what are some properties related to division multiplication and other with respect to xor used in cp

Here are the **XOR properties that actually matter in Competitive Programming**, especially when combined with **addition / subtraction / multiplication / division**.
(Short, first-principles, CP-useful only.)

Core XOR properties (must-know)

1. **Self-inverse**

```nginx
a ⊕ a = 0
a ⊕ 0 = a
```

2. **Commutative & Associative**

```r
a ⊕ b = b ⊕ a
(a ⊕ b) ⊕ c = a ⊕ (b ⊕ c)
```

3. **Prefix trick**

```nginx
a ⊕ b ⊕ b = a
```

→ Used in prefix XOR, range XOR queries.

XOR vs Addition / Subtraction

✕ No linear relation

```css
```

Copy code

```
a ⊕ b ≠ a + b
a ⊕ b ≠ a − b
```

✔ Bit-level relation

nginx

```
a + b = (a ⊕ b) + 2·(a & b)
```

Copy code

📌 Very important
→ explains carry generation.

Consequences

- If `(a & b) == 0` → `a + b = a ⊕ b`
- Used in:
  - carry-free addition
  - bitmask DP
  - constraints like `a + b = a ⊕ b`

XOR vs Multiplication

✕ No distributive property

css

```
a ⊕ (b × c) ≠ (a ⊕ b) × (a ⊕ c)
```

Copy code

✔ Special cases

perl

```
a ⊕ (2a) = left-shift effect
```

Copy code

Power of 2 trick

bash

```
a × 2^k = a << k
```

Copy code

→ XOR unaffected by left shift positions unless overlapping bits.

XOR vs Division

✗ No relation

```css
(a ⊕ b) / c has no simplification
```

Division destroys bit patterns → avoid combining with XOR unless brute forcing bits.

XOR vs AND / OR (VERY IMPORTANT)

Identity

```nginx
a + b = (a ⊕ b) + 2(a & b)
```

Bounds

```less
a ⊕ b ≤ a + b
a ⊕ b ≥ |a − b|
```

Useful identities

```nginx
a | b = (a ⊕ b) + (a & b)
```

XOR patterns used in CP

1️⃣ XOR from 1 to n

```matlab
f(n) = 1 ⊕ 2 ⊕ ... ⊕ n

n % 4 == 0 → n
```

```
n % 4 == 1 → 1
n % 4 == 2 → n + 1
n % 4 == 3 → 0
```

Copy code

Used in:

- range XOR
- missing number
- parity games

2️⃣ XOR in ranges

powershell

```powershell
XOR(l, r) = f(r) ⊕ f(l − 1)
```

Copy code

3️⃣ XOR invariants

- XOR preserves **parity of bits**
- XOR cancels duplicates
  → Used in:
- odd occurrence problems
- multiset equivalence
- Grundy / Nim

XOR in Modular Arithmetic

✕ XOR ≠ mod

css

```css
(a ⊕ b) % m ≠ (a % m) ⊕ (b % m)
```

Copy code

Never mix blindly.

XOR in Bitwise DP / Greedy

Key facts:

- XOR works **bit independently**
- MSB decides magnitude
- Greedy from MSB → LSB

Used in:

- max XOR pair
- trie problems
- minimizing `(a ⊕ x)`

XOR & Comparison

```css
Maximize a ⊕ b → differ at highest bit
Minimize a ⊕ b → match highest bits
```

Copy code

Summary Table

Operation	Works with XOR?	Note
`+`	⚠️ Partial	carry via `&`
`-`	×	no structure
`×`	×	except shifts
`/`	×	destroys bits
`&`	✅	carry relation
`	`	✅
`%`	×	unsafe

CP Golden Rules

- Think **bitwise**, not arithmetic
- If addition involved → check carries (`&`)
- If division involved → avoid XOR

- XOR problems → **prefix, parity, highest bit**

If you want, I can give:

- **XOR problem patterns**
- **Tricks used in CF/LeetCode**
- **XOR + DP**
- **XOR + greedy proofs**

Just tell me which.

not these, properties related to divison x^y

Got it — you mean **exponentiation properties** (`x^y`) **with division / multiplication**, **not XOR**.
Here are the **exact CP-relevant laws**, clean and precise.

Core exponent properties (division & multiplication)

1️⃣ Same base, multiply

$$x^a \cdot x^b = x^{a+b}$$

2️⃣ Same base, divide

$$\frac{x^a}{x^b} = x^{a-b} \quad (x \neq 0)$$

📌 **Most used identity in CP**

3️⃣ Power of a power

$$(x^a)^b = x^{a \cdot b}$$

4️⃣ Different bases, same power

$$x^a \cdot y^a = ($$

no related to division

Understood 👍

You want **only division-related properties of exponentiation** (`x^y`) — **no XOR, no multiplication extras**.

