Enigma

Enigma cipher machine has several rotors, and each of them has a value. Each value pair of two rotors should be coprime. To develop such a cipher, can you calculate how many ordered coprime integer pairs are there in a given range?

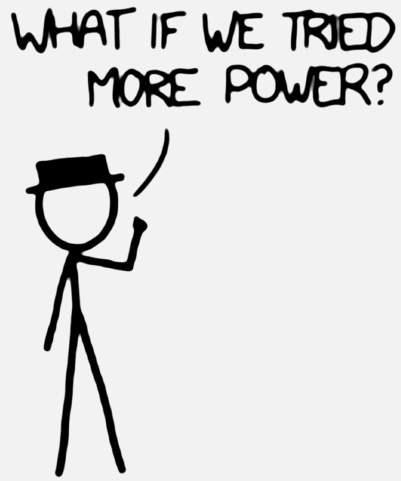
Naive Approach

```
for i = lo, lo + 1, ..., hi
    for j = lo, lo + 1, ..., hi
        if gcd(i, j) == 1
            count += 1
```

Analysis

```
for i = <hi - lo + 1 times>
    for j = <hi - lo + 1 times>
        if <log(min(i, j))-ish stuff>
            count += 1
```

Time complexity: $O((hi - lo)^2 log(hi))$


WHAT IF WE TRIED
MORE POWER?

Observation

42 = 2 * 3 * 7

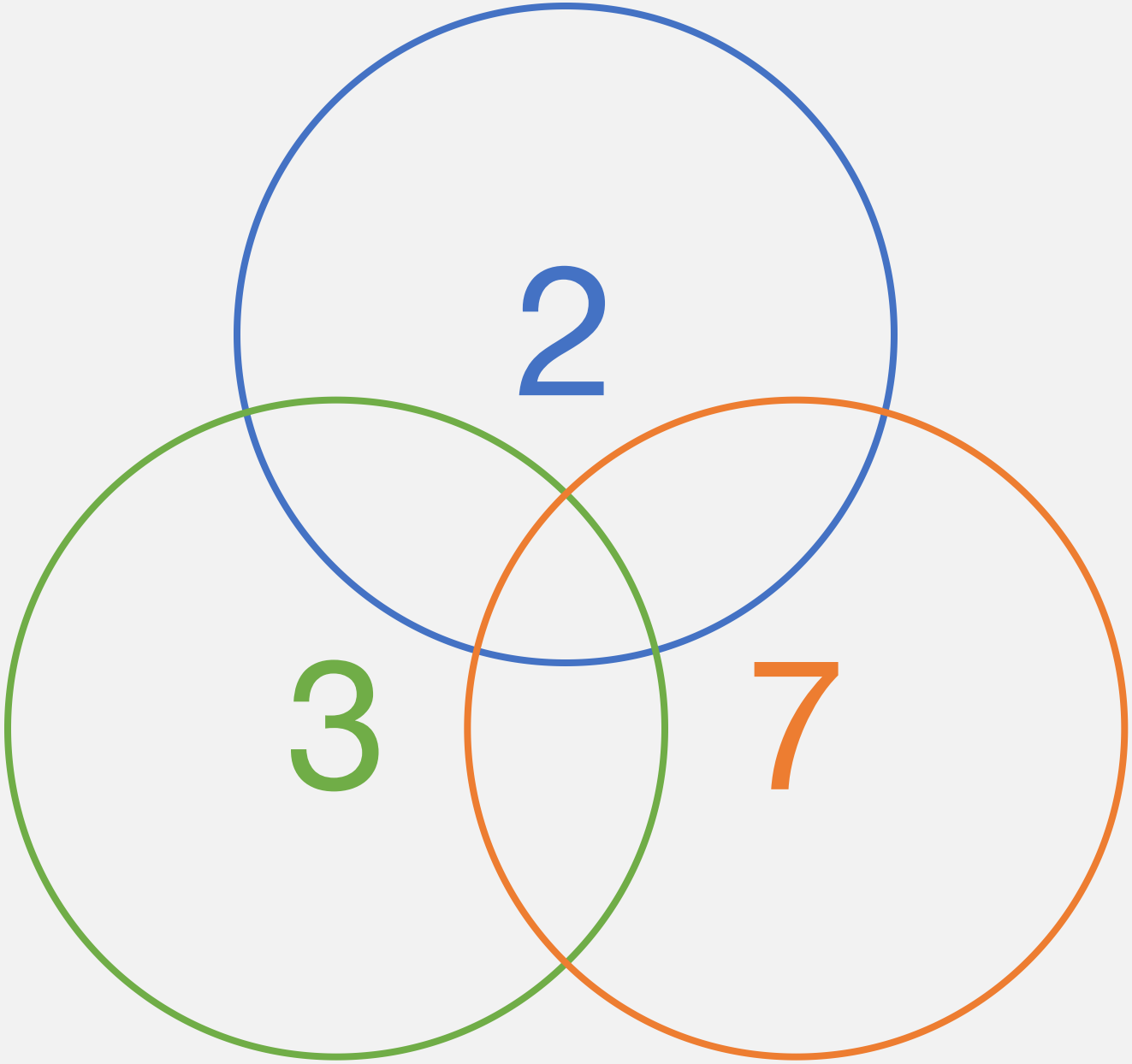
If number X and 42 are coprime...

