17. Rearrange an array with O(1) extra space

Medium Accuracy: 54.65% Submissions: 20753 Points: 4

Given an array **arr[]** of size **N** where every element is in the range from **0 to n-1**. Rearrange the given array so that **arr[i]** becomes **arr[arr[i]]**.

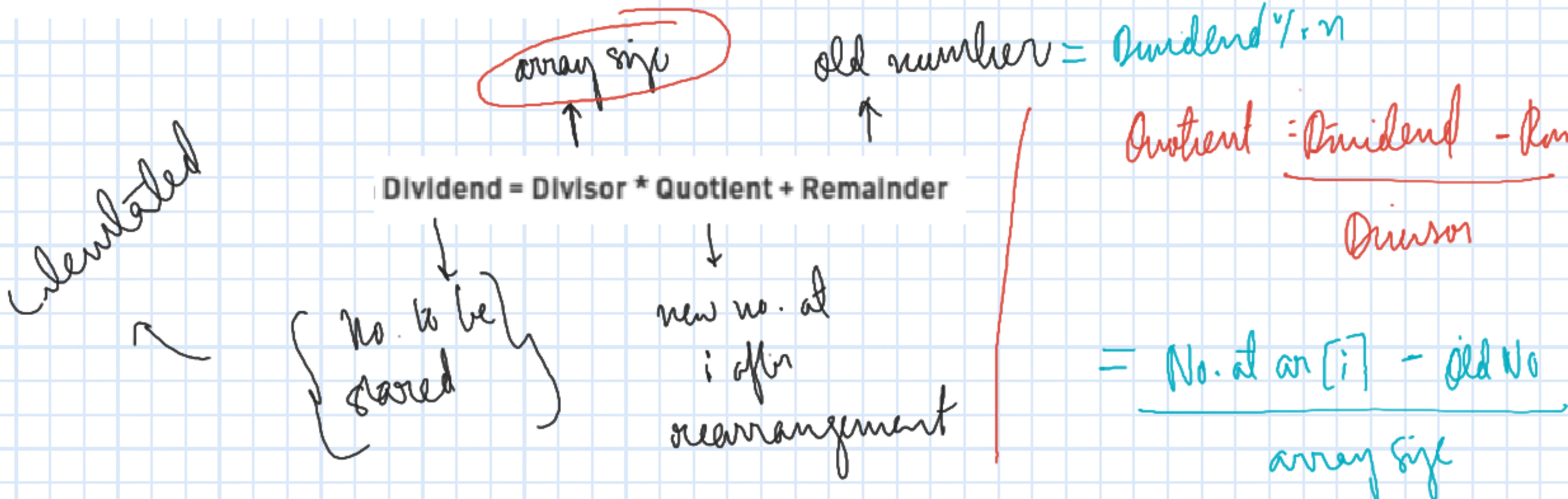
Example 1:

```
Input:
N = 2
arr[] = {1,0}
Output: 0 1
Explanation:
arr[arr[0]] = arr[1] = 0.
arr[arr[1]] = arr[0] = 1.
```

Here, We will use the formula **Dividend = Divisor * Quotient + Remainder**
where Divisor = size of array
Quotient = New number at index i after rearrangement
Remainder = Old Number at index i before rearrangement
Dividend = The number stored at index i

While Traversing the array, we will Look for the value at arr[arr[i]] (which is to be stored at index i), multiply it with Divisor (size of array), and add the old value present at arr[i] to it. Divisor is a value which is higher then values in array (in this case n - size of array, as array elements are between 0 to n-1)

Obviously, don't forget to remove the multiplier n from the values while accessing and outputting the new values.

