Array Manipulation Patterns

1) Product of Array except self.

given an array nums, return an array answer where answer[i] = product of all the elements of nums except num[i]

required: O(N) Solution
Note: do not use division operation.

Approach#1 :- (Brute Force Approach):

```
int n= nums.length;
int[] answer = new int[n];

for(int i=0 ;i<n; i++){
        int ans =1
        for(int j=0; j<i ; j++){
              ans= ans*nums[j];
        }
        for (int k=i+1; k<n; k++){
              ans=ans*nums[k];
        }
        answer[i]=ans;
}
return answer;
```

T.C: $O(N^2)$
S.C: $O(1)$

Approach #2:- (using division operator)

find # of zeroes if # of zeroes > 1 then all elements of answer array is 0

else

find product of non zero elements.

```
int n= nums.length;
int[] answer = new int[n];
int zeroes=0;
int prod= 1;
for(int i=0 ;i<n ;i++){
        if (nums[i] = 0){
            zeroes++;
        }else{ prod*= nums[i];
        }
}
if (zeroes > 1){
        return answer;
}
~~for(int i=0;i<~~
else if (zeroes == 1){
        for(int i=0; i<n; i++){
                if (nums[i]= = 0){
                    answer[i] = prod;
                }
        }
}
else {  for(int i=0; i<n; i++){
                answer[i]= prod/ nums[i];
        }
}
return answer;
```

T.C: O(N)

S.C: O(1)

Approach #3: (using prefix and postfix arrays)

Find postfix and prefix product arrays.

Say if nums = [1, 2, 3, 4]

then

prefix = [1, 1, 2, 6]

postfix = [24, 12, 4 1]

then final output = [24, 12, 8, 6]

```
int n= nums.length;
int[] answer = new int[n];
int[] prefix= new int[n];
int[] postFix = new int[n];
Prefix[0] = 1;
for(int i=1; i<n; i++){
        prefix[i]= prefix[i-1]*num[i-1];
}
postfix[n-1]=1;
 for(int i=n-1; i>0; i--){
        postfix[i-1]= postfix[i]* nums[i];
 }

 for(int i=0; i<n; i++){
        answer[i] = prefix[i]* postfix[i];
 }
 return answer;
```

T.C: O(N)

SC: O(N)

Approach#4:- (Prefix array with O(1) space)

```
int n= nums.length;
int[] answer= new int[n];

 answer[0]= 1;

 for (int i=1; i<n; i++){
        answer[i]= answer[i-1]*nums[i-1];
   }
   var lastZeroAtIndex= [illegible]
int postProd = 1

  for (int i=n-1; i>=0 ;i--){
        answer[i]= answer[i]* postProd;
        postProd *= nums[i];

  }

  return answer;
```

T.C: O(N)

S.C: O(1)

2. Move zeroes

given an array nums, you have to move all zeroes to end restoring the relative order of non-zero elements.

example:-

nums=[0,1,0,3,12]

O/p:- [1,3,12,0,0].

Approach ~~#~~ 1:- (Brute Force)

count zeroes
store non zero elements in new array then store them back to original Array.

```
int zeroes=0;
int n= nums.length;
int[] ans = new int[];
for (int i=0 ; i<n ; i++){
        if (nums[i]==0){          } not needed.
            zeroes++;
        }
}
int k=0;
for (int i=0 ; i<n ; i++){
        if(nums[i] != 0){
            ans[k++]= nums[i];
        }
}
while (k<n){
        ans[k++]=0;
}
for (int i=0 ; i<n ; i++){
        nums[i]=ans[i];
}
return ans;
```

T.C: O(N)
S.C: O(N)

Approach#2: (using 2 pointers) {In place modify}

```
int  n= nums.length;
int  lastZeroAtIndex=0;

for(int i=0; i<n ;i++){
        if (nums[i] != 0){
            nums[lastZeroAtIndex++]= nums[i];
        }
}
while(lastZeroAtIndex<n){
        nums[lastZeroAtIndex++] =0;
 }
return num;
```

T.C: O(N)
S.C: O(1)

Summarize

How to avoid extra space in array manipulation problems?

1) Use Two pointers to in place modify array.
2) Instead of making copy, overwrite existing array.
3) encode information into array itself.
4) Reuse input array as output
5) Sometimes, you can encode two values into one array slot, {Bit manipulation, modulo}