Here are the **only valid & CP-useful division rules**, from first principles.

1️⃣ Same base division (MOST IMPORTANT)

$$\frac{x^a}{x^b} = x^{a-b} \quad (x \neq 0)$$

Examples:

- $\frac{2^7}{2^3} = 2^4$
- $\frac{10^5}{10^2} = 10^3$

2️⃣ Same exponent division

$$\frac{x^a}{y^a} = \left(\frac{x}{y}\right)^a \quad (y \neq 0)$$

Example:

- $\frac{8^2}{2^2} = (4)^2 = 16$

Used when **exponents match**.

3️⃣ Negative exponent via division

$$x^{-a} = \frac{1}{x^a}$$

So,

$$\frac{1}{x^{-a}} = x^a$$

Example:

- $2^{-3} = \frac{1}{8}$

4️⃣ Fractional exponents (roots via division)

$$x^{\frac{a}{b}} = \sqrt[b]{x^a}$$

Example:

- $16^{1/2} = 4$
- (27^{2/3} = (\sqrt[3]{

i want xor property related to division

Short answer (important for CP):

> **There is NO direct algebraic property linking XOR and division.**
> But there are **a few very specific, safe facts & edge cases** you *can* use.

1️⃣ XOR is not compatible with division (core fact)

In general:

```css
(a ⊕ b) / c   ≠   (a / c) ⊕ (b / c)
a ⊕ (b / c)   ≠   (a ⊕ b) / c
```

👉 **Division destroys bit structure**, XOR works purely on bits.

So **never distribute / factor division over XOR**.

2️⃣ Power-of-2 division (ONLY useful case)

Division by power of 2 is bit-shift:

```bash
a / 2^k = a >> k   (integer division)
```

Now XOR interaction:

```arduino
(a ⊕ b) >> k = (a >> k) ⊕ (b >> k)   ❌ generally false
```

But ✔ **TRUE if lower k bits of both are zero**:

```markdown
if (a % 2^k == 0 && b % 2^k == 0):
    (a ⊕ b) / 2^k = (a / 2^k) ⊕ (b / 2^k)
```

📌 CP use: problems that say

> “numbers are multiples of 2^k”

3️⃣ XOR parity vs division

Lowest bit:

```nginx
a ⊕ b = odd   ⇔   a and b have different parity
```

Division effect:

```css
(a / 2) loses parity info
```

So:

```bash
(a ⊕ b) % 2 != ((a/2) ⊕ (b/2)) % 2
```

👉 **Division kills XOR parity invariants**

4️⃣ XOR & modulo (related to division)

Since:

```bash
a % m = a − m ⌊a/m⌋
```

XOR does NOT commute:

```css
(a ⊕ b) % m ≠ (a % m) ⊕ (b % m)
```

Exception:

```lua
m = 2^k
a % 2^k = lower k bits
```

Then:

```matlab
(a ⊕ b) % 2^k = (a % 2^k) ⊕ (b % 2^k)
```

📌 Very important in bitmask problems.

5️⃣ Division-based identities that DO NOT exist

❌ These are **always wrong**:

```r
(a ⊕ b) / a = b
(a ⊕ b) / b = a
(a / b) ⊕ c = a / (b ⊕ c)
```

If you see something like this → **reject immediately**.

6️⃣ CP Golden Rule (memorize)

> **XOR respects only:**

- bit positions
- powers of two
- exact bit alignment

> **Division respects only:**

- magnitude
- quotient, not bits

👉 Mixing them is almost always invalid.

One-line summary (exam / CP ready)

> XOR has **no distributive or inverse relation** with division,
> except in **power-of-2 aligned cases** where division is just a shift.

If you want, I can give:

- CF problems where this trap appears
- When setters *intentionally* mix XOR + division
- Proof-based explanation (bit-level)

Just tell me.

just give me list of correct propertiese

Here is a **clean list of ONLY correct properties** involving **XOR (⊕)** and **division (/)**.
No explanations, no false ones — **just facts** (CP-safe).

