

3.

1 Sort Array of 0s, 1s, 2s

1

MEDIUM

■ What You Will Learn

- ✓ Dutch National Flag algorithm by Dijkstra
- ✓ Three pointers: low, mid, high
- ✓ $O(n)$ single-pass in-place sort for this specific case

■ Intuition First

Imagine sorting a flag with three colors — red, white, blue (0, 1, 2). You want all reds left, whites middle, blues right. Three pointers divide the array into four regions: sorted 0s | unsorted region | sorted 2s. The mid pointer marches through the unsorted zone.

■ Three Pointers

- low: boundary of 0s (elements before low are all 0)
- mid: current element being examined
- high: boundary of 2s (elements after high are all 2)
- Invariant: $[0..low-1]=0$, $[low..mid-1]=1$, $[mid..high]=\text{unknown}$, $[high+1..n-1]=2$

■ The 3 Cases

Case $arr[mid]=0$: `swap(low,mid), low++, mid++`

Case $arr[mid]=1$: `mid++` (already in place)

Case $arr[mid]=2$: `swap(mid,high), high--` (don't move mid!)

Step 1

Start: $low=0, mid=0, high=4$

2	0	1	1	0
---	---	---	---	---

Step 2

$arr[mid]=2 \rightarrow \text{swap}(0,4), high=3$

0	0	1	1	2
---	---	---	---	---

Step 3

$arr[mid]=0 \rightarrow \text{swap}(low=0, mid=0), low=1, mid=1$

0	0	1	1	2
---	---	---	---	---

Step 4 arr[mid]=0→swap(low=1,mid=1),low=2,mid=2

0	0	1	1	2
---	---	---	---	---

Step 5 arr[mid]=1→mid=3; arr[mid]=1→mid=4>high: DONE

0	0	1	1	2
---	---	---	---	---

■ Java Code

```
public static void sort012(int[] arr) {
    int low = 0, mid = 0, high = arr.length - 1;
    while (mid <= high) {
        if (arr[mid] == 0) {
            // Swap with low section
            int t=arr[low]; arr[low]=arr[mid]; arr[mid]=t;
            low++; mid++;
        } else if (arr[mid] == 1) {
            mid++; // Already in correct section
        } else {
            // Swap with high section (don't increment mid)
            int t=arr[mid]; arr[mid]=arr[high]; arr[high]=t;
            high--;
        }
    }
}
```

■ ■ Complexity

Case	Time	Space	What happens?
All Cases	O(n)	O(1)	Single pass — mid goes from 0 to high

■ ■ Why Not mid++ When Swapping with high?

- When you swap from high, the new arr[mid] is unknown — it hasn't been examined yet
- If you incremented mid, you'd skip examining it!
- When swapping with low, arr[low] was already processed (it was a 1) — safe to advance

■ Why This Over Counting Sort?

- Counting sort: count 0s, 1s, 2s then overwrite → 2 passes
- Dutch Flag: 1 pass, no counting — faster in practice
- Both $O(n)$ time but Dutch Flag is better constant factor

■ Practice Problems

[Med] Sort array with 4 distinct values

[Med] Segregate even and odd using same idea

[Hard] Sort colors where each color is a range (generalized DNF)

■ Quick Revision

- Three pointers: low, mid, high
- 0 → swap(low,mid), low++, mid++
- 1 → mid++ only
- 2 → swap(mid,high), high-- (no mid++)
- $O(n)$ time, $O(1)$ space — single pass