O(n)?

* DAY-3 → 29/3/2023

* Print (10 * 4 / 6 + 3 - 1.1.2) in all languages

1. Python

Print (10 * 4 / 6 + 3 - 1.1.2)

=> 8.666 —

2. C we get 10 * 4 / 6 + 3 - 1.1.2

=> 8.

3. Java

we get 10 x 4 / 6 + 3 - 1 * 1.2

=>

* 10 * 4 / 6 + 3 - 1 % 2

① 10 * 4 ⇒ ② 40 / 6 ⇒ 6.6

③ 1 % 2 ⇒ 1

④ 6.6 + 3 = 9.6

⑤ 9.6 - 1 = 8.6

int ⇒ 32 bits memory

* Bitwise operators

	C	Python	Java
and	&	&	&
OR	|	|	|
XOR	^	^	^
negation	~	~	
1's comp	!		

* 7 + 2 & 4 + 3 & 9 Convert binary and check precendence.

0 1

(8 4 2 1)

7 = 0111

2 = 0010

4 = 0100

3 = 0011

9 = 1001

① 2 & 4

0010
0100
0000 ⇒ 0

② 3 & 9

0011
1001
0001 ⇒ 1

7 + 0 + 1

0111
0000
0001
0001

0000 → 0
0001 → 1
0001
0111 → 7

0000 → 2 & 4
0111 → 4
0000

And operation

0 0 — 0
0 1 — 0
1 0 — 0
1 1 — 1

OR operation

| => or operation.

0 0 — 0
0 1 — 0
1 0 — 0
1 1 — 1

2 && 4 => 0000
3 && 9 => 0001

7 + 2 && 4

0111
(+) 0000
0111

0111 + 3 && 9 => 0001

0111
(+) 0001
0001

final answer = 1
= 0001

* && → high precedence
* || → low precedence
* find ① 10 && 3 && 4 && 5 => 0
② 10 | 3 && 4. => 10

Precedence Bitwise

1. ~ — negation
2. & — and
3. ^ — Power / exponent
4. | — or

1. Check valid or not
2. If valid what is priority } Conditions
3. System o/p.

1. 10 & 4 ~2

Not valid

2. 6 | 3 & 9 + 6

⇒ 7

3. 2 ~ 4 ^ 3 * 2

In valid.

4. ~9 + 4 < 6.

2 valid

Cause negation takes only one value

~value ⇒ valid

~~ve~~ V~V ⇒ Invalid

V~ ⇒ Invalid

* XOR operation

Types

1. Inclusive ⇒ OR
2. Exclusive ⇒ XOR

0	0	0
0	1	1
1	0	1
1	1	0

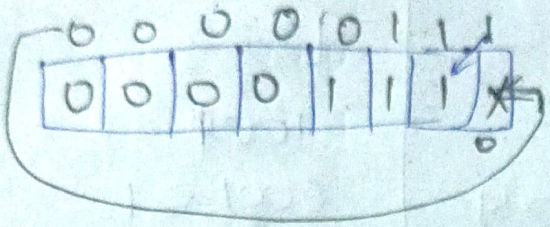
Case 1

0	1
0	1
0	0

* Shifting operators

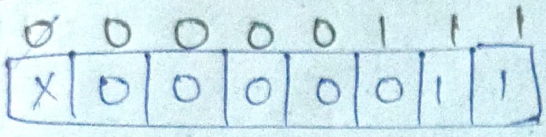
1. left shift -> << -> Right to left
2. Right shift -> >> -> left to right

7 << 1



=> 01110
=> 14

2. 7 >> 1



=> 0011
=> 3

* Bit manuplation tricks

=> xor of numbes itself is 0
=> xor of numbes 0 is numbes itself

1. 5^5 => 0
2. 5^0 => 5
3. 0^5 => 5

* (1^2^3) => 0

~~0000~~
0001
0010
0011 => 3^3 => 0

4^6^5

~~00~~ 0100
0110
0010 -> 2
0101
0111 => 7

* 5>>1

0101 0010
~~0001~~ => 2

5>>2

~~00~~ 0101
~~0001~~

0101 0101
0001 => 1

* 5>>1

0101 0010
~~0001~~ => 2

14 >> 4

8 4 2 1

1 1 1 0

0000 ⇒ 0.

6 >> 3

0 1 1 0

0000 ⇒ 0.

