Dutch National Flag Algorithm

Problem Statement

Given an array consisting of only 0s, 1s and 2s, sort the array.

Naive Approach:

Simply sort the array with the help of sorting algorithms like Merge Sort, Quick Sort. This gives a time complexity of O(N*logN), where N is the number of elements in the array.

Two-Pass Algorithm:

The solution involves iterating through the original array and counting the number of 0s, 1s, and 2s, and just overwriting the original array in a second pass. The only disadvantage is that we need to traverse the array twice to get a sorted array.

Steps:

- Traverse the array once and keep track of the count of 0s, 1s and 2s encountered.
- Now traverse the array again and overwrite the array starting from the beginning, first with 0s, then 1s, and finally all 2s.

Pseudocode:

```
/*
        array of size N from 0 to N-1 is considered
*/
function sort012(arr, N)

        //      Initialize the cnt0, cnt1 and cnt2 variables to 0.
        cnt0 = 0
        cnt1 = 0
        cnt2 = 0

        //      Count the number of 0s, 1s and 2s
        for idx = 0 to N-1
                if arr[idx] == 0
                        cnt0 += 1
                else if arr[idx] == 1
                        cnt1 += 1
                else
                        cnt2 += 1

        //      Now overwrite the array from the beginning
        for idx = 0 to N-1
                if cnt0 > 0
                        arr[idx] = 0
                        cnt0 -= 1
                else if cnt1 > 0
                        arr[idx] = 1
                        cnt1 -= 1
                else
                        arr[idx] = 2
                        cnt2 -= 1
```

Time complexity: O(N), where N is the number of elements in the array, as we traverse the array twice only.

Space complexity: O(1), as no extra space is required.

Dutch National Flag algorithm or Three-way partitioning

The Dutch National Flag algorithm or three-way partitioning algorithm allows sorting the array consisting of 0s, 1s, and 2s in a single traversal only and in constant space.

Steps:

- Maintain three indices low = 0, mid = 0, and high = N-1, where N is the number of elements in the array.
    1. The range from 0 to low denotes the range containing 0s.
    2. The range from low to mid denotes the range containing 1s.
    3. The range from mid to high denotes the range containing any of 0s, 1s, or 2s.
    4. The range from high to N-1 denotes the range containing 2s.
- The mid pointer denotes the current element, traverses the array while mid<=high i.e we have exhausted the search space for the range which can contain any of 0s, 1s, or 2s.
    1. If A[mid] == 0, swap A[mid] and A[low] and increment low and mid pointers by 1.
    2. If A[mid] == 1, increment the mid pointer by 1.
    3. If A[mid] == 2, swap A[high] and A[mid] and increment mid by 1 and decrement high by 1.

The resulting array will be a sorted array containing 0s, 1s, and 2s.

Pseudocode:

```
/*
    array of size N from 0 to N-1 is considered
*/
function DNF(arr, N)

    //    Initializing low, mid and high pointers
    low = 0
    mid = 0
    high = N-1

    while mid <= high
        /*
    Check if arr[mid] == 0, swap arr[low] and arr[mid], increment mid and
    low pointers
        */
        if arr[mid] == 0
            swap(arr[mid],arr[low])
            low += 1
            mid += 1

        //    Check if arr[mid] == 1, increment mid pointer
        else if arr[mid] == 1
            mid += 1

        /*
    Check if arr[mid] == 2, swap arr[mid] and arr[high], decrement high pointer and
increment low pointer
        */
        else if arr[mid] == 2
            swap(arr[mid],arr[high])
            high -= 1
```

Time complexity: O(N), where N is the number of elements in the array, as we sort the array in a single traversal only.

Space complexity: O(1), as no extra space is required.