Observation

$42 = 2 * 3 * 7$

If number X and 42 are not coprime

$X = 2A$ or $X = 3B$ or $X = 7C$


2
3
7

Inclusion-exclusion Principle

let f(<...>) =

number of integers that divisible by <...> within the range lo..hi

f(a or b) =

f(a) + f(b) - f(a and b)

Inclusion-exclusion Principle

f(a or b or c) =

f(a) + f(b) + f(c)

- f(a and b) - f(a and c) - f(b and c)

+ f(a and b and c)

Inclusion-exclusion Principle

f(at least one member in set S*) =

sum of f(all member in S)

where S is an odd-size subset of S*

- sum of f(all member in S)

where S is an even-size subset of S*

How to Calculate f(...) Components

For all member in S:

f(2, 3, and 7) = f(2 * 3 * 7) = f(42)

f(all member in S) = f(product of S)

For a single integer:

f(42) = floor(hi / 42) - floor((lo - 1) / 42)

f(x) = floor(hi / x) - floor((lo - 1) / x)

I-e Principle Approach

```
for i = lo, lo + 1, ..., hi
    S* = factor(i)
    foreach S = subset of S*
        x = PI(S)
        if |S| % 2 == 1
            count += floor(hi / x)
                - floor((lo - 1) / x)
        else
            count -= floor(hi / x)
                - floor((lo - 1) / x)
```

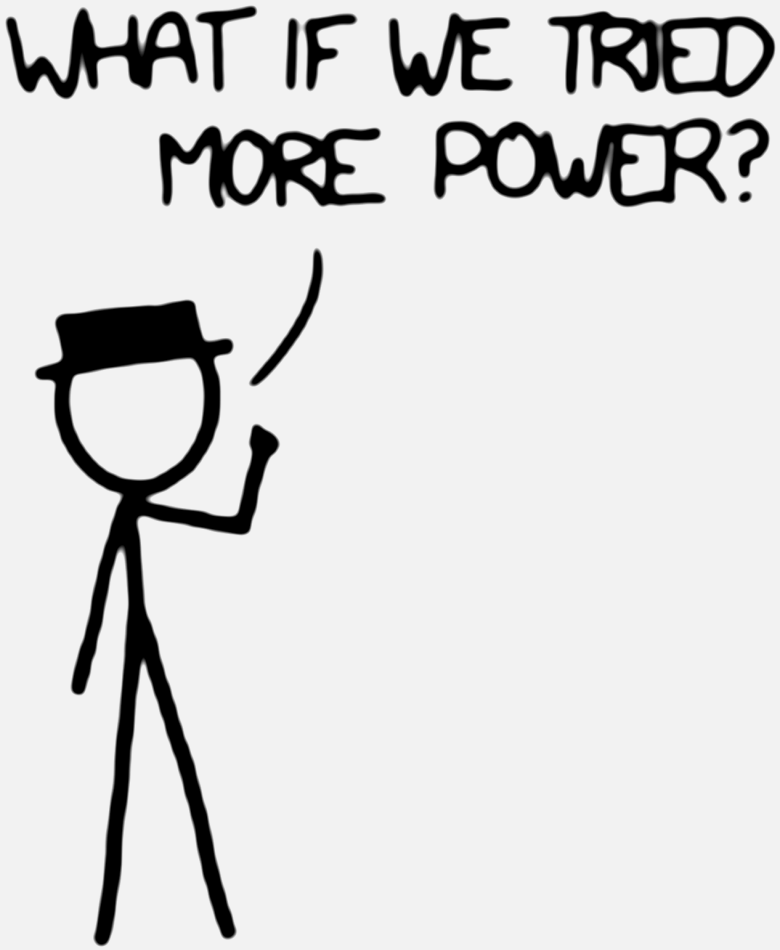
Analysis

```
for i = <hi - lo + 1 times>
    <sqrt(i)-ish stuff>
    foreach S = <"count of integers
        divisible by i" times>
            <|S|-ish stuff>
```

Analysis

```
for i = <hi - lo + 1 times>
    <sqrt(i)-ish stuff>
    foreach S = <1 + 0.5 + 0.333
        + ... = log(i)>
            <log(log(i))-ish stuff>
```

Time complexity: $O((hi - lo)^{1.5})$


WHAT IF WE TRIED
MORE POWER?

Dynamic Programming

Factor Tree

2 -> 1

3 -> 1

4 -> 2 -> 1

6 -> 2 -> 1

42 -> 6 -> 2 -> 1

...

