Array Lab-2

Direct question: Sort an Array of 0s, 1s, and 2s: Given an array containing only 0s, 1s, and 2s, sort the array in linear time.

Can be asked as:

Given balls of these three colors arranged randomly in a line (it does not matter how many balls there are), the task is to arrange them such that all balls of the same color are together and their collective color groups are in the correct order.

[Naive Approach] Sorting – O(N * log(N)) Time and O(1) Space:

The naive solution is to simply **sort the array using a standard sorting algorithm** or sort library function. This will simply place all the 0s first, then all 1s and 2s at last. This approach requires **O(N * log(N)) time** and **O(1) space**.

[Better Approach] Counting 0s, 1s and 2s – Two Pass – O(N) Time and O(1) Space:

A better solution is to t**raverse the array once and count number of 0s, 1s and 2s**. **Let these counts be c0, c1 and c2.**

Now **traverse the array again**, **put c0 (count of 0s) 0s first, then c1 1s and finally c2 2s**.

This solution works in **O(n) time**, but this solution is not stable and requires **two traversals** of the array. This approach requires to traverse the array twice, therefore **O(N) time** and **O(1) space**.

[Expected Approach] Dutch National Flag Algorithm – One Pass – O(N) Time and O(1) Space:

The idea is to sort the array of **size N** using **three pointers**: **lo = 0, mid = 0 and hi = N – 1** such that the array is divided into three parts:

arr[0] to arr[lo – 1]: This part will have all the zeros.

arr[lo] to arr[mid – 1]: This part will have all the ones.

arr[hi + 1] to arr[N – 1]: This part will have all the twos.

Steps:

1. Initialize Three Pointers:

low pointer starts at the beginning of the array (i**ndex 0**).

mid pointer also starts at the beginning of the array (**index 0**).

high pointer starts at the end of the array (**index 5** in this case)

2. Traverse the Array:

- You will iterate through the array using the mid pointer until it exceeds the high pointer.
- Depending on the value at arr[mid], you'll take different actions:
    - If **arr[mid] == 0**: Swap **arr[mid]** with **arr[low]**, **increment** both **low** and **mid**.
    - If **arr[mid] == 1**: **No swap** is needed; simply **increment mid**.
    - If **arr[mid] == 2**: **Swap arr[mid]** with **arr[high]**, and **decrement high**. (Do not increment mid immediately, because the swapped value at arr[mid] might need to be processed.)

Detailed Execution

- **Initial Array:** [2, 0, 1, 2, 1, 0]
- **low = 0, mid = 0, high = 5**

Iteration 1:

- arr[mid] = 2, swap arr[mid] with arr[high]:
    - Array: [0, 0, 1, 2, 1, 2]
    - high = 4 (decremented)
    - mid = 0 (remains same, need to check arr[mid] again)

Iteration 2:

- arr[mid] = 0, swap arr[mid] with arr[low]:
    - Array: [0, 0, 1, 2, 1, 2]
    - low = 1, mid = 1 (both incremented)

Iteration 3:

- arr[mid] = 0, swap arr[mid] with arr[low]:
    - Array: [0, 0, 1, 2, 1, 2]
    - low = 2, mid = 2 (both incremented)

Iteration 4:

- arr[mid] = 1, no swap needed:
    - Array: [0, 0, 1, 2, 1, 2]
    - mid = 3 (incremented)

Iteration 5:

- arr[mid] = 2, swap arr[mid] with arr[high]:
    - Array: [0, 0, 1, 1, 2, 2]
    - high = 3 (decremented)
    - mid = 3 (remains same, need to check arr[mid] again)

Iteration 6:

- arr[mid] = 1, no swap needed:
    - Array: [0, 0, 1, 1, 2, 2]
    - mid = 4 (incremented)

Termination:

