TODO:

Modulo distribution over division

↳ Extended Euclid's algorithm

09/30/2023

Problem :-

i/p		o/p
a	b	a^b
5	2	$5^2 = 25$
2	10	$2^{10} = 1024$
3	3	$3^3 = 27$

```
func powerOf ( a: Int , b: Int ) -> Int {

        return pow (a, b)

    or

}
```

or

```
                                                         long
func powerof ( a: Int , b: Int) -> ~~Int~~ {
long     let ~~ans~~ = 1
         for _ in 0 ... b {
             ans *= ans
         }
```

return ans

}

→ for big output like 4^{200} →

do $\underline{\% M}$ to get it in range

where $M = 10^9 + 7$

will get the range $[0, \underline{10^9 + 7}]$

↳ excluded value

- it is prime number
- largest prime in the range of Int

- 10^9+7 → $\boxed{1e9+7}$ → can be written like this in code

→ $1 \times 10^9 + 7$

Exercise: • 24007 in e notation

→ $24e^3 + 7$

- 9002 → 9e3+2
- 0.0004 → 4e-4
- -1.002 → -2e-3 -1
- -49.00075 → -75e-5 - 49

$$(a+b) \% M = (a \% M + b \% M) \% M$$

$$\rightarrow (4+5) \% 6 = ((4 \% 6) + (5 \% 6)) \% 6$$

$$= (4+5) \% 6$$

$$(9 \% 6) = 9 \% 6 = 3$$

$$= 3$$

$$(a-b) \% M = (a \% M - b \% m + M) \% M$$

$$(7-5) \% 6 = (7 \% 6 - 5 \% 6 + 6) \% 6$$

$$2 \% 6 = \underline{\underline{2}} \quad = (1 - 5 + 6) \% 6$$

$$= 2 \% 6 = \underline{\underline{2}}$$

$$(a * b) \% M = (a \% M * b \% M) \% M$$

$$(a / b) \% M = \cancel{\left(\frac{a \% M}{b \% M}\right) \% M}$$

$$(5/6) \% 2 = \left(\frac{5 \% 2}{6 \% 2}\right) \% 2$$

$$= \left(\frac{2}{0}\right) \% 2 \quad \text{(will not work)}$$

- modulo distribution over division
  - ↳ Extended Euclid's algorithm

→ Use % M where the operation is happening.

like ans = (ans * a) % M

wherever there is possibility of overflowing.

→ by default e is double, type cast it to int (1e9+7)

- int a = 1e6 , int b = 1e7

  int c = a*b × (does not work)

  long long c = a*b (still not work)
  ↳ as intermediate result is still int)

- long long a = 1e6 , b = 1e7

  long c = a * b : will work

  but no need for both a, b to be long long

- int a = 1e6 , int b = 1e7;

  long long c = $\underbrace{\text{(long long)}}_{\text{type casting.}}$ a * b

---

Problem:: i/p o/p

i/p	o/p
N	divisors count
12	1, 2, 3, 4, 6, 12 (6)
5	1, 5 (2)
24	1, 2, 3, 4, 6, 8, 12, 24 (8)

```
func divisorCount ( N : int) -> Int {
    var count = 0
    for i in 1 ... N {
        if N % i == 0 {
            count += 1
```

}

}

return count

}

→ constraints $1 \leq N \leq 10^9$

N iterations $= 10^9$ iterations

1 GHz $= 10^9$ instructions / sec

10^9 iterations → 1 sec ? ✗

↳ 1 iteration $\approx$ 5 instructions

now
10^9 iterations → 5 secs

→ Now if $1 \leq N \leq 10^8$

10^9 iterations → 1 sec

2×10^9 iterations → 2 sec

$10^9 \times 10^9$ iterations → 10^9 secs

$$\frac{10^9}{60 \times 60 \times 24 \times 365} \text{ yrs} \rightarrow$$

$= \underline{31.71}$ yrs

for instructions $= 31.71 \times 5$ yrs

$\approx\ > 150$ yrs

- optimisation

→ Back from question

24

i	N/i
1	24
2	12
3	8
4	6
6	4
8	3
12	2
24	1

$i < \dfrac{N}{i}$

100

1	100
2	50
4	25
5	20
10	10
20	5
25	4
50	2
100	1

$i \le \frac{N}{i}$

$= i^2 \le N$

$= i \le \sqrt{N}$

- so previously $N \rightarrow 10^{18}$ iterations $\rightarrow 10^9$ secs

$\sqrt{N} \rightarrow \sqrt{10^{18}} \rightarrow 10^9$ iterations

$= 1 \times 5$ secs

```
for countDivisors ( N : Int) -> Int {
    let count = 0
        for i in 1 ... N/i {
            if N % i == 0 {
                if N%i == i {
                    count += 1
                }
            else { count += 2 }
```

```
    }
    return count.

}
```

```
func solve (int N) {

    int a=20, b=40;

    long long c=100;

    float z = 2.5

  for i in 0...5 {
      a= b+c
  }
}
```

5 iterations

Not dependent on N

Constant time complexity $O(1)$

```
func solve (int N) {
    for j in 0....N {
        int a=20, b=40;

        long long c=100;

        float z = 2.5

        for i in 0...5 {
            a= b+c
        }
    }
}
```

$N = 10^3, 10^5$

5

$N \times 5 \approx N$

Linear time complexity $O(N)$

```
func solve (int N) {
    for i in 0...N {
        for j in 0....N {
            — — ,
        }
    }
    for i in 0...N {
        — —
    }
}
```