ChatGPT 5.2

1. **No distributive property**

$$(a \oplus b)/c \neq (a/c) \oplus (b/c)$$

2. **No inverse relation**

$$(a \oplus b)/a \; \neq \; b$$

3. **No associativity with division**

$$a \oplus (b/c) \; \neq \; (a \oplus b)/c$$

4. **XOR does not preserve quotients**

$$\lfloor (a \oplus b)/c \rfloor \; \neq \; \lfloor a/c \rfloor \oplus \lfloor b/c \rfloor$$

✅ TRUE WITH CONDITIONS (important)

5. **Division by power of 2 = right shift**

$$a/2^k = a \gg k$$

6. **XOR preserved after division by power of 2**

$$\text{If } a \equiv 0 \pmod{2^k},\ b \equiv 0 \pmod{2^k}$$
$$(a \oplus b)/2^k = (a/2^k) \oplus (b/2^k)$$

7. **Modulo power of 2 compatibility**

$$(a \oplus b) \bmod 2^k = (a \bmod 2^k) \oplus (b \bmod 2^k)$$

✅ RELATED BUT INDIRECT

8. **Parity relation**

$$(a \oplus b) \bmod 2 = (a \bmod 2) \oplus (b \bmod 2)$$

9. **Division destroys lower XOR bits**

$$(a/2^k) \oplus (b/2^k) \text{ ignores lowest } k \text{ bits}$$

✕ NEVER TRUE (memorize)

- XOR distributive over division
- Division distributive over XOR
- XOR acting as inverse of division
- Any simplification of `(a ⊕ b) / c` for arbitrary `c`

🔑 One-line CP rule

> **XOR only works cleanly with division when the divisor is a power of 2 and bits are aligned.**

That's the **complete, correct list**.

when a xor b is divisle by a,b or by both

Below is the **exact, correct list of cases** for **when**

$$(a \oplus b) \text{ is divisible by } a,\ b,\ \text{or both.}$$

No guesses, no patterns that fail — **only guaranteed truths**.

1️⃣ Divisible by a

$$a \mid (a \oplus b)$$

✅ Happens iff

- $a = 1$
  (since every integer is divisible by 1)
- $b = 0$

$$a \oplus 0 = a \Rightarrow a \mid a$$

× Otherwise

No general divisibility guarantee.

2️⃣ Divisible by b

$$b \mid (a \oplus b)$$

✅ Happens iff

- $b = 1$

- $a = 0$

Same reasoning as above.

3️⃣ Divisible by both a and b

$$\operatorname{lcm}(a, b) \mid (a \oplus b)$$

✅ Only guaranteed cases

- $a = b$

$$a \oplus a = 0 \Rightarrow \text{divisible by any nonzero } a$$

- $a = b = 1$
  (trivial subset of above)

4️⃣ Important negative results (CP traps)

× These are **NOT true**:

- If `a | b` then `a | (a ⊕ b)`
- If `a ⊕ b` is even, then divisible by `a` or `b`
- If `(a & b) = 0`, then divisibility holds

All false in general.

5️⃣ Special case (power of two)

Let:

```ini
a = 2^k
```

Then:

$$a \mid (a \oplus b)$$

✅ Happens iff

- the lowest `k` bits of `b` are zero
  (i.e., `b % a == 0`)

Because:

```css
a ⊕ b ≡ b (mod a)
```

6️⃣ Summary Table

Condition	a	b	Divisible by
b = 0	any	0	a
a = 0	0	any	b
a = b	x	x	both
a = 1	1	any	a
b = 1	any	1	b
a = 2^k, b % a = 0	2^k	multiple	a

🔑 Final CP Rule

> **Except trivial or power-of-two cases,**
> $a \oplus b$ has **no reliable divisibility relation** with a or b.

If you want:

- proof for any case
- CF-style trick problems
- counterexamples for wrong assumptions

say the word.

give me result i am not able to understand what you are saying

Understood. Here is the **FINAL RESULT**, written **very plainly**, **no theory**, **no explanations**.

RESULT: When is `a XOR b` divisible?

Let `c = a ⊕ b`.

1️⃣ Divisible by a

`a | c` **ONLY in these cases:**

```java
b = 0
OR
a = 1
OR
a = b
OR
(a is power of 2 AND b % a == 0)
```

2️⃣ Divisible by b

`b | c` **ONLY in these cases:**

```java
a = 0
OR
b = 1
OR
a = b
OR
(b is power of 2 AND a % b == 0)
```

3️⃣ Divisible by both a and b

`a | c AND b | c` **ONLY in these cases:**

makefile Copy code

```makefile
a = b
OR
a = 0
OR
b = 0
```

4️⃣ Otherwise

css Copy code

```css
a ⊕ b is NOT divisible by a
a ⊕ b is NOT divisible by b
```

One-line rule (memorize)

> **XOR almost never preserves divisibility.**
> **Only trivial or power-of-two cases work.**

That's it.

