Arrays:Prefix and Suffix Sum/Product

A prefix sum/product is the cumulative sum/product of elements in an array up to a given index.
A suffix sum/product is the cumulative sum/product of elements in an array from a given index to the end.

Var array=[1,2,3,4,5]

Prefix Sum:

```
function PrefixSum(nums){
    var sum=0;
    var array=[];
    for(var i=0;i<nums.length;i++){
        sum+=nums[i];
        array.push(sum);
    }
    return array;
}
```

Prefix Product:

```
function PrefixProduct(nums){
    var product=1;
    var array=[];
    for(var i=0;i<nums.length;i++){
        product*=nums[i];
        array.push(sum);
    }
    return array;
}
```

Suffix Sum:

```
function PrefixSum(nums){
    var sum=0;
    var array=[];
    for(var i=nums.length;i>=0;i-){
        sum+=nums[i];
        array.push(sum);
    }
    return array;
}
```

Suffix Product:

```
function SuffixProduct(nums){
    Var product=1;
    var array=[];
    for(var i=nums.length;i>=0;i-){
        product*=nums[i];
        array.push(sum);
    }
    return array;
}
```

Problems Based on Prefix and Suffix Sum/Product

Level:Easy:

1.Running Sum of 1D Array:

Problem Link-https://leetcode.com/problems/running-sum-of-1d-array/description/

Solution 1:

Algorithm:

1.Create a new array to store the running sum at every 0 till ith index.
2.Take a variable named as sum to store the sum till ith index.
3.Each time we compute sum,push it into the array.

Code:

```
function getRunningSum(array,nums){
    var sum=0;
    for(var i=0;i<nums.length;i++){
        sum+=nums[i];
        array.push(sum);
    }
    return array;
}
var runningSum = function(nums) {
    var array=[];
    getRunningSum(array,nums);
    return array;
```

```
};
```

Example:
nums=[1,2,3,4]

Output:
[1,3,6,10]

Time Complexity:
O(n),to iterate over n elements to calculate sum till every ith index.

Space Complexity:O(n),to store sums upto every ith index.

2.Find Pivot Index/Equilibrium Index/Point:
Level:Medium

Problem Link:https://leetcode.com/problems/find-pivot-index/

Solution 1:
Code:

```
function getLeftSum(i,nums){
     var ls=0;
     for(var t=0;t<i;t++){
        ls+=nums[t];
     }
     return ls;
}
function getRightSum(i,nums){
     var rs=0;
     for(var t=i+1;t<nums.length;t++){
        rs+=nums[t];
     }
     return rs;
}
var pivotIndex = function(nums) {
    for(var i=0;i<nums.length;i++){
        var leftSum=0;
        var rightSum=0;
        leftSum=getLeftSum(i,nums);
        rightSum=getRightSum(i,nums);
```

```
        if(leftSum==rightSum){
            return i;
        }
        console.log(i+" ls-> "+leftSum+" rs->"+rightSum);
    }
    return -1;
};
```

Time Complexity:O(n^2)
Space complexity:O(1)

Solution 2:

Code:

```
var pivotIndex = function(nums) {
    var prefixSumArray=[];
    var suffixSumArray=new Array(nums.length);
    var psum=0;
    for(let i=0;i<nums.length;i++){
        psum+=nums[i];
        prefixSumArray.push(psum);
    }
    var ssum=0;
    for(let i=nums.length-1;i>=0;i--){
        ssum+=nums[i];
        suffixSumArray[i]=ssum;
    }
    console.log(prefixSumArray);
    console.log(suffixSumArray);
    for(let i=0;i<nums.length;i++){
        var leftSum=0;
        var rightSum=0;
        if(i-1>=0){
            leftSum=prefixSumArray[i-1];
        }
        if(i+1<nums.length){
            rightSum=suffixSumArray[i+1];
        }
        if(leftSum==rightSum){
            console.log(leftSum);
            console.log(rightSum);
            return i;
```

```
        }
    }
    return -1;
};
```

Time Complexity:O(n)
Space Complexity:O(n)

3.Product Of Array Except Self:

Level:Medium

Problem Link:

https://leetcode.com/problems/product-of-array-except-self/description/

Solution 1:
Algorithm:
1.Iterate a first for loop on i->0 to array length.
2.For every i,initialise a product variable with value 1
3.Iterate in a second for loop using jth variable from j->0 to array length
4.Each time inside j,
Check if i!=j(as we have to take the product except self where self is i)
Except for i==j,multiply all nums[j] in the array with each other to get the product excluding nums[i].
5.After coming out of the inside for loop,store the product into a new result array.