Factor Tree

```
for i = lo, lo + 1, ..., hi
    if factor[i] is null
        fill(i)
```

Factor Tree

```
procedure fill(i)
    for j = i, i * i, ..., hi
        for k = 1, 2, ..., hi / j
            if factor[k] < i
                put_node(i, j, k)
```

Factor Tree

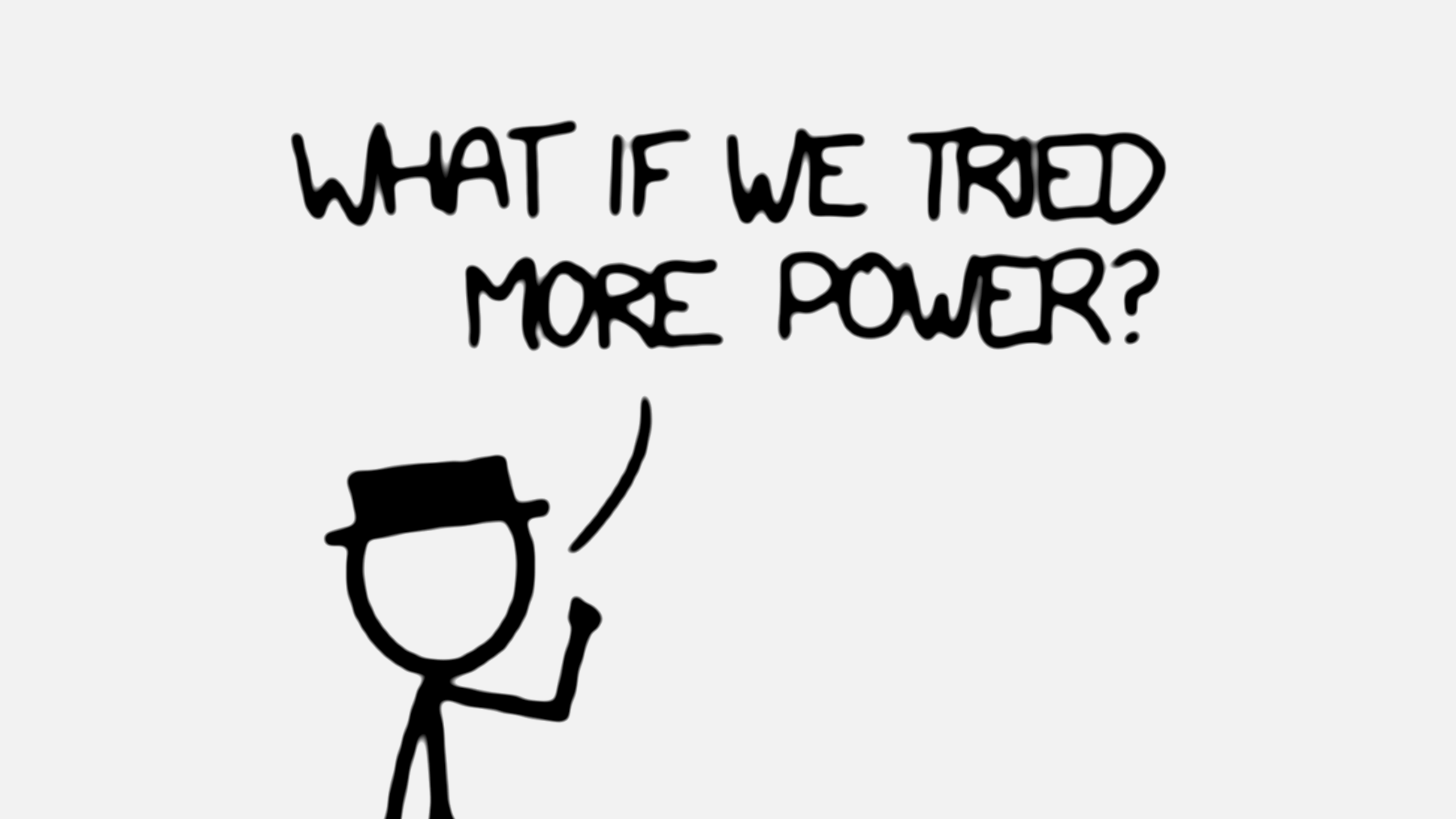
```
procedure put_node(i, j, k)
    factor[k * j] = i
    remain[k * j] =
        remain[k] * factor[k]
```

Analysis

```
for i = <hi - lo + 1 times>
    <sqrt(i)-ish stuff>
    foreach S = <1 + 0.5 + 0.333
        + ... = log(i)>
            <log(log(i))-ish stuff>
```

Time complexity:

O(hi log(hi)log(log(hi)))


WHAT IF WE TRIED MORE POWER?

Factor Tree

42 -> 6 -> 2 -> 1

84 -> 12 -> 4 -> 2 -> 1

126 -> 24 -> 8 -> 4 -> 2 -> 1

...

Factor Tree

42 -> 6 -> 2 -> 1

~~84 -> 12 -> 4 -> 2 -> 1~~

~~126 -> 24 -> 8 -> 4 -> 2 -> 1~~

...

Analysis

Build the tree: O(hi log(log(hi)))

Number of nodes: O(hi / log(hi))

I-e Principle: O(log(hi)log(log(hi)))

(per node)

Comprehensive: O(hi log(log(hi)))