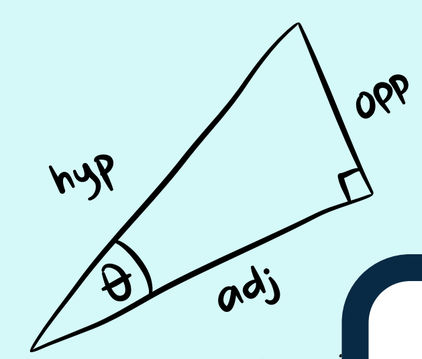
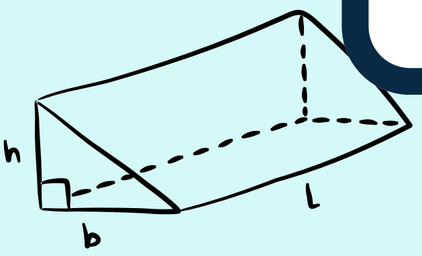
Test de primalidad de

FERMAT Y OTROS TESTS

Jose Angel Morales



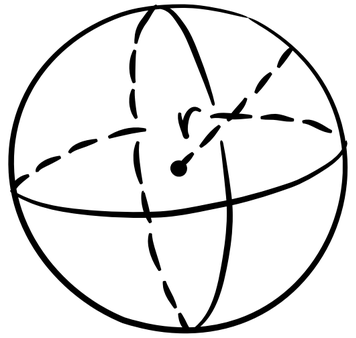


Un poco de historia

Desde la antigüedad los primos han sido un tema de sumo interés por la importancia que se veia en estos dado que los números se componen de factorización. Hoy en día también destacan por su importancia en ámbitos como la criptografía.

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$y = mx + b$$



$$V = \frac{4}{3}\pi r^3$$

Cribas

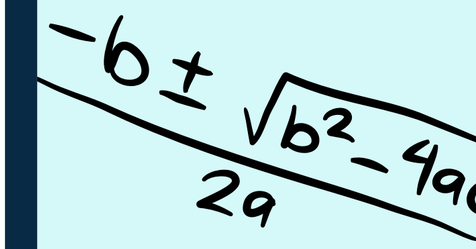
Una criba es un método que busca hallar los números primos menores a un numero n dado

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~

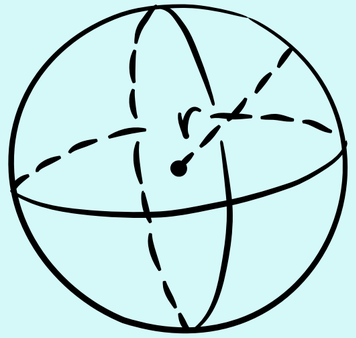
QUE SON LOS TEST DE PRIMALIDAD?

Pueden dividirse en 2 categorías:

- Test de primalidad
- Test de primalidad verdadero (pruebas de primalidad)



$y = mx + b$



$V = \frac{4}{3}\pi r^3$

CUAL ES LA DIFERENCIA?

Test

- Algoritmo probabilístico
- Velocidad alta
- Faciles de utilizar

Prueba

- Algoritmo determinístico
- Mas lento que el test según el algoritmo que se use (Agrawal-Kayal-Saxena)
- Mas complejos de utilizar

Agrawal-Kayal-Saxena (AKS)

Si n es primo entonces por el pequeno teorema de Fermat:

