rows(), x.Columns());
        for (int i = 0; i < c.Rows(); i++) {
            for (int j = 0; j < c.Columns(); j++) {
                for (int k = 0; k < this->Columns(); k++) {
                    c(i, j) += elems[i][k] * x(k, j);
                }
            }
        }
        return c;
    }
};
```

```cpp
int main() {
    auto A = Matrix(2, 2), B = Matrix(2, 2);
    for (int i = 0; i < A.Rows(); i++) {
        for (int j = 0; j < A.Columns(); j++) {
            cin >> A(i, j);
        }
    }
    for (int i = 0; i < B.Rows(); i++) {
        for (int j = 0; j < B.Columns(); j++) {
            cin >> B(i, j);
        }
    }
    auto AB = A * B, BA = B * A;
    for (int i = 0; i < AB.Rows(); i++) {
        for (int j = 0; j < AB.Columns(); j++) {
            cout << AB(i, j) << " ";
        }
        cout << "\n";
    }
    for (int i = 0; i < BA.Rows(); i++) {
        for (int j = 0; j < BA.Columns(); j++) {
            cout << BA(i, j) << " ";
        }
        cout << "\n";
    }
    return 0;
}
```

Success #stdin #stdout 0.01s 5316KB comments (0)

stdin copy

```
1 0
0 -1
0 1
1 0
```

stdout copy

```
0 1
-1 0
0 -1
1 0
```

edit fork download template

```python
import numpy as np

a = np.matrix([[1, 0], [0, -1]])
b = np.matrix([[0, 1], [1, 0]])
print(a.dot(b))
```

Success #stdin #stdout 0.15s 28768KB

stdin

```
Standard input is empty
```

stdout

```
[[ 0  1]
 [-1  0]]
```

All codes

prime(c++) - https://ideone.com/gNpLk0

prime(python) - https://ideone.com/zGdLes

prime faster(c++) - https://ideone.com/a63wBm

prime faster(python) - https://ideone.com/f0FHSB

erat(c++) - https://ideone.com/KQMJMH

erat(python) - https://ideone.com/ytA3wV

bin power(c++) - https://ideone.com/GaqMlx

bin power(python) - https://ideone.com/FrheMb

reverse prime(c++) - https://ideone.com/p46I2U

reverse prime(python) - https://ideone.com/yGXF1y

gcd(python) - https://ideone.com/hZzvSi

gcd(c++) - https://ideone.com/an2IpZ

binary gcd(python) - https://ideone.com/6gD8at

binary gcd(c++) - https://ideone.com/vFvQCF

gcd_ext(python) - https://ideone.com/3Jz6iJ

gcd_ext(c++) - https://ideone.com/lFkCEz

matrix(c++) - https://ideone.com/rdsYJ7

matrix(python) - https://ideone.com/XAjvUS

That's All Folks!