Product of array except itself

nums = [1, 2, 3, 4]

ans = [24, 12, 8, 6] (2x3x4, 1x3x4, 1x2x4, 1x2x3)

CAN'T USE DIVISION OPERATOR

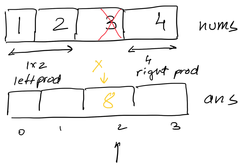
① Brute force

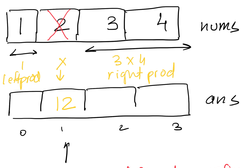
```
vector<int> ans(n, 1);
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        if (i != j) {
            ans[i] *= nums[j];
        }
    }
}
return ans
```

Time complexity : O(n²)

② Optimal approach

need it in O(n) TC & O(1) SC

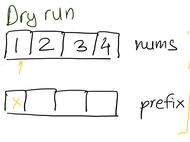




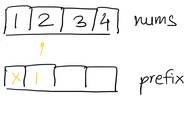
for simplicity : leftprod → prefix
rightprod → suffix

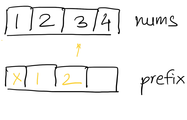
① calculate prefix for all indexes

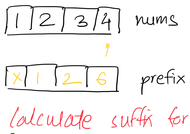
Dry run



This step won't happen because we are starting from index 1 not 0



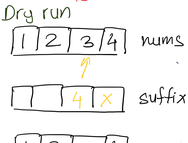


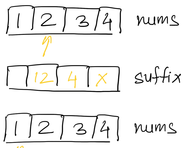


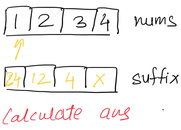

```
vector<int> prefix
for (i=1; i<n; i++) {
    prefix[i] = prefix[i-1] * nums[i-1]
}
```

② calculate suffix for all indexes

Dry run

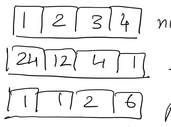


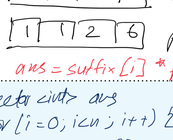





```
vector<int> suffix
for (i= n-2; i>=0; i--) {
    suffix[i] = suffix[i+1] * nums[i+1];
}
```

③ calculate ans





ans = suffix[i] * prefix[i]

```
Vector<int> ans
for (i=0; i<n; i++) {
    ans = suffix[i] * prefix[i];
}
return ans;
```

BUT SC here is O(2) excluding ans vector but we need to make it O(1)

so instead of making a storage for prefix we will directly multiply in answer thereby reducing space comp

```
vector<int> ans(n, 1);
for (int i=1; i<n; i++) {
    ans[i] = ans[i-1] * nums[i-1];
}
int suffix = 1;
for (int i=n-2; i>=0; i--) {
    suffix *= nums[i+1];
    ans[i] *= suffix;
}
return ans;
```