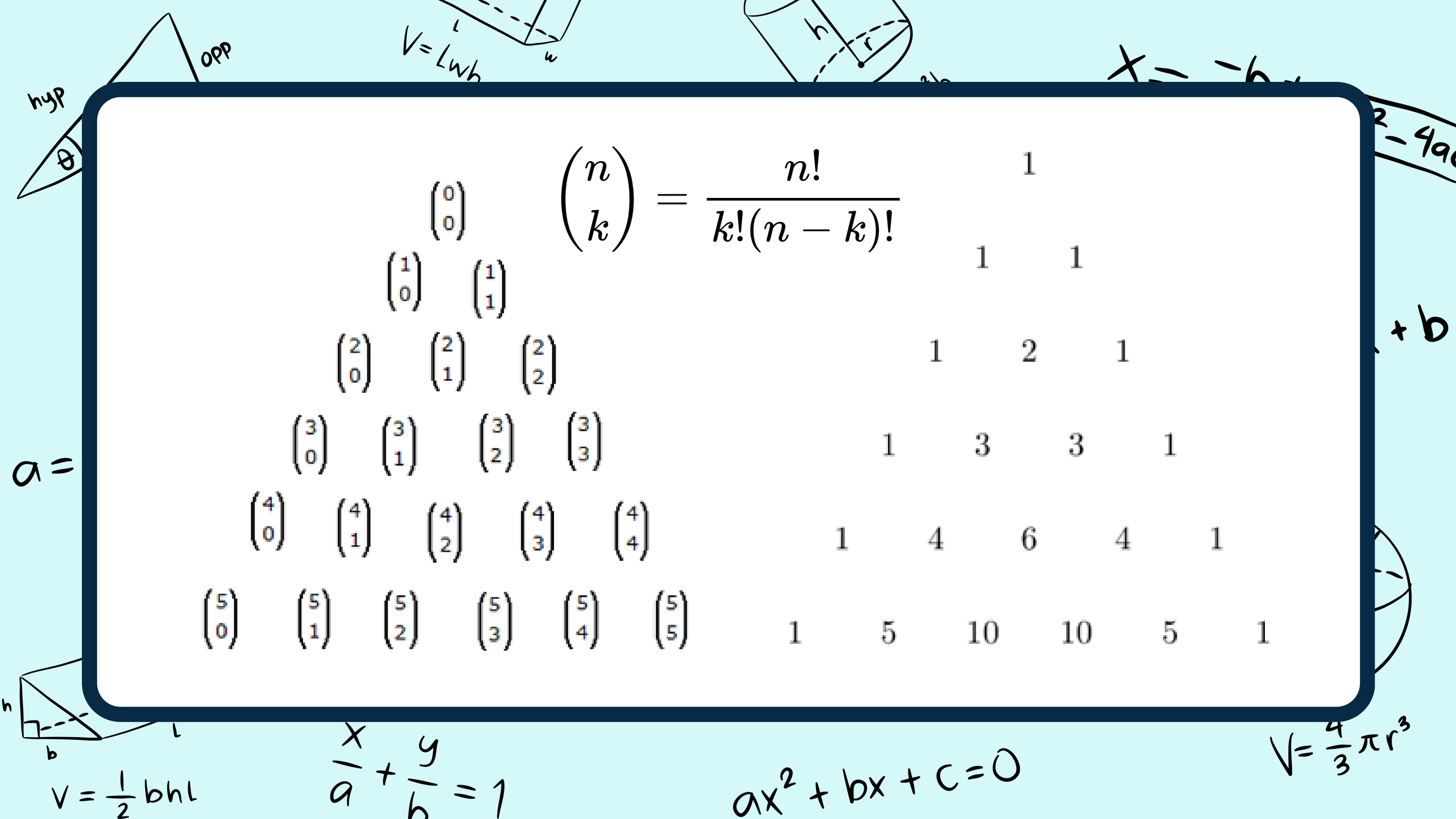
$$a^{n-1} \equiv 1 \pmod{n}$$

$$a^n \equiv a \pmod{n}$$

$$x^n + a^n \equiv x^n + a \pmod{n}$$

$$(x + a)^n \equiv x^n + a \pmod{n}$$

Para a $<$ n


$\binom{n}{k} = \frac{n!}{k!(n-k)!}$
$\binom{0}{0}$
$\binom{1}{0}\ \binom{1}{1}$
$\binom{2}{0}\ \binom{2}{1}\ \binom{2}{2}$
$\binom{3}{0}\ \binom{3}{1}\ \binom{3}{2}\ \binom{3}{3}$
$\binom{4}{0}\ \binom{4}{1}\ \binom{4}{2}\ \binom{4}{3}\ \binom{4}{4}$
$\binom{5}{0}\ \binom{5}{1}\ \binom{5}{2}\ \binom{5}{3}\ \binom{5}{4}\ \binom{5}{5}$
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1

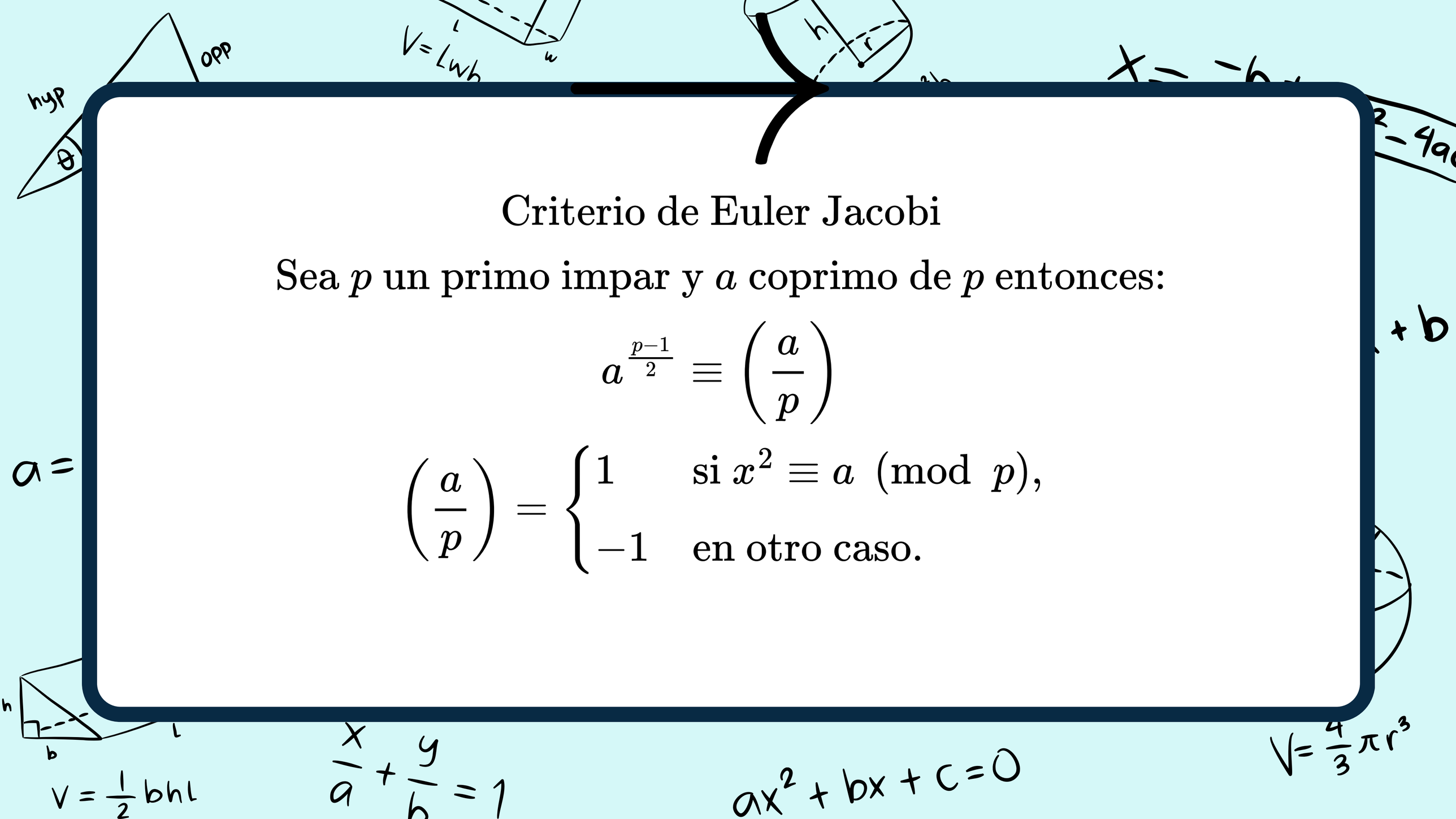
Puntos clave

- Primera prueba de primalidad con tiempo de computo polinomial $O(log^{12}n) \rightarrow O(log^{6}n)$
- Pierde eficacia con números muy grandes