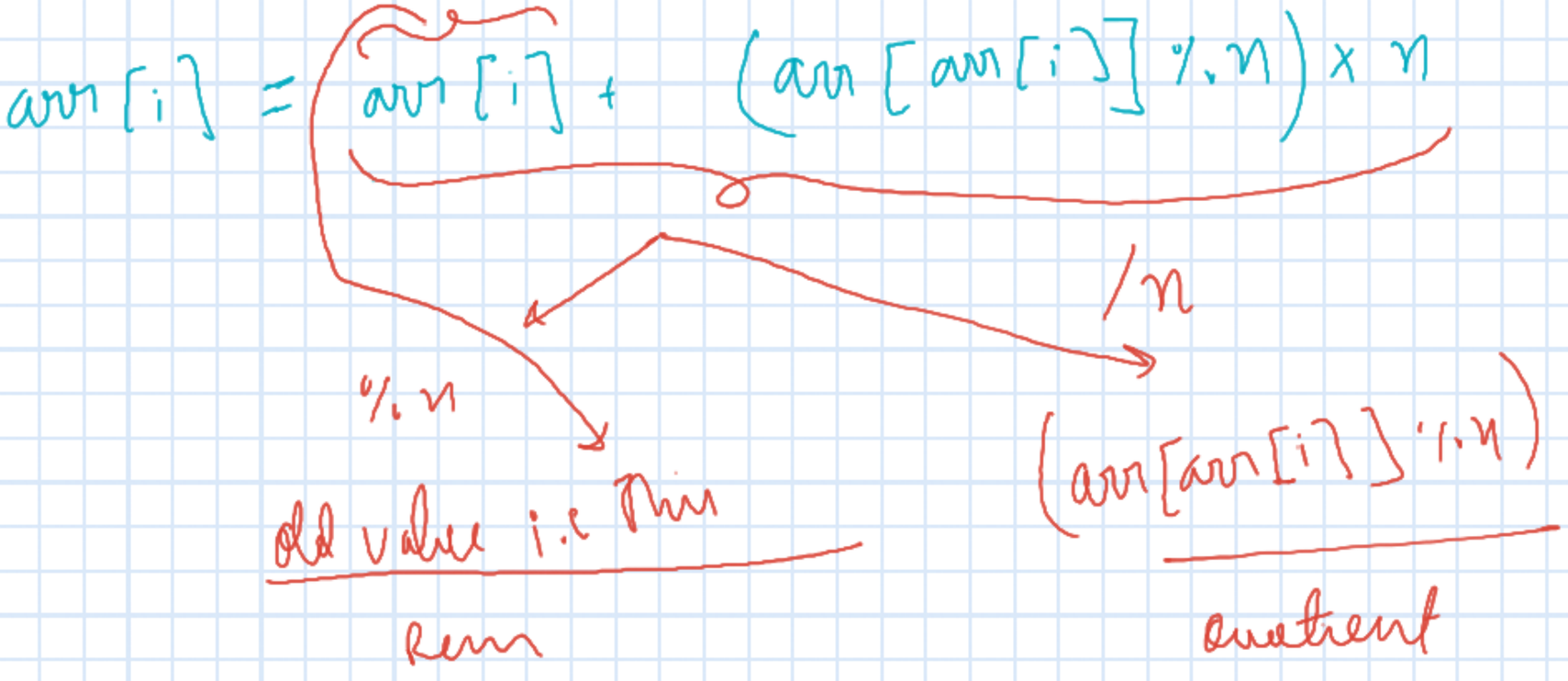


Algorithm:

1. Traverse the array from start to end.
2. For every index increment the element by *array[array[index]]* % n. To get the ith element find the modulo with n, i.e array[index]%n.
3. Again Traverse the array from start to end
4. Store the ith element after dividing the ith element by n, i.e. array[i]/n

Dividend = Divisor × Quotient + Rem

Approach: The array elements of the given array lies from 0 to n-1. Now an array element is needed that can store two different values at the same time. To achieve this increment every element at *i*th index is incremented by *(arr[arr[i]] % n)*n*.After the increment operation of first step, every element holds both old values and new values. Old value can be obtained by *arr[i]%n* and a new value can be obtained by *arr[i]/n*.



Algorithm:

1. Traverse the array from start to end.
2. For every index increment the element by *array[array[index]]* % n. To get the ith element find the modulo with n, i.e array[index]%n.
3. Again Travsese the array from start to end
4. Print the ith element after dividing the ith element by n, i.e. array[i]/n.

$arr[i] = arr[i] + arr[arr[i]] \% n$

$\{ a + b \% n \} \% n$

{1, 2, 3, 4, 0, 4}

$1 + \{\{2\} \% 5\} \times 5$

$0 + \{\{11\} \% 5\} \times 5$

11 /n → 2

i^th element

a

Algorithm:

$$y = a + \{\overline{b \;\%_c\; n}\} \times n$$

- a → $arr[i]$
- $\overline{b \;\%_c\; n}$ → $arr[arr[i]]$

given a & b $arr < n$

$\therefore b \%_c n \rightarrow b$ for unchanged $arr[i]$ and would be diff for the one on which transformation

① has been applied

$y/_c n$ { on this case $y > n$, and will give $arr[i]$ original as answer.

Basically the concept that we're using is

Dividend = Divisor × Quotient + Rem

n

arr[arr[i]]

arr[i]

i

/ n

%. n

Rem arr[j] < n

Else arr[j] > n

Simply arr[j]
if not transformed

arr[j] % n

give Quotient

```
class Solution{
    public:
    //Function to rearrange an array so that arr[i] becomes arr[arr[i]]
    //with O(1) extra space.
    void arrange(long long arr[], int n) {
        int i;

        //Increasing all values by (arr[arr[i]]%n)*n to store the new element.
        for(i=0;i<n;i++){
            arr[i]+=(arr[arr[i]]%n)*n;
        }

        //Since we had multiplied each element with n.
        //We will divide by n too to get the new element at that
        //position after rearranging.
        for(i=0;i<n;i++){
            arr[i]=arr[i]/n;
        }
    }
};
```