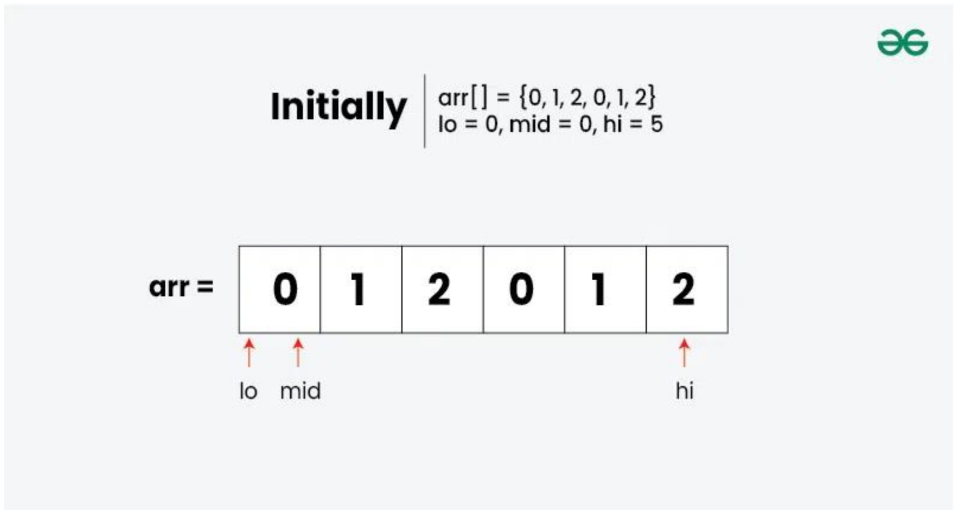
- The loop terminates when mid > high. The array is now sorted.

Final Array: [0, 0, 1, 1, 2, 2]

Time and Space Complexity

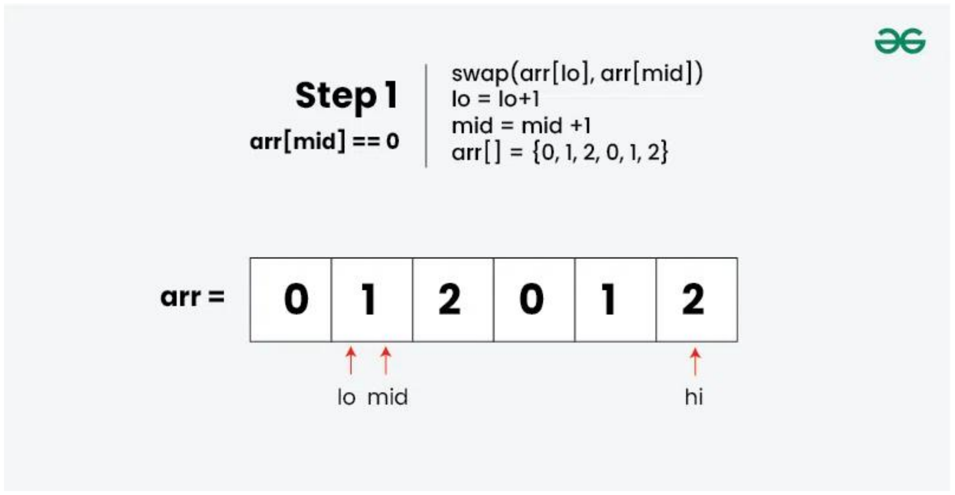
- **Time Complexity:** O(n), where n is the length of the array.
- **Space Complexity:** O(1), as no extra space is used apart from the pointers.

This algorithm is efficient for sorting arrays with three distinct values and is often used in problems involving similar constraints.



Initially

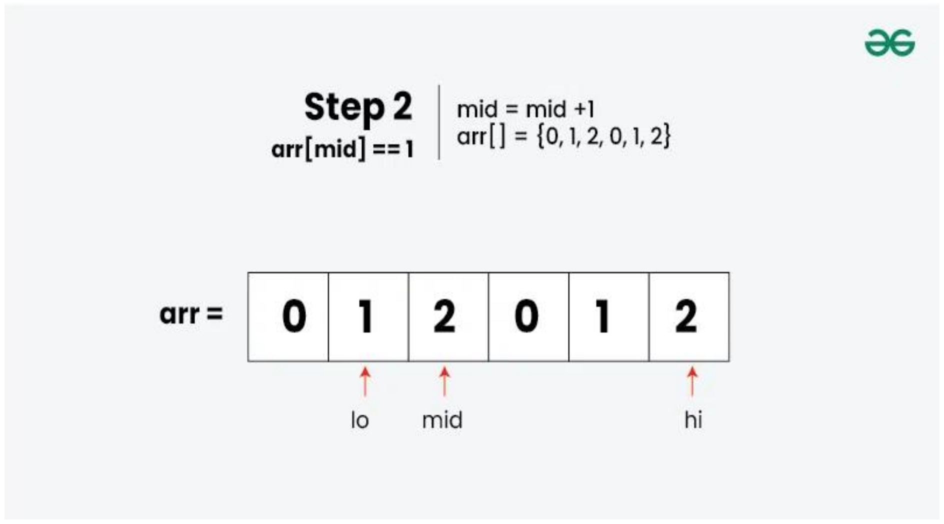
arr[] = {0, 1, 2, 0, 1, 2}
lo = 0, mid = 0, hi = 5