Prueba de Pepin

Si $F_n = 2^{2^n}$ es el n-esimo numero de Fermat entonces:

$$F_n \text{ es primo ssi } 3^{\frac{F_n - 1}{2}} \equiv -1 \pmod{F_n}$$

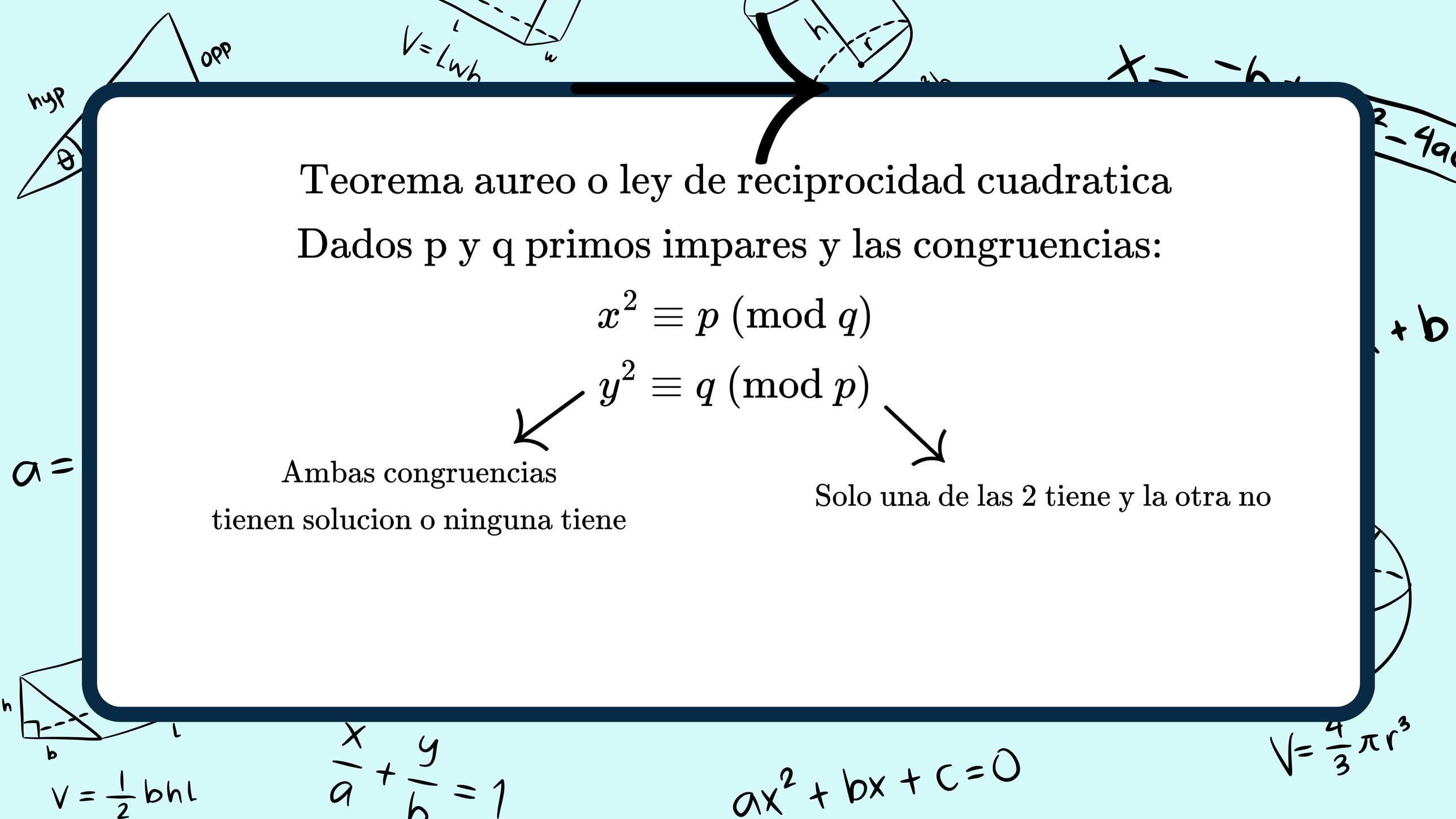


Criterio de Euler Jacobi

Sea p un primo impar y a coprimo de p entonces:

$$a^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right)$$

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{si } x^2 \equiv a \pmod{p}, \\ -1 & \text{en otro caso.} \end{cases}$$



Teorema aureo o ley de reciprocidad cuadratica

Dados p y q primos impares y las congruencias:

$$x^2 \equiv p \pmod{q}$$

$$y^2 \equiv q \pmod{p}$$

Ambas congruencias tienen solucion o ninguna tiene

Solo una de las 2 tiene y la otra no

Supongamos que F_n no es primo y $3^{\frac{F_n-1}{2}} \equiv -1 \pmod{F_n}$.

Entonces sea p factor primo de F_n, esto implica que

$$3^{\frac{F_n-1}{2}} \equiv -1 \pmod{p} \rightarrow 3^{F_n-1} \equiv 1 \pmod{p}.$$

Entonces $O_p(3) | F_n - 1$, pero $O_p(3) \nmid \dfrac{Fn-1}{2} \rightarrow$

$$O_p(3) = F_n - 1 \ (\rightarrow\leftarrow).$$

Puntos clave

- Algoritmo teóricamente rápido
- Pierde eficacia rápidamente debido a la naturaleza de los numeros de Fermat

$$F_0 = 3, F_1 = 5, F_2 = 17, F_3 = 257, F_4 = 65537$$

$$F_5 = 4294967297 = 641 \times 6700417$$

Para F_3 el algoritmo debe calcular:

$$3^{\frac{257-1}{2}} = 3^{128} \pmod{257}$$

Test de primalidad de Fermat

Recordando:

Si p es primo que no divide a a entonces

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

Orden de complejidad: $\mathcal{O}\left(k \times \log^2 n \times \log\log n \times \log\log\log n\right)$

Entrada: Un número natural $n > 1$, el número k de veces que se ejecuta el test y nos determina la fiabilidad del test.
Salida: COMPUESTO si n es compuesto y POSIBLE PRIMO si n es un posible primo.

1. Para j desde 1 hasta k haga lo siguiente:
   (a) $a \leftarrow$ Función Genera_número_aleatorio_en_intervalo $[2, n-2]$
   (b) Si $a^{n-1} \not\equiv 1 \pmod{n}$ entonces:
      i. Retorne COMPUESTO
2. Retorne POSIBLE PRIMO

```python
for i in range(10):
    n = random.randint(10**5, 10**6)
    if prime_fermat(n, 20):
        print(f"{n} es probablemente primo")
    else:
        print(f"{n} es compuesto")
```

Python

```
560797 es probablemente primo
744919 es compuesto
573295 es compuesto
819586 es compuesto
351976 es compuesto
945345 es compuesto
623420 es compuesto
384192 es compuesto
539665 es compuesto
347187 es compuesto
```

