Sum of Digits :-

input : 38 $\quad 1+1=2$

output = 2

Trick :- any power of 10 can be written as;

$10^n = 9\times 111 + 1 \quad 10^7 = 9\times 1111111 + 1$

number : 138 $= 1\times 10^2 + 3\times 10^1 + 8\times 10^0$

number $(n)\ c_3c_2c_1 = c_3\times(9\times k_3+1) + c_2(9\times k_2+1) + c_1(9\times k_1+1)$

$= 9\times(k_3c_3 + k_2c_2 + k_1c_1) + \underbrace{(c_1+c_2+c_3)}_{n_2 \equiv c_4c_5c_6}$

$= 9(\ ) + \underbrace{(c_4+c_5+c_6)}_{<9}$

* $n \% 9 = c_4+c_5+c_6 < 9$

$==$ answer.

if $n\%9 == 0$ ans $= 9$ because sum can't be 0.

ip 27 → op 9.

Bulb on off :-

there are n bulbs, at each round all multiples of ith index will be toggled.

return : how many bulbs are turned on after the end of round n.

	⊗	⊗	⊗	⊗
round 1	O	O	O	O
round 2	O	⊗	O	⊗
round 3	O	⊗	⊗	⊗
round 4	O	⊗	⊗	O

* at round n only perfect squares are on.

because only perfect squares are toggled even number of times.

16 → 2, 4, 8, 16

15 → 3, 5, 15

12 → 2, 3, 4, 6, 12

6 → 2, 3, 6

36 → 2, 3, 4, 6, 9, 12, 18, 36

* there are $\sqrt{n}$ number of perfect squares between $(0, n)$

#⭐, #algorithm/math, #Trick

#Math_algo_numberTheory

#LeetCode/Hard

Number of ways to reorder array to get the same BST :-

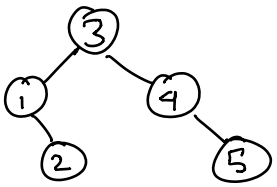
input [3, 1, 4, 2, 5]

o/p : 3

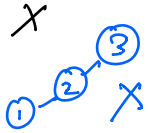
---

* Root has to be same in all the possible BST's here its 3.

* Relative order of appearance is important. in all subsequent rearrangements 2 must come after 1.
  ✓ doesn't matter if 4 comes before 1 or after ; it's gonna go to the right subtree.



[3 1 4 2 5]
[3 1 2 4 5] ✓
[3 4 1 2 5] ✓
[3 2 1 4 5] ✗



#Combinatorics

Left = (1, 2)
Right = (4, 5)

3 _ _ _ _

to place 1, 2 there are $^{n-1}C_2$ ways.
to place 4, 5 " " $^{n-1}C_2$ ways.

if right = (4, 5, 8, 9) then this become another subproblem.

⟹

```
int solver(arr){
    n = arr.size();
    if (n < 3) return 1;
    vector<int> leftsubtree, rightsubtree;
                * arr[i] < arr[0]   * arr[i] >= arr[0]
    int x = solver(leftsubtree)
    int y = solver(rightsubtree)
    int z = pascalTriangle[n-1][leftsubtree.size()]
    return (x * y * z)
```

$^{n-1}C_{leftsize}$ x solver(left) x solver(right)
on right

Excel Sheet :-

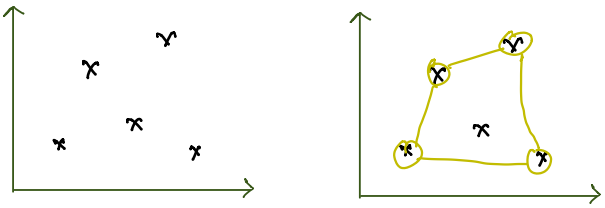
	A	B	C	...	Y	Z	AA	AB	...	AZ	BA ...
	1	2	3		25	26	27	28			
Q	0	0	0		0	1	1	1			
R	1	2	3		25	0	1	2			

```
input -> integer n
while (n > 0){
    n--;
    rem = n % 26;
    char res = 'A' + rem;
    ans.pushback(res);
    n = n/26;
}
```

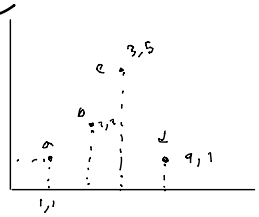
Expect the Fence :-

#LeetCode/Hard

return the minimum distance fence.



Dry run



arr [a b c d]

i = 0

L = a(1,1) U = a(1,1)

i = 1

L = a(1,1), b(2,2) U = a(1,1) b(2,2)

lower got popped

i = 2

L = a(1,1) c(3,5)

U = a(1,1) b(2,2) c(3,5)

i = 3

L = a c d U = a b c d

a & d is upper — b c d is upper

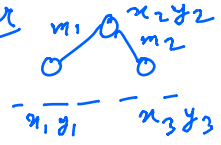
= a d.

* main concept

→ a tree on fence will be either upper or lower but not both.

→ a tree inside will be both upper & lower relative to other trees.

upper



$m_2 - m_1 < 0$

$\frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1} < 0$

$(y_3 - y_2)(x_2 - x_1) - (y_2 - y_1)(x_3 - x_2) < 0$

( ) > 0

lower