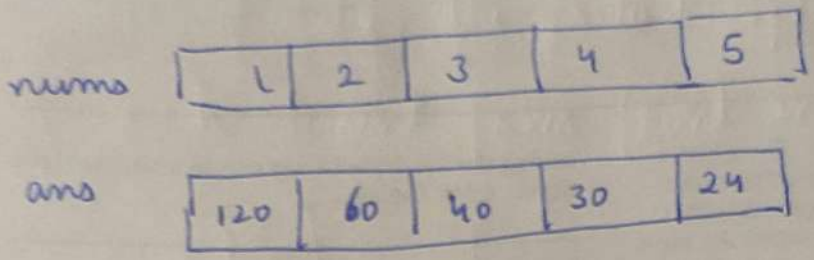
Product except self.



Understanding the problem.

Conditions given in the question.

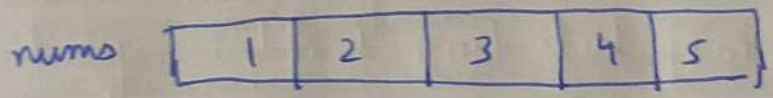
1.) $a * b = c$

where a, b are numbers from nums array

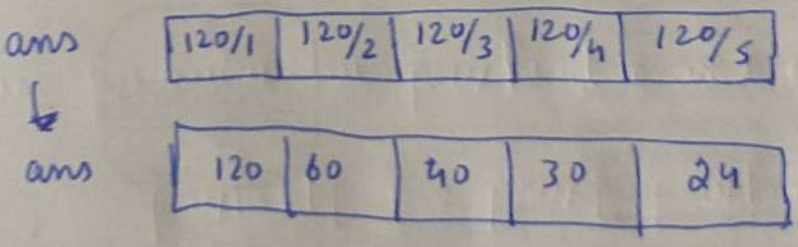
c: 32 bit int even after multiplying 2 32 bit ints.

2.) No division algo to be used.

Although its suggested, division algo is not to be used but let's explore it once to know what it is.



Product = $1 * 2 * 3 * 4 * 5 = 120$



But why we divided ??

Because we want to remove that element from product.

Why this is not suitable method?

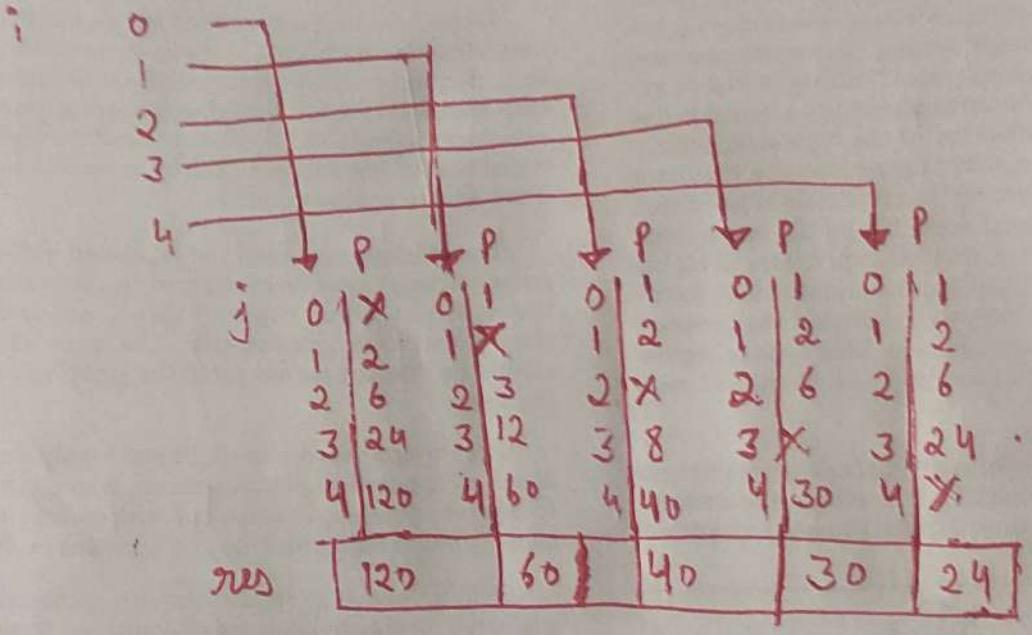
→ Division by zero is possible

→ large values may cause underflow or overflow.