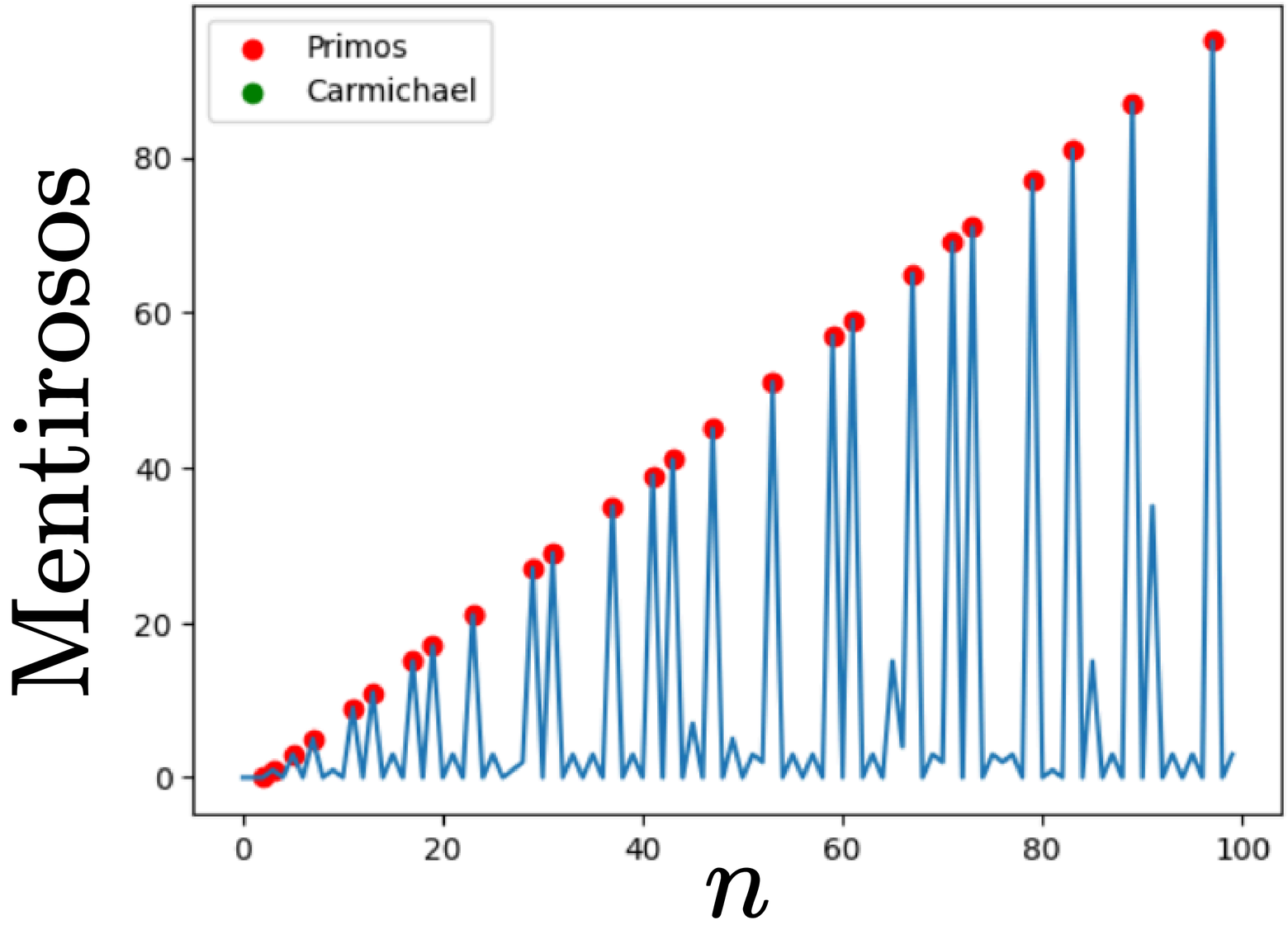
```python
    if prime_fermat(560797,100000):
        print("560797 es probablemente primo")
    else:
        print("560797 es compuesto")
```

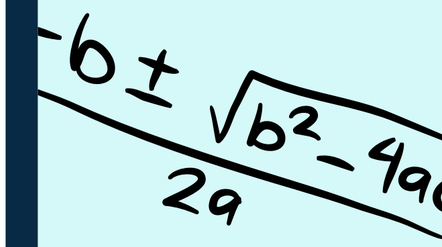
[16] ✓ 0.2s Python

```
560797 es probablemente primo
```

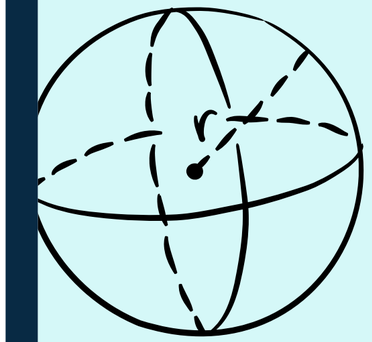
https://numerosprimos.org/numeros-primos-menores-de-1-a-1000000