Step 1

arr[mid] == 0

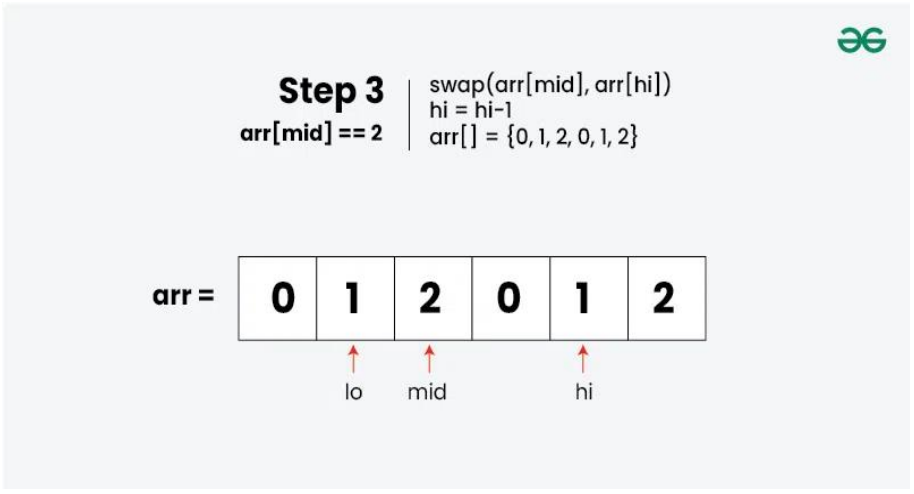
swap(arr[lo], arr[mid])
lo = lo+1
mid = mid +1
arr[] = {0, 1, 2, 0, 1, 2}

