October 21, 2021 ▪ Arrays / Data Structure

Sort an array of 0s, 1s and 2s

Problem Statement: Given an array consisting of only 0s, 1s and 2s. Write a program to in-place sort the array without using inbuilt sort functions. (Expected: Single pass-O(N) and constant space)

Example 1:

```
Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]

Input: nums = [2,0,1]
Output: [0,1,2]

Input: nums = [0]
Input: nums = [0]
```

Solution

***Disclaimer**: Don't jump directly to the solution, try it out yourself first.*

Solution 1: Sorting (even if it is not the expected solution here but it can be considered as one of the approaches). Since the array contains only 3 integers, 0, 1, and 2. Simply sorting the array would arrange the elements in increasing order.

Time Complexity: O(N*logN)

Space Complexity: O(1)

Solution 2: Keeping count of values

Intuition: Since in this case there are only 3 distinct values in the array so it's easy to maintain the count of all, Like the count of 0, 1, and 2. This can be followed by overwriting the array based on the frequency(count) of the values.

Approach:

1. Take 3 variables to maintain the count of 0, 1 and 2.
2. Travel the array once and increment the corresponding counting variables

(let's consider **count_0 = a, count_1 = b, count_2 = c**)

3. In 2nd traversal of array, we will now over write the first 'a' indices / positions in array with '0', the next 'b' with '1' and the remaining 'c' with '2'.

Time Complexity: O(N) + O(N)

Space Complexity: O(1)

Solution 3: 3-Pointer approach

This problem is a variation of the *popular **Dutch National flag algorithm***

Intuition: In this approach, we will be using 3 pointers named low, mid, and high. We will be using these 3 pointers to move around the values. The primary goal here is to move 0s to the left and 2s to the right of the array and at the same time all the 1s shall be in the middle region of the array and hence the array will be sorted.

Approach:

1. Initialize the 3 pointers such that low and mid will point to 0th index and high pointer will point to last index

int low = arr[0]

int mid = arr[0]

int high = arr[n – 1]

2. Now there will 3 different operations / steps based on the value of arr[mid] and will be repeated until mid <= high.

Subscribe

I want to receive latest posts and interview tips

Name*

John

Email*

abc@gmail.com

Join takeUforward

Search

Search

Recent Posts

- Find the highest/lowest frequency element
- A Guide on Online C Compiler
- Burst Balloons | Partition DP | DP 51
- Evaluate Boolean Expression to True | Partition DP: DP 52
- Palindrome Partitioning – II | Front Partition : DP 53

Accolite Digital Amazon Arcesium Bank of America Barclays BFS Binary Search Binary Search Tree Commvault CPP DE Shaw DFS DSA Self Paced google HackerEarth infosys inorder Java Juspay Kreeti Technologies Morgan Stanley Newfold Digital Oracle post order queue recursion Samsung SDE Core Sheet SDE Sheet Searching set-bits sorting Strivers A2ZDSA Course sub-array subarray Swiggy takeuforward TCQ NINJA TCS TCS CODEVITA TCS DIGITA; TCS Ninja TCS NQT VMware XOR

1.
   arr[mid] == 0:
   swap(arr[low], arr[mid])
   low++, mid++

2.
   arr[mid] == 1:
   mid++

3.
   arr[mid] == 2:
   swap(arr[mid], arr[high])
   high–;

The array formed after these steps will be a sorted array.

C++ Code

```cpp
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int lo = 0;
        int hi = nums.size() - 1;
        int mid = 0;

        while (mid <= hi) {
            switch (nums[mid]) {

            // If the element is 0
            case 0:
                swap(nums[lo++], nums[mid++]);
                break;

            // If the element is 1 .
            case 1:
                mid++;
                break;

            // If the element is 2
            case 2:
                swap(nums[mid], nums[hi--]);
                break;
            }
        }

    }
};
```

Java Code

```java
class Solution {
    public void sortColors(int[] nums) {
        int lo = 0;
        int hi = nums.length - 1;
        int mid = 0;
        int temp;
        while (mid <= hi) {
            switch (nums[mid]) {
                case 0: {
                    temp = nums[lo];
                    nums[lo] = nums[mid];
                    nums[mid] = temp;
                    lo++;
                    mid++;
                    break;
                }
                case 1:
                    mid++;
                    break;
                case 2: {
                    temp = nums[mid];
                    nums[mid] = nums[hi];
                    nums[hi] = temp;
                    hi--;
                    break;
                }
            }
        }
    }
}
```

Python Code

```python
from typing import List


class Solution:
    def sortColors(self, nums: List[int]) -> None:
        lo = 0
        hi = len(nums) - 1
        mid = 0
        while mid <= hi:
            if nums[mid] == 0:
                nums[lo], nums[mid] = nums[mid], nums[lo]
                lo += 1
                mid += 1
            elif nums[mid] == 1:
                mid += 1
            elif nums[mid] == 2:
                nums[mid], nums[hi] = nums[hi], nums[mid]
                hi -= 1
```

Time Complexity: O(N)

Space Complexity: O(1)

> Special thanks to **Aditya Shahare** and **Sudip Ghosh** for contributing to this article on takeUforward. If you also wish to share your knowledge with the takeUforward fam, please check out this article.

Sort an array of 0's 1's & 2's | Leetcode | C++ and Java | Brute-Better-Optimal

« Previous Post
Find middle element in a Linked List

Next Post »
Barclays Graduate Analyst Interview Experience | Set 1

Load Comments