Mentirosos de Fermat



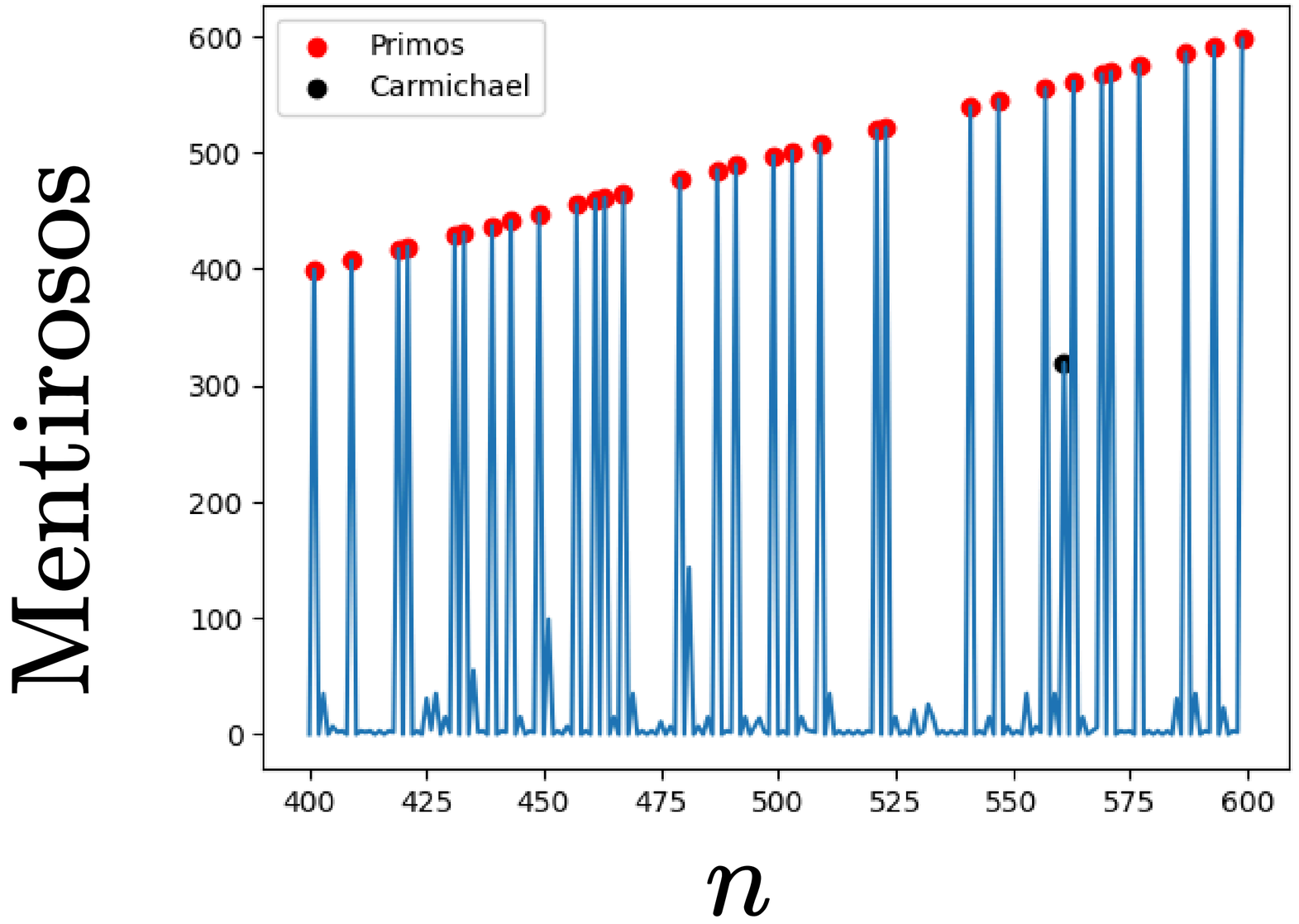


$y = mx + b$



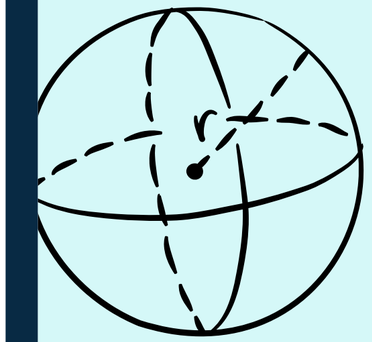
$V = \frac{4}{3}\pi r^3$

Mentirosos de fermat



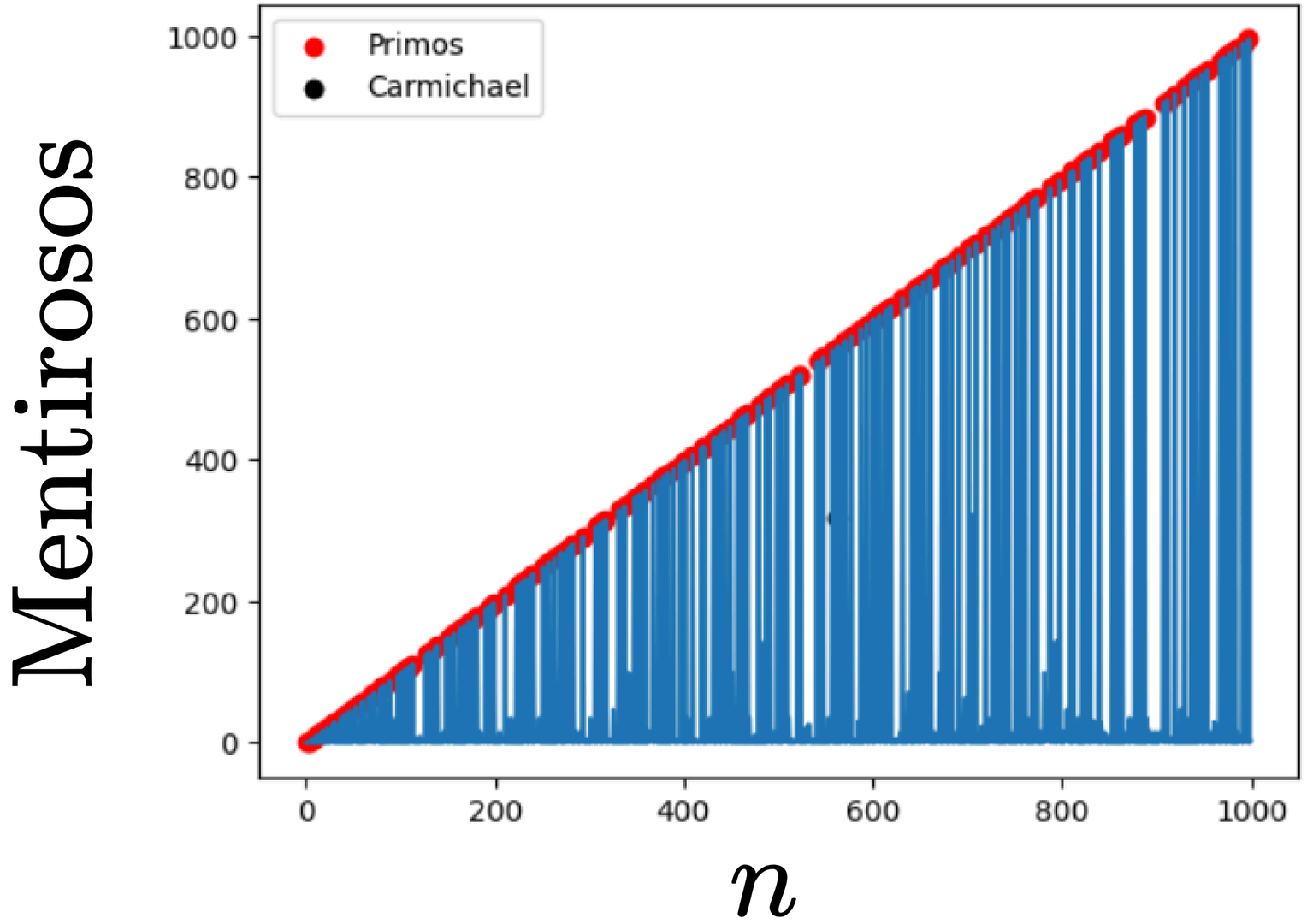
$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$y = mx + b$$