Product except self.

②

nums	1	2	3	4	5
	0	1	2	3	4



Brute force.

~~for~~ product = 1;

for i to n
  product = 1;
  for j to n
    if i == j
      ~~continue;~~ continue;
    else
      product *= arr[j];
  res[i] = product;

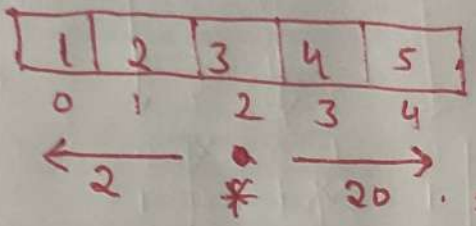
T: $O(n^2)$

S: $O(1)$

Repeatitive operations observed.

⇒ we are calculating the product repeatedly for each element from front and behind

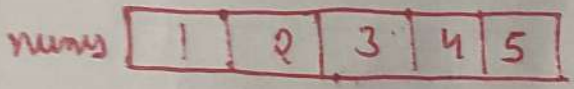
For example



↓

product except self for '3'

So if we calculate previous and next for each element we can find product except self for all.



'prevProd' iterations must start from left as the first element would have no prev product and can be set to '1'

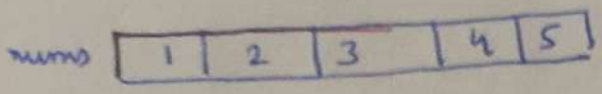
Similarly 'nextProd' must start from right as the last element would n't have any right product. and can be set to '1'

Thus formulae for prevProd and nextProd are

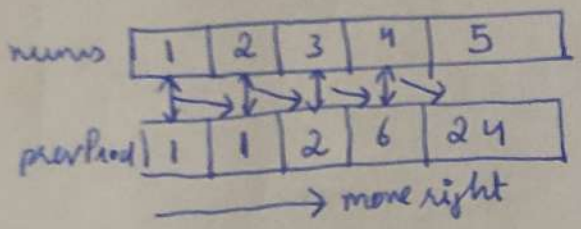
prevProd[i] = ~~prev~~ prevProd[i-1] * nums[i-1]

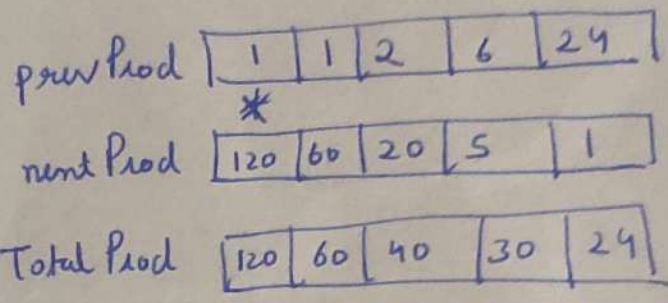
nextProd[i] = nextProd[i+1] * nums[i+1]

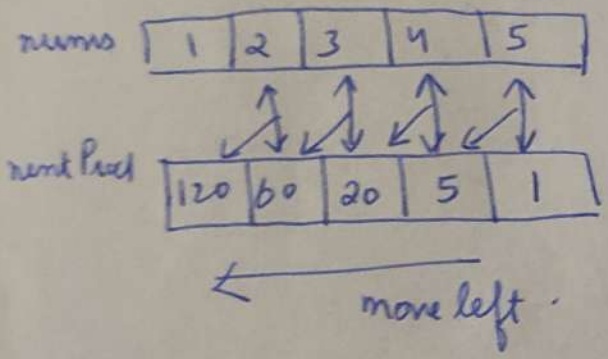
Semi optimised code



prevProd[0] = 1 , nextProd[n-1] = 1







Pseudo code

```
n = nums.size();
res[n]
prevProd[n,1]
nextProd[n,1]
prevProd[0] = 1 , nextProd[n-1];
for i to n        (i++)
    prevProd[i] = prevProd[i-1] * nums[i-1];

for i = n-2 to i >= 0  (i--)
    ~~next[i]~~
    nextProd[i] = nextProd[i+1] * nums[i+1];

for i to n
    res[i] = prev[i] * nextProd[i];
             prod
return res;
```