THOUGHT: Here x>0
and y>0
. So x⊕y
is neither equal to x
nor y
. So x⊕y
is a divisor of x
or y
and x⊕y<x
or x⊕y<y
.

OBSERVATION: Any divisor d
of p

such that $d<p$
 we know that $d \le \lfloor p2 \rfloor$
.

Also, the highest bits of d
 and p
 are different when $d \le \lfloor p2 \rfloor$
.

THOUGHT: Wait but $x \oplus y$
 has the same highest bit as y
 if $y \ge 2 \cdot x$
.

CONCLUSION: So if $y \ge 2 \cdot x$
, then $x \oplus y$
 can not be a divisor of y
.

THOUGHT: But can it be a divisor of x
?

OBSERVATION: If $y \ge 2 \cdot x$
, then $x \oplus y > x$
 because the highest bit in $x \oplus y$
 is greater than that in x
. So $x \oplus y$
 can not be a divisor of x
.

CONCLUSION: If $y \ge 2 \cdot x$
, then $x \oplus y$
 can not be a divisor of x
 or y
. So no solution in this case.

THOUGHT: Now we need to consider the case when $y<2 \cdot x$
. But x
 is small in this problem, making it feasible to iterate over all possible values of y
.

ACTION: Iterate over all possible values of $y<2 \cdot x$

and check if x⊕y
is a divisor of either x
or y
.

Time Complexity: O(x

Here is the **clean, final result**, rewritten **clearly and correctly**, with **no extra theory**.

FINAL RESULT (Correct Logic)

Let

- $x > 0,\ y > 0$
- $d = x \oplus y$

We want:

$$d \mid x \quad \text{or} \quad d \mid y$$

Key facts (used)

1. If d divides a number p and $d < p$, then

$$d \leq \left\lfloor \frac{p}{2} \right\rfloor$$

2. Any number $\leq \lfloor p/2 \rfloor$ has a **smaller highest set bit** than p.
3. XOR keeps the **highest set bit** of the larger number (unless canceled).

Case 1: $y \geq 2x$

- Highest bit of y is **strictly greater** than highest bit of x
- So:

$$x \oplus y \text{ has the same highest bit as } y$$

Check divisibility:

- **Can $x \oplus y \mid y$?**
  × No, because a divisor $< y$ must be $\leq y/2$, but
  $x \oplus y \geq y/2$

- **Can $x \oplus y \mid x$?**

  × No, because $x \oplus y > x$

✅ Conclusion

$$\boxed{\text{If } y \geq 2x, \text{ no solution}}$$

Case 2: $y < 2x$

- x is small
- Number of possible y values is limited

✅ Action

Iterate:

```yaml
for y = 1 to 2x - 1:
    d = x ^ y
    if x % d == 0 or y % d == 0:
        valid
```

Final Answer (one line)

$$\boxed{\text{Only possible solutions occur when } y < 2x. \text{ For } y \geq 2x, \text{ no solution exists.}}$$

Time Complexity

$$\boxed{O(x)}$$

Your reasoning is **correct** and **logically sound**.

give code

Here is the **direct code** implementing exactly the logic you derived.

No extra checks, no tricks.

Python

python

Copy code

```python
def find_valid_y(x):
    res = []
    for y in range(1, 2 * x):
        d = x ^ y
        if d != 0 and (x % d == 0 or y % d == 0):
            res.append(y)
    return res


x = int(input())
ans = find_valid_y(x)

print(len(ans))
print(*ans)
```

Go

go

Copy code

```go
package main

import "fmt"

func main() {
    var x int
    fmt.Scan(&x)

    ans := []int{}

    for y := 1; y < 2*x; y++ {
        d := x ^ y
        if d != 0 && (x%d == 0 || y%d == 0) {
            ans = append(ans, y)
```

```
        }
    }

    fmt.Println(len(ans))
    for _, v := range ans {
        fmt.Print(v, " ")
    }
}
```

C++

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int x;
    cin >> x;

    vector<int> ans;
    for (int y = 1; y < 2 * x; y++) {
        int d = x ^ y;
        if (d != 0 && (x % d == 0 || y % d == 0)) {
            ans.push_back(y);
        }
    }

    cout << ans.size() << "\n";
    for (int y : ans) cout << y << " ";
}
```

Complexity

```css
Time:  O(x)
Space: O(1) (excluding output)
```

This matches **exactly** the reasoning you wrote.