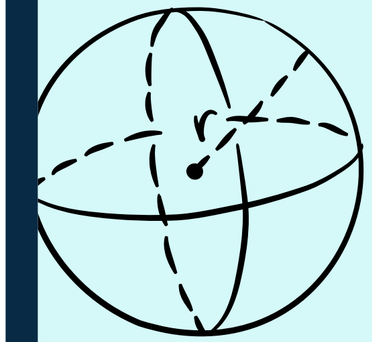
$$V = \frac{4}{3}\pi r^3$$

Mentirosos de Fermat



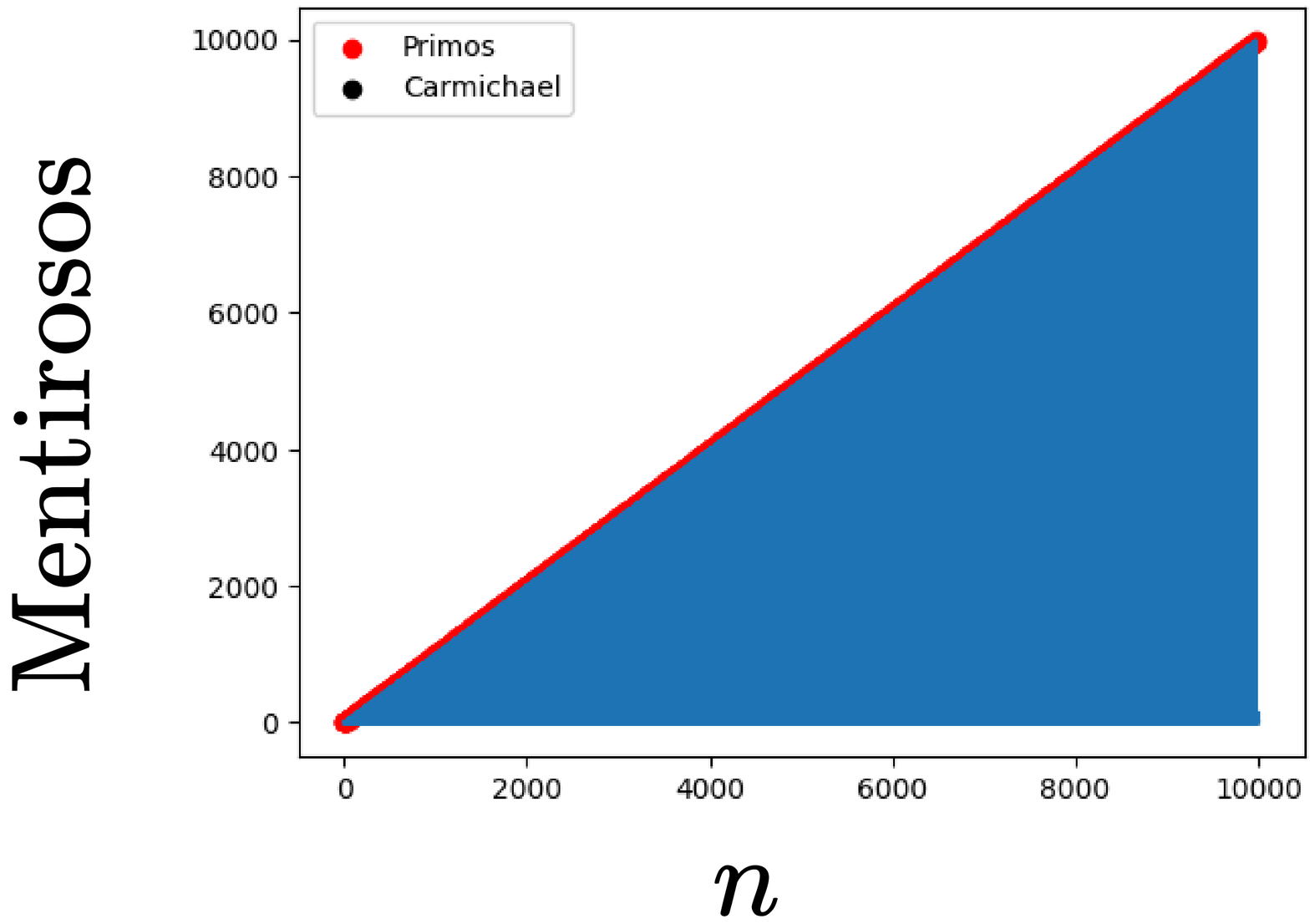
$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

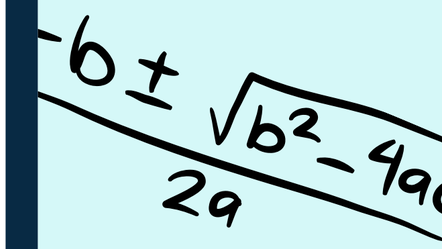
$$y = mx + b$$



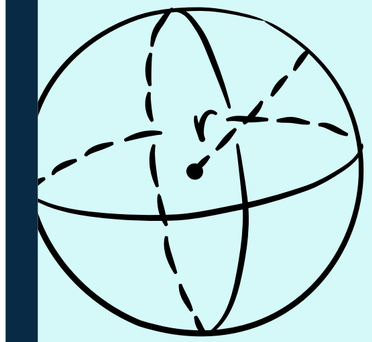
$$V = \frac{4}{3}\pi r^3$$

Mentirosos de Fermat





$y = mx + b$



$V = \frac{4}{3}\pi r^3$

Error y carmichael

```
Número de mentirosos de 561: 318 (no toma en cuenta el 1 ni n-1)
Total de coprimos con n (ϕ(561)): 320
La probabilidad de que se elija un mentiroso es: 0.5698924731182796
```

$k = 1$

```
Probando el número de Carmichael 561
Probabilidad de compuesto: 0.430395
Probabilidad de primo: 0.569605
```

$k = 10$

```
Probando el número de Carmichael 561
Probabilidad de compuesto: 0.9965
Probabilidad de primo: 0.0035
```