N^2

N

$N^2 + N \approx O(N^2)$

↳ % of N in N^2

- $N = 10^3$

$$\frac{N}{N^2} \times 100 = \frac{10^3}{10^6} \times 100 = 0.1\%$$

- $N = 10^{\cancel{2}4} \Rightarrow 10^{-2} \Rightarrow 0.01\%$

with increasing value of N^2, contribution of N is less.

```
func solve (N: Int) {
                          1 → then   O(log N)
    for i ... 0 ... N {
          i = i*2              → infinite loop
    }
```

	i	total	
2^0	1	1	1
		+	
2^1	2	1	2
		+	
2^2	4	1	3
		+	
2^3	8	1	4
	⋮		
2^k		$k+1$	

but what in terms in N?

when $2^k = N$

$\log_2 2^k = \log_2 N$

$\log_a a = a \longrightarrow \quad k = \log_2 N$

now $k+1 = \log_2 N + 1$

$= \log_2 N$

Sol 1 : $4n^2 + 3n + 1000 \rightarrow O(n^2)$ which is better?

Sol 2 : $n^3 + n \rightarrow O(n^3)$

	$n=1$	$n=10$	$n=10^3$	$n=10^4$
Sol 1	1007	10430	$4\times10^6 + ..$ ✓	$4\times10^8 + ...$ ✓
Sol 2	2 ✓	1010 ✓	10^9	10^{12}

2 is better after a certain threshold

Sol 1 : $10n^2 + n \rightarrow O(n^2)$

Which is better

Sol 2 : $n^2 + 10n \rightarrow O(n^2)$

2 is better

1 $\quad 10^2 \quad 10^4$

Sol 1	11	10^5	10^9 + ...
Sol 2	11	10^4 ✓	10^8 + ... ✓

Linear Search

```
func linearSearch ( arr: [Int], N: Int , key: Int) {
                                        -> Bool
    for i in 0 ... arr.count-1 {
        if arr[i] == k {
            return true
        }
        return false
}
```

→ O(N)

- Best Case = O(1)
- Average case = O(N/2)
- Worst case = O(N)

upper bound

★ Big-O defination:-

Puts an upper bound on complexity of an algorithm based on the input size after a certain threshold.

A)

$[a,b] = b-a+1$

$[a,b) = b-a$

$(a,b] = b-a$

$(a,b) = b-a-1$

---

it		i	total
1	1	$N \quad N/2^0$	
2	1	$N/2 \quad N/2^1$	
3	1	$N/4 \quad N/2^2$	
	1	$N/8 \quad N/2^3$	
	1		
	1		
		1	$\frac{N}{2^k}$
	1		
$k+1$		$0 \quad k$	

$O(\log_2 N)$

$$\frac{N}{2^k} = 1$$

$$N = 2^k$$

$$\log_2 N = k$$

i j

N [0 - N) N

$N/2$ [0 - $N/2$] $N/2$ (log N)

$N/4$

⋮

i j

$N/2$ [2 N] N - 2 + 1

$N/2 + 1$ [4 N) N - 4 + 1 $O(N^2)$

$N/2 + 2$ [8 N] N - 8 + 1

→ (N x P)

N x i j

1 1 - 1^k ⇒ 1

2 1 - 2^k ⇒ 2^k

3 ⇒ 3^k

$1 + 2^k + 3^k + \cdots + N^k$

$\Rightarrow$

$\frac{N^{k+1}}{N^k}$

[2, 4, 8, 16, 20]

---

i j

$2N$

N

0 → 1 N

1 → 1 N

$O(N)$

$1\ GHZ = 10^9$ inst / sec $\quad \underline{N = 10^6}$

Big O	N^3	N^2	$\log \underset{2}{N}$	N^2	$N \log N$	N	$\sqrt{N}$	$\log N$	1
# iterations	10^{18}			10^{12}					
Time	31.7 yrs			1000 sec					

$N = 60$	$N = 30$
2^N	2^N
10^9	10^{18}
1 sec	31.7 yrs

Errors

Wrong Answer →
- logic
- Edge Cases
- Overflow (datatype)

* TLE →
  - Infinite loop
  - Happens more in Brute force approach.

* RTE → out of bound, division by 0, null pointer, undhandled except

* CE → syntax issue.

Complexity : Test cases $\times$ ________

$\Rightarrow$ $t \times$ ________

TLE $\underline{\Leftarrow}$ 10^8 iterations ✓

$T = 10^3$ $N = 10^2$ $\rightarrow$ $T \times N^2$

$\Rightarrow 10^3 \times 10^4 \rightarrow 10^7 \leq 10^8$

$\rightarrow T \times N^3$

$\Rightarrow 10^3 \times 10^6 \rightarrow 10^9 \underline{\leq} 10^8$ ✗

TLE

$T = 100$ $N = 20 \rightarrow$ $100 \times 2^{\cancel{20}} 2^{10} \times 10^6$

$\Rightarrow \underline{10^8}$ ✓ No change the logic

$T = 100, \; N = 30 \rightarrow T \times 2^N$

$\rightarrow 10^2 \times 2^{30} \Rightarrow 10^{11} \; \times$

$\log_2 10^3$

$T = 100, \; N = 10^{18} \rightarrow 10^{10} \; \times$

$\Rightarrow \log_2 2^{10}$

$T = 10^6, \; N = 10^3 \rightarrow 10^9 \; \times$

$\Rightarrow 10$

$(T+N) \rightarrow$ precomputation

Importance of constraints :

1. Data type to use
2. Decide the logic
3.