9 >> 2

1001

0010 ⇒ 2

9 >> 1

1001

0100 ⇒ 4

9 >> 3

1001

0001 ⇒ 1

9 >> 4

1001

0000 ⇒ 4

* left shift

* 5 << 2

0101

01010

5 * 2^2 ⇒ 5 x 4

⇒ 20

10 << 3

10 * 2^3 ⇒ 10 x 8

⇒ 80

9 << 4

9 * 2^4

9 * 16 ⇒ 144

2^{31}

* to creating an array find out the smallest missing positive integer After

```
def finding_missing_number(nums):
    n = len(nums)
    r = n
    for i in range(n):
```

```
        r ^= i ^ nums[i]
    return r
n = int(input())
nums = list(map(int, input().split(" ")))
print(finding_missing_duplicate_numbers(nums))
```

o/p

* In the given array every number occurs twice only one number occurs once

```
arr = [1,5,1,2,3,2,3]
def findSingle(ar,n):
    res = ar[0]
# doing XOR
    for i in range(1,n):
        res = res ^ ar[i]
    return res
ar = [2,3,5,4,5,3,4,2,88]
print(findSingle(ar,len(ar)))
```

* Swapping using XOR

```
a = 100
b = 200
print("a:",a,"b:",b)
```

o/p a: 100 b: 200
b: 100 a: 200

a = a^b => a = 100^200

b = a^b => b = 100^100^200 ans => b = 100

a = a^b => a = 100^200^100 ans a = 200

```
print("a:",a,"b:",b)
```

* For the given number n check k^{th} bit set or not

n = 10
k = 2

Condⁿ = n & (1 << k-1)

10 & (1 << 2-1)

10 & (0)

⇒ 0

For given number n. find out XOR of all n numbers.

o/p : 12

n = 12
xor = 0
for i in range(1, n+1):
    xor = xor ^ i
Print(xor)

n = 7
xor = 0
If n%4 == 0:
    Print(n)
elif n%4 == 1:
    Print(1)
elif n%4 == 2:
    Print(n+1)
elif n%4 == 3:
    Print(0)

Space Complexity is
O(1)
⇓
Constant

* Given a range l-r/ print xor

l=2

r=5

2^3^4^5

* lets say

4 to 9

xor (1 to 9) ^ xor (1 to 3)

(1^2^3^4^5^6^7^8^9) ^ (1^2^3)

in above 123 xors will get Cancelled

generalise xor(r) ^ xor(l-1)

* Check given number is odd or even.

normal way n%.2==0 even else odd.

n=13

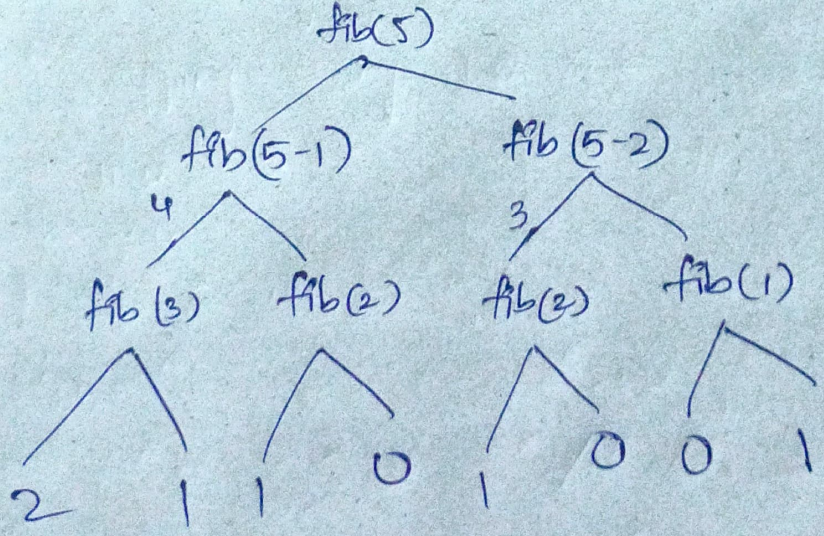
if (n&1==0):

    Print ("Even").

else:

    Print ("odd")

* Fibonacci using Recursion



* fibonacci ith term using iterative

```
n = int(input("enter the term:"))
n1 = 0
n2 = 1
if(n<0):
    Print("not possible")
else:
    for i in range(0,n-1):
        n3 = n1+n2
        n1 = n2
        n2 = n3
    Print(n2)
```

* fibonacci using recursion

```
def fib(n):
    if n<0:
```