Day-3

Equilibrium index of an array

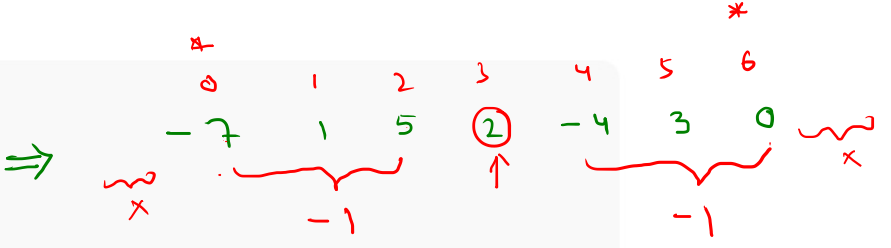
Difficulty Level : Easy • Last Updated : 27 May, 2021

Equilibrium index of an array is an index such that the sum of elements at lower indexes is equal to the sum of elements at higher indexes. For example, in an array A:

Example :

Input: *A[] = {-7, 1, 5, 2, -4, 3, 0}*
Output: *3*
3 is an equilibrium index, because:
A[0] + A[1] + A[2] = A[4] + A[5] + A[6]

Input: *A[] = {1, 2, 3}*
Output: *-1*



Brute-Force

- $O(n^2)$
- $O(1)$

$\Rightarrow$ 3 1 4 4 3 1

1 2 3 3 2 1

1 2 1

0	1	2	3	4	5	6
-7	1	5	2	-4	3	0

L j R

lsum = -7 + 1

```
for( ..... ) →
{
    for( ..... )
    {
        → lsum
    }
    for( ..... )
    {
        → rsum
    }
}
```

$\left.\right\}$ -7+1+5

$\left.\right\}$ -4+3+0

```
int equilibrium(int arr[], int n)
{
    int i, j;
    int leftsum, rightsum;
    for (i = 0; i < n; ++i)
    {
        leftsum = 0;
        for (j = 0; j < i; j++)
            leftsum += arr[j];
        rightsum = 0;
        for (j = i + 1; j < n; j++)
            rightsum += arr[j];
        if (leftsum == rightsum && i!=0 && i!=n-1 )
            return i;
    }

    /* return -1 if no equilibrium index is found */
    return -1;
}
```

0	1	2	3	4	5	6
-7	1	5	2	-4	3	0

$n[n+n] = n^2 + n^2 = 2n^2$

$\therefore O(n^2)$

Ap-2 [Taking 2 arrays]

- → O(n) → T.C ✓
- → O(n) → S.C

arr

0	1	2	3	4	5	6
-7	1	5	2	-4	3	0

left

0	1	2	3	4	5	6
-7	-6	-1	1	-3	0	0

→ loop₁ n

left[i] = left[i-1] + arr[i] ⇐ L to R

right

0	1	2	3	4	5	6
0	7	6	1	-1	3	0

→ loop₂ n

R to L

Right[i] = right[i+1] + arr[i] ⇐

→ loop₃ n

```java
static int equilibrium(int a[], int n)
{
    if (n == 1) return (0);

    int[] front = new int[n];

    int[] back = new int[n];

    for (int i = 0; i < n; i++){

        if (i != 0){
            front[i] = front[i - 1] + a[i];
        }
        else{
            front[i] = a[i];
        }
    }

    for (int i = n - 1; i > 0; i--){

        if (i <= n - 2){
            back[i] = back[i + 1] + a[i];
        }
        else{
            back[i] = a[i];
        }
    }

    for(int i = 0; i < n; i++){

        if (front[i] == back[i]){

            return i;
        }
    }


    // If no equilibrium index found,then return -1

    return -1;
}
```

✓ optimized $\begin{cases} O(n) \\ O(1) \end{cases}$

	0	1	2	3	4	5	6
arr →	-7	1	5	2	-4	3	0

i ✓

Loop$_1$

① sum = $-4 + 3 + 0$ ✓ $= (-1)$

Loop$_2$

② sum = $-7 + 1 + 5 = (-1)$

$\Rightarrow$ $n^2 + 1 \longrightarrow n + n \longrightarrow n + n + 1$ ✱

```
int equilibrium(int arr[], int n)
{
    int sum = 0;
    int leftsum = 0;
    for (int i = 0; i < n; ++i)
        sum += arr[i];
    for (int i = 0; i < n; ++i) {
        sum -= arr[i]; // sum is now right sum for index i

        if (leftsum == sum)
            return i;

        leftsum += arr[i];
    }

    /* If no equilibrium index found, then return 0 */
    return -1;
}
```

$\rightarrow O(n)$

$\rightarrow O(n)$

P[ ] = { _ _ _ }

Prod = (10 x 3 x 5 x) 6 x 2)

arr[i]

2 *

Given an array arr[] of n integers, construct a Product Array prod[] (of same size) such that prod[i] is equal to the product of all the elements of arr[] except arr[i]. Solve it **without division operator in O(n) time**.

