→ Given an array which only contains 0s & 1s in a random shuffled order, rearrange the array such that all the 0s are placed before 1s.

Input :- [1, 0, 0, 1, 0]

Output :- [0, 0, 0, 1, 1]

- Conditions :- 1) You can iterate over the array only once.
  2) Do not create any additional array. Do it in-place.

* Solution :-

Approach 1 :- Bruteforce

→ Consider if we don't have to follow the mentioned condition, then we can just iterate over the array, count the no. of 0s & update the first count elements of the array with 0 & the next count elements to 1.

```
function sort01(arr) {
    let countZero = 0;
    for (let i = 0; i < arr.length; i++) {
        if (arr[i] === 0) countZero++;
    }
    for (let i=0; i < countZero; i++) {
        arr[i] = 0;
    }
    for (let i= countZero; i < arr.length; i++) {
        arr[i] = 1;
    }
}
```

Time complexity = O(n)

Space complexity = O(1)

Approach 2 :- Simplest approach

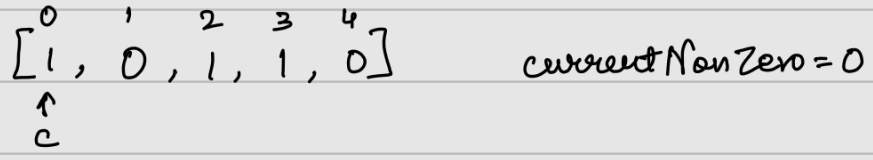
[1, 0, 1, 1, 0]

→ The element at index 0 is not at its correct position.

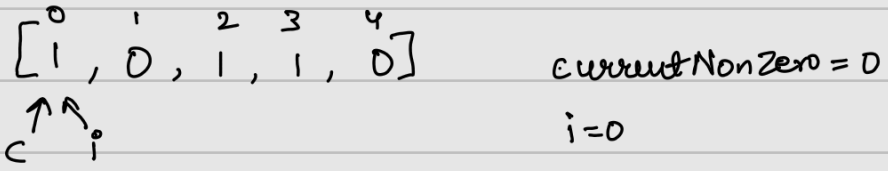
→ We don't know how many 0s exist in the array, but we are sure that

at index 0, 0 should be placed.

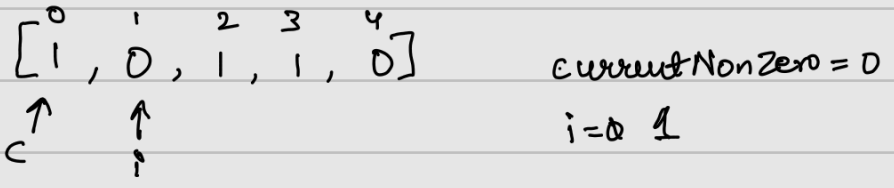
→ We create a variable currentNonZero, which keeps a track on where we need to place 0.



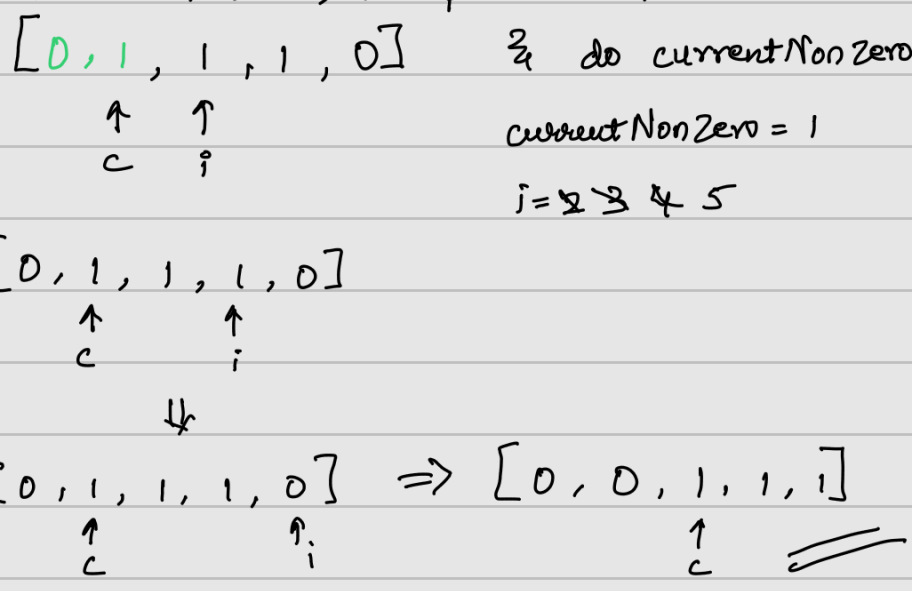
→ We iterate over the array & check if the current element is 0



→ Check if arr[i] === 0 → false, then do i++.



→ Check if arr[i] === 0, true, swap (arr[i], arr[currentNonZero]).



#Code :-

```
function sort01 (arr) {
    let currentNonZero = 0;
    for (let i=0; i<arr.length; i++) {
        if (arr[i] === 0) {
            [arr[i], arr[currentNonZero]] = [arr[currentNonZero],
                                             arr[i]];
            currentNonZero ++;
```

}
}
}

Approach 3:- Two pointer approach

arr = [1, 1, 0, 0]

→ Consider i = 0 & j = arr.length - 1.

→ By observations, we see that at arr[i], 0 should be placed & at arr[j], 1 should be placed.

→ Since, arr[i] = 1 & arr[j] = 0, we can swap these.

[1, 1, 0, 0] —swap→ [0, 1, 0, 1]

→ Since, arr[i] == 0, i++ & arr[j] == 1, j--.

[0, 1, 0, 1]

→ Same steps to be followed.

→ But what if arr[i] = 1 & arr[j] = 1

[1, 0, 1]

→ In this case, if arr[i] = 1, then swap it with j

→ We don't know what element is at j, but one thing is assurred that after swapping, arr[j] = 1 & it will be at its correct position.

[1, 0, 1] —swap⇒ [1, 0, 1] ⇒ j--

[1, 0, 1]

→ Check if arr[i] = 1 → true, swap arr[j] then j--

" " " → false, i++ because 0 will be present at i.

Code

Code :-

```
function sort01 (arr) {
    let i=0, j= arr.length-1;
    while (i<=j) {
      if (arr[i]===1) {
          [arr[i], arr[j]] = [arr[j], arr[i]];
          j--;
      } else {
          i++;
      }
    }
```

Time complexity = O(n)

Space complexity = O(1)