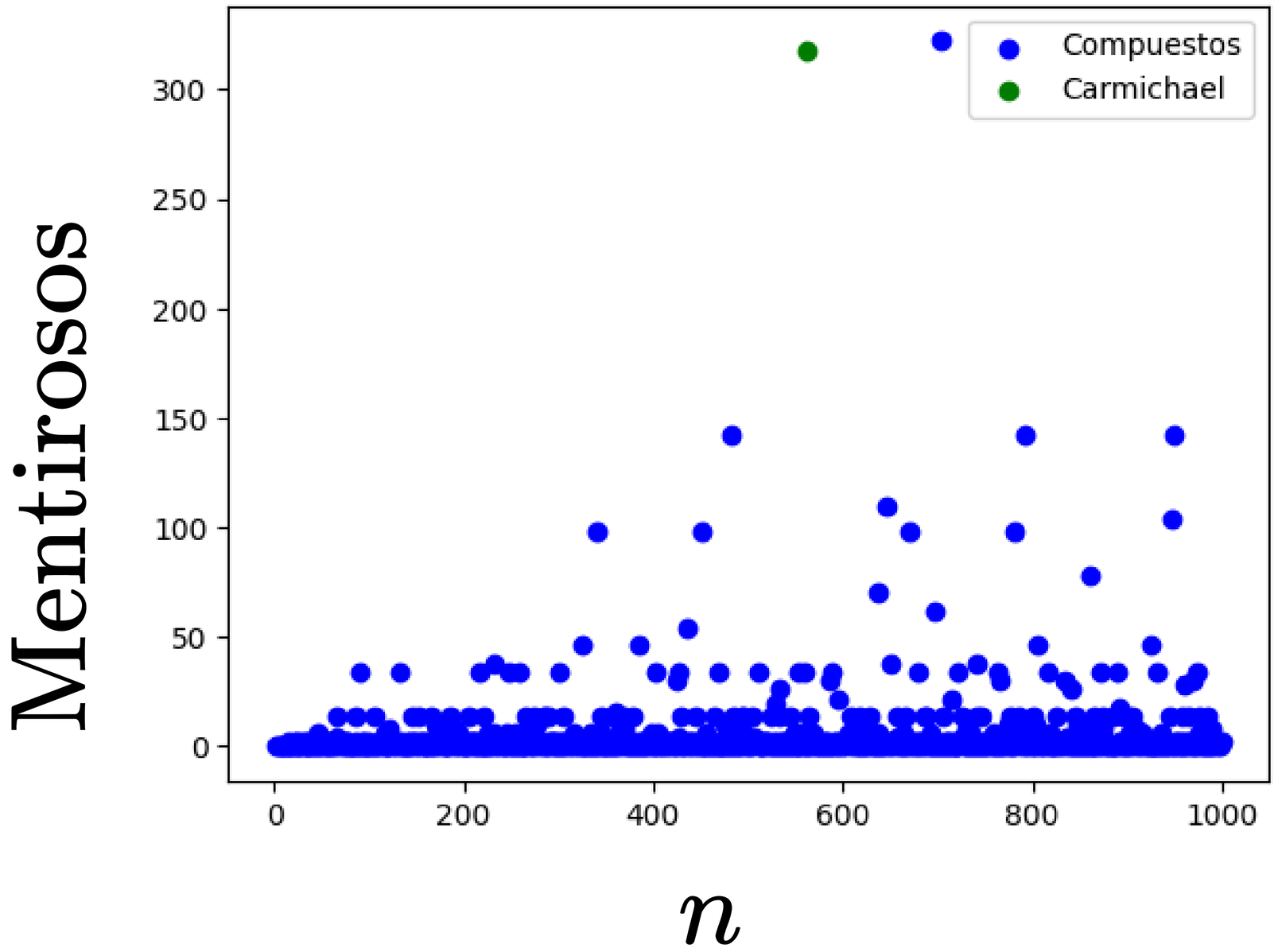
$k = 1$

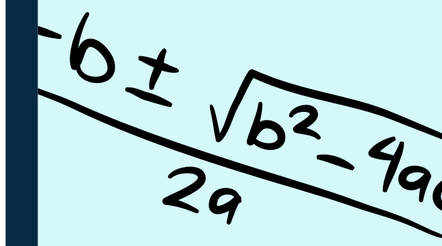
```
Probando el número de Carmichael 46657
Probabilidad de compuesto: 0.111112
Probabilidad de primo: 0.888888
```

$k = 10$

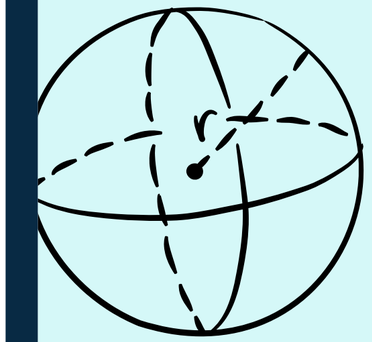
```
Probando el número de Carmichael 46657
Probabilidad de compuesto: 0.691766
Probabilidad de primo: 0.308234
```