Code:

```
var productExceptSelf = function(nums) {
    var array=[];
    for(var i=0;i<nums.length;i++){
        let product=1;
        for(var j=0;j<nums.length;j++){
            if(j!=i){
                product*=nums[j];
            }
        }
        array.push(product);
    }
    return array;
};
```

Time Complexity:O(n^2)

Space Complexity:O(1)

Solution-2:

Algorithm:

1.Given the array nums,Calculate the prefix product array for the given array.

2.Calculate the suffix product array for the given array.

Example:	0	1	2	3	4
	1	2	3	4	5
Prefix product:	1	2	6	24	120
Suffix product:	120	120	60	20	5

Generally if we find the product except self,

Like for index=2,the product except 2nd index element will be 1*2*4*5=40.

Same if we calculator this product using prefix and suffix array,

i=2,

pp=i-1th value=2(in prefix product array)

sp=i+1th value=20(in suffix product array)

pp*sp will give you the product except ith index value.

3.Take care of the edge cases like for i=0 there is only i+1(in suffix array)

For i=array,length-1,there is only i-1(if prefix array)

Code:

```
var productExceptSelf = function(nums) {
    var prefixProduct=[];
    var suffixProduct=new Array(nums.length);
    var pp=1;
    var sp=1;
    for(var i=0;i<nums.length;i++){
        pp*=nums[i];
        prefixProduct.push(pp);
    }

    for(var i=nums.length-1;i>=0;i--){
        sp*=nums[i];
        suffixProduct[i]=sp;
```

```
    }
    console.log(prefixProduct);
    console.log(suffixProduct);
    var array=[];
    for(var i=0;i<nums.length;i++){
        var p=1;
        var s=1;
        if(i>=1){
           p=prefixProduct[i-1];
        }
        if(i<nums.length-1){
            s=suffixProduct[i+1];
        }
        array.push(p*s);
    }
return array
```

Time Complexity:O(n)+O(n)+O(n)->3*O(n)->O(n)
Space Complexity:O(n)+O(n)->2*O(n)->O(n)

Homework Questions:

1.Magical Prefix Sum

Story: In the magical land of Arrayville, each village keeps track of its treasure by maintaining an array of magical values. The villagers are curious about how many ways they can pick a prefix of their treasure array that has a product divisible by a certain number `k`.

Problem: Given an array of integers `arr` and a number `k`, find the number of prefixes of `arr` whose product is divisible by `k`.

Example:

- For `arr = [2, 3, 4, 6]` and `k = 12`:
    - The prefixes `[2, 3, 4]` and `[2, 3, 4, 6]` have products divisible by `12`. So, the result is `2`.

2. Suffix Magic

Story: Sarah is designing a new puzzle game where players need to calculate the product of suffixes to find hidden treasures. In this game, players are given an array and need to determine the number of suffixes whose product is greater than a specified value `x`.

Problem: Given an array of integers `arr` and a number `x`, count the number of suffixes of `arr` whose product is greater than `x`.

Example:

- For `arr = [1, 2, 3, 4]` and `x = 10`:
    - The suffixes `[2, 3, 4]` and `[3, 4]` have products greater than `10`. So, the result is `2`.

tory: A company is analyzing sales data where they need to find the maximum product of any prefix of the sales array to gauge their best performance streak.

Problem: Given an array of positive integers `arr`, calculate the maximum product that can be obtained from any prefix of the array.

Example:

- For `arr = [3, 2, 5, 6]`:
    - Prefix products are `[3, 6, 30, 180]`
    - The maximum product is `180`.

4. Prefix Product Modulo

Story: In a competitive coding challenge, participants are asked to calculate the prefix p**3. Maximum Product of Prefixes**

Sroduct modulo a large prime number for a given array.

Problem: Given an array of integers `arr` and a prime number `p`, compute the prefix product array modulo `p`.

Example:

- For `arr = [2, 3, 5, 7]` and `p = 11`:
    - Prefix product array modulo `11` would be `[2 % 11, (2 * 3) % 11, (2 * 3 * 5) % 11, (2 * 3 * 5 * 7) % 11] = [2, 6, 4, 10]`.