Example :

```
Input: arr[]  = {10, 3, 5, 6, 2}
Output: prod[]  = {180, 600, 360, 300, 900}
3 * 5 * 6 * 2 product of other array
elements except 10 is 180
10 * 5 * 6 * 2 product of other array
elements except 3 is 600
10 * 3 * 6 * 2 product of other array
elements except 5 is 360
10 * 3 * 6 * 5 product of other array
elements except 6 is 300
10 * 3 * 6 * 5 product of other array
elements except 2 is 900
```

⇒

0	i			
10	3	5	6	2
↓	↓	↓	↓	↓
180	600	360	300	900

P = ~~1~~ 10 50

```
int[] prodArray(int arr[], int n)
{
    int product[]=new int[n];
     for(int i=0;i<n;i++)
      {
            for(int j=0;j<n;j++)
            {
                if(i!=j)
                    product[i]=arr[i]*arr[j]
            }
      }
      return product;
}
```

→ Space comp

arr

0	1	2	3	4
10	3	5	6	2

→ $O(n^2)$

→ $O(1)$

$O(n)$ → Algo sahani

$\Rightarrow$

arr

0	1	2	3	4
10	3	5	6	2

L to R

Left

0	1	2	3	4
1	10	30	150	900

$\rightarrow loop_1$

Right

0	1	2	3	4
180	60	12	2	1

$\rightarrow loop_2$

$\leftarrow$ R to L

$\rightarrow loop_3$

$\Rightarrow O(n)$ T.C

+

$\Rightarrow O(n)$ S.C

prod[] = {180, 60, 36, 30, 900}

prod[i] = left(i) * right(i)

taking two arrays

arr

0	1	2	3	4
10	3	5	6	2

taking one array

1) take one product array of size n

2)

```
temp=1
for(i=0;i<n;i++)
{
    product[i]=temp
    temp=temp*arr[i]
}
```

L to R

3)

```
temp=1
for(i=n-1;i>=0;i--)
{
    product[i]=product[i]*temp
    temp=temp*arr[i]
}
```

arr

0	1	2	3	4
10	3	5	6	2

n=5

Product

0	1	2	3	4
1	10	30	150	900

180 600 360 300

temp = 1 2 12 60 1800

→ O(n) + c

+

→ O(n) s.c

1) take one product array of size n

2)

```
temp=1
for(i=0;i<n;i++)
{
     product[i]=temp
     temp=temp*arr[i]
}
```

3)

```
temp=1
for(i=n-1;i>=0;i--)
{
    product[i]=product[i]*temp
    temp=temp*arr[i]
 }
```

Find the duplicates of array [1 <= arr[i] <= n-1]

```
Input : n = 7 and array[] = {1, 2, 3, 6, 3, 6, 1}
Output: 1, 3, 6

Explanation: The numbers 1 , 3 and 6 appears more
than once in the array.

Input : n = 5 and array[] = {1, 2, 3, 4 ,3}
Output: 3

Explanation: The number 3 appears more than once
in the array.
```

n → 0 to n-1

n = 7

1 to 6

→ n + 1

→ n + n

→ n + 1

arr

0	1	2	3	4	5	6
1	2	3	6	3	6	1

0 to 6

7

17

arr

0	1	2	3	4	5	6
1	-2	-3	-6	3	6	-1
i	i	i	i	i	i	i

Abs ⇒ +ve value

① j = Abs(arr[i])

3, 6, 1

j = ~~1~~ ~~2~~ 3 ~~4~~ 3 6 1

```java
void printRepeating(int arr[], int size)
    {

        int i;
        System.out.println("The repeating elements are : ");


        for (i = 0; i < size; i++) {

            int j = Math.abs(arr[i]);

            if (arr[j] >= 0)
                arr[j] = -arr[j];
            else
                System.out.print(j + " ");
        }
    }
```

n = 7

1 to 6

0	1	2	3	4	5	6
1	2	3	6	3	6	1

→ n + 1