703 tiene 322 mentirosos pero no es de Carmichael



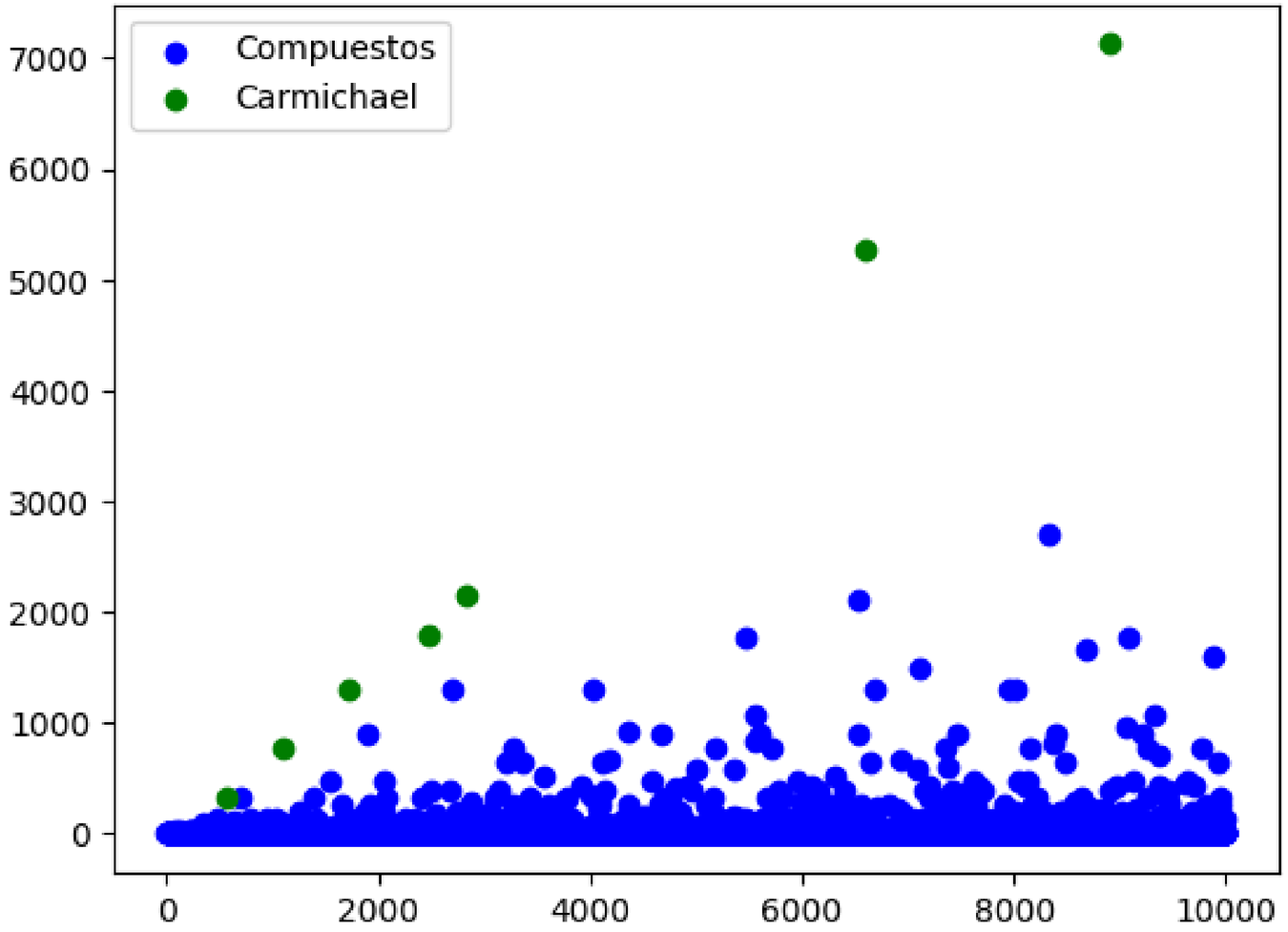


$y = mx + b$

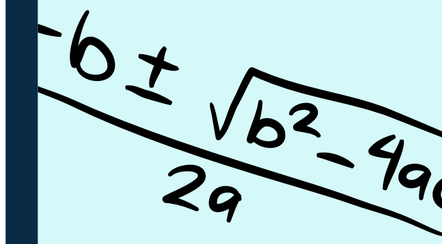


$V = \frac{4}{3}\pi r^3$

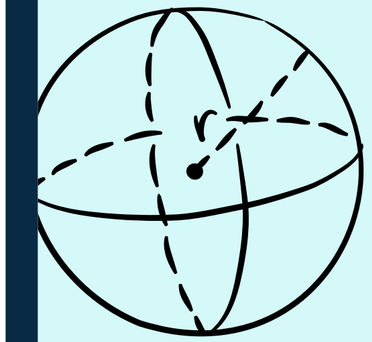
Mentirosos



n



$y = mx + b$



$V = \frac{4}{3}\pi r^3$

Test de primalidad de Miller-Rabbin

Vamos a decir que n es un primo probable fuerte para la base a si:

$$a^d \equiv 1 \pmod{n} \text{ o}$$

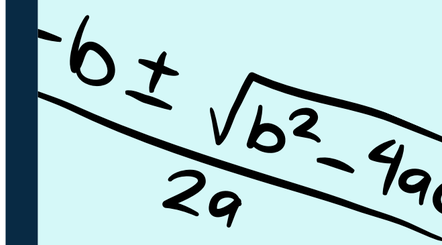
$$a^{2^r d} \equiv -1 \pmod{n}, \text{ para } 0 \leq r < s$$

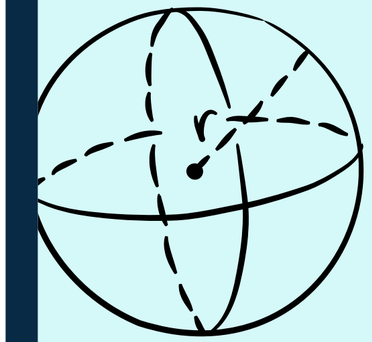
Algoritmo test de primalidad de Miller-Rabin (Orden de complejidad $\mathcal{O}(k \times (\log n)^3)$)

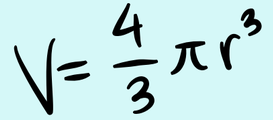
Entrada: Un número natural $n > 2$ (impar), el número k de veces que se ejecuta el test.

Salida: COMPUESTO si n es compuesto y POSIBLE PRIMO si n es un posible primo.

1. Escribir $n - 1 = 2^r \cdot d$ con d impar.
2. Para j desde 1 hasta k haga lo siguiente:
   (a) $a \leftarrow$ Función Genera_número_aleatorio_en_intervalo$(2, n - 2)$
   (b) $x \leftarrow a^d \bmod n$
   (c) Si $x = 1$ o $x = n - 1$ entonces continuar con la siguiente iteración.
   (d) Para i desde 1 hasta $r - 1$ haga:
      i. $x \leftarrow x^2 \bmod n$
      ii. Si $x = n - 1$ entonces detener el ciclo interior y continuar con la siguiente iteración externa.
   (e) Si $x \neq n - 1$ al final del ciclo, entonces:
      i. Retorne COMPUESTO.
3. Retorne POSIBLE PRIMO.







Mentirosos fuertes

Si n es compuesto y a es una base para la cual

n es primo probable fuerte decimos que

n es pseudoprimo fuerte base a

y decimos que a es un mentiroso fuerte.

(2047 es pseudoprimo fuerte base 2)

Mentirosos fuertes

2047 es pseudoprimo fuerte base 2

```
pow(2, 1023, 2047)
```

```
[37]  ✓  0.0s
```

```
...  1
```

ERROR DE MILLER

```
Para n=15, el numero de mentirososo fuertes: 2, phi(n): 8, proporcion: 0.25
Para k=1, error approx: 0.250000, (1/4)^k: 0.250000
Para k=2, error approx: 0.062500, (1/4)^k: 0.062500
Para k=3, error approx: 0.015625, (1/4)^k: 0.015625
Para k=4, error approx: 0.003906, (1/4)^k: 0.003906
Para k=5, error approx: 0.000977, (1/4)^k: 0.000977

Para n=57, el numero de mentirososo fuertes: 2, phi(n): 36, proporcion: 0.055555555555555555
Para k=1, error approx: 0.055556, (1/4)^k: 0.250000
Para k=2, error approx: 0.003086, (1/4)^k: 0.062500
Para k=3, error approx: 0.000171, (1/4)^k: 0.015625
Para k=4, error approx: 0.000010, (1/4)^k: 0.003906
Para k=5, error approx: 0.000001, (1/4)^k: 0.000977
```