Move 0s To The End II(Easy)

Given an array of integers, move all the 0s to the right end of the array. The relative order of the elements in the original array **need to be maintained**.

Assumptions:

- The given array is not null.

Examples:

- {1} --> {1}
- {1, 0, 3, 0, 1} --> {1, 3, 1, 0, 0}

Same answer as Move 0s To The End I (Easy) in class2 homework.

//Ok, good!

Rainbow Sort III(Medium)

Given an array of balls with k different colors denoted by numbers 1- k, sort the balls.

Examples

- k=1, {1} is sorted to {1}
- k=3, {1, 3, 2, 1, 2} is sorted to {1, 1, 2, 2, 3}
- k=5, {3, 1, 5, 5, 1, 4, 2} is sorted to {1, 1, 2, 3, 4, 5, 5}

Assumptions

- The input array is not null.
- k is guaranteed to be >= 1.
- k << logn.

Same answer as Rainbow Sort (Medium) in class2 homework.

//Ok, no problem.

Array Deduplication II(Medium)

Given a **sorted** integer array, remove duplicate elements. For each group of elements with the same value keep at most two of them. Do this in-place, using the left side of the original array and maintain the relative order of the elements of the array. Return the array after deduplication.

Assumptions

- The given array is not null

Examples

- {1, 2, 2, 3, 3, 3} → {1, 2, 2, 3, 3}

```
int removeDuplicates2(vector<int>& arr){
    if(arr.empty())
        return 0;
    int n = arr.size();
    int curIndex = 0;
    for(int i = 0; i < n; i++) {
        if (curIndex < 2 || arr[i] > arr[curIndex - 2])
            arr[curIndex++] = arr[i];
    }
    return curIndex;
}
```

Time Complexity: O(n), we traverse the array one time.
//Ok, great!

Array Deduplication III(Medium)

Given a sorted integer array, remove duplicate elements. For each group of elements with the same value do not keep any of them. Do this in-place, using the left side of the original array and maintain the relative order of the elements of the array. Return the array after deduplication.

Assumptions

- The given array is not null

Examples

- {1, 2, 2, 3, 3, 3} → {1}

```
void removeDuplicates3(vector<int>& arr){
    if(arr.empty())
        return;
    int n = arr.size();
    int slow = 0, fast = 0, cur = 0;
    int count = 0;
    while(fast < n){
        if(arr[fast] == arr[slow]) {
            fast++;
            continue;
        }
        if(fast - slow == 1)
            arr[cur++] = arr[slow];
        slow = fast;
    }
    // check if the last element is unique
    if(slow == n - 1)
        arr[cur++] = arr[slow];
    // erase the elements at the end of the array
    arr.erase(arr.begin() + cur, arr.end());
}
```

Time Complexity: O(n), we traverse the array one time.
//Ok, no problem!

Binary Search II

Shift Position (Medium)

Given an integer array A, A is sorted in ascending order first then shifted by an arbitrary number of positions, For Example, A = {3, 4, 5, 1, 2} (shifted left by 2 positions). Find the index of the smallest number.

Assumptions

- There are no duplicate elements in the array

Examples

- A = {3, 4, 5, 1, 2}, return 3
- A = {1, 2, 3, 4, 5}, return 0

Corner Cases

- What if A is null or A is of zero length? We should return -1 in this case.

```
int findMin(vector<int>& arr){
    if(arr.empty())
        return -1;
    int left = 0, right = arr.size() - 1;

    // it is a sorted array without rotation
    if(arr[left] < arr[right])
        return arr[left];// return left;

    while(left < right){
        int mid = left + (right - left) / 2;
        if(arr[mid] > arr[mid + 1])
            return arr[mid + 1];// etc..
        if(arr[mid] > arr[0])
            left = mid + 1;
        else
            right = mid;
    }
    return arr[left];// etc..
}
```

Time Complexity: O(logn), same as binary search.

//Ok, excellent method! But you should return the index!