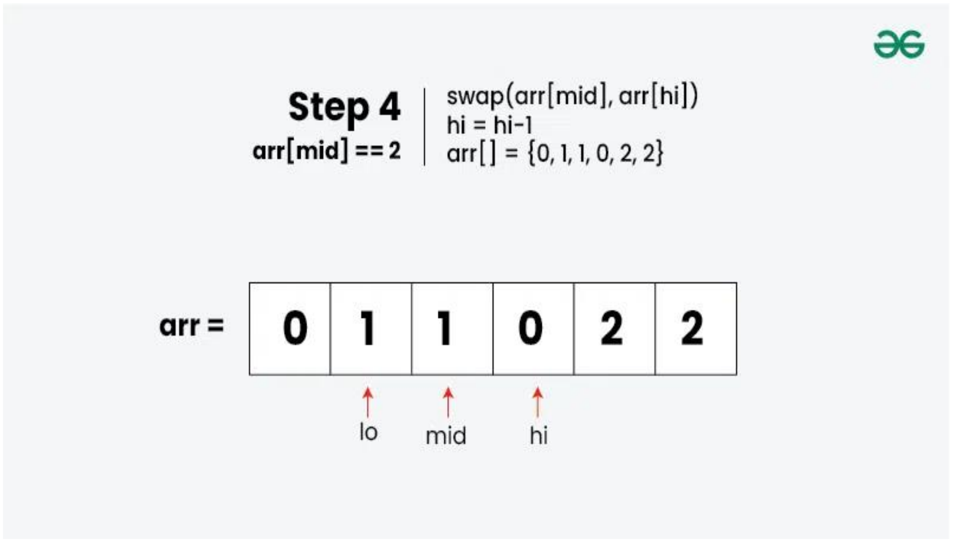


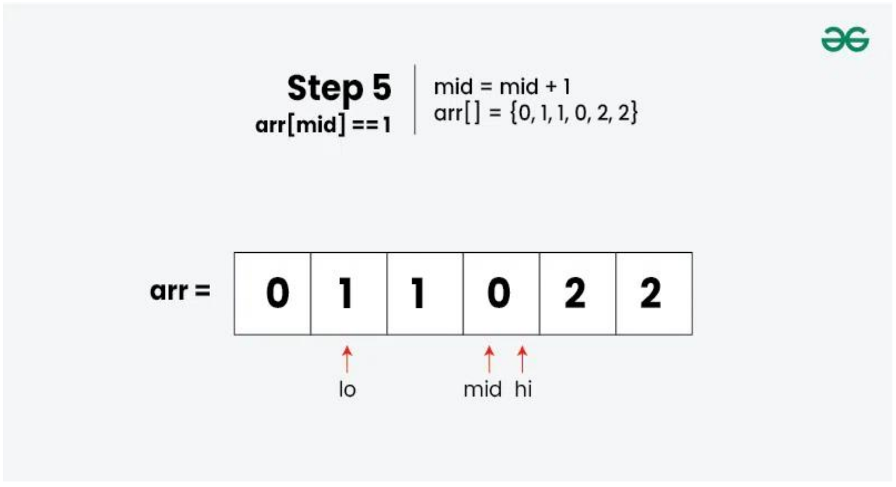
Step 2

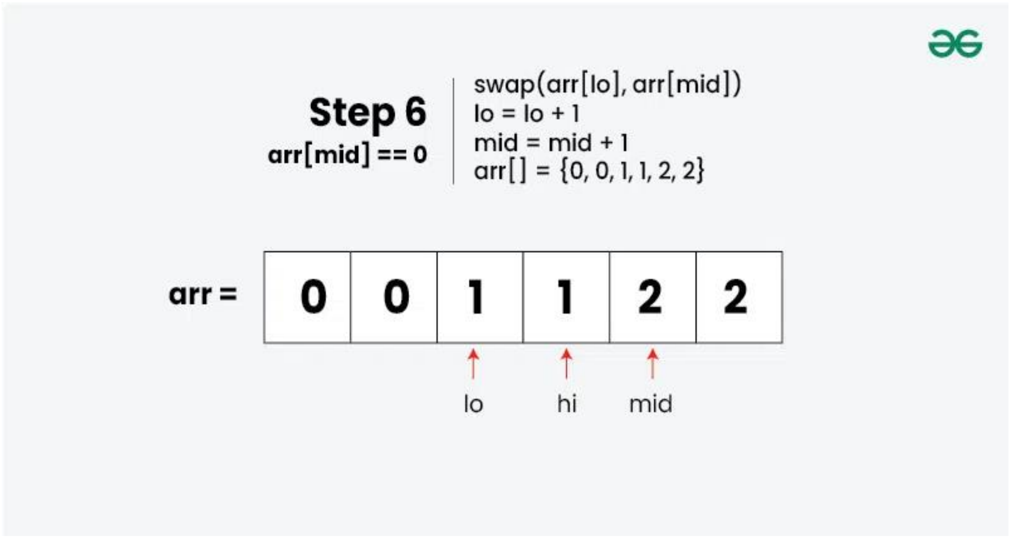
arr[mid] == 1

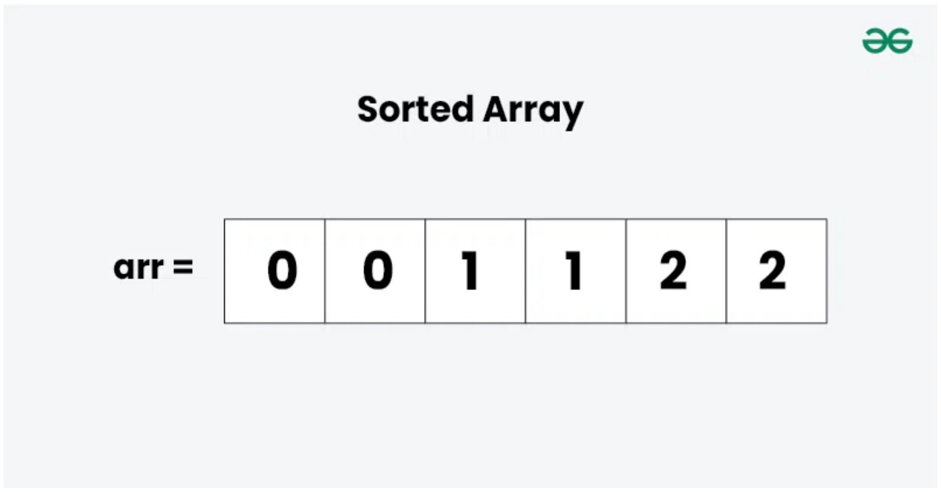
mid = mid +1
arr[] = {0, 1, 2, 0, 1, 2}


Step 3
arr[mid] == 2
swap(arr[mid], arr[hi])
hi = hi-1
arr[] = {0, 1, 2, 0, 1, 2}
arr =
0
1
2
0
1
2
lo
mid
hi


Step 4
arr[mid] == 2
swap(arr[mid], arr[hi])
hi = hi-1
arr[] = {0, 1, 1, 0, 2, 2}
arr =
0
1
1
0
2
2
lo
mid
hi


Step 5
arr[mid] == 1
mid = mid + 1
arr[] = {0, 1, 1, 0, 2, 2}
arr =
0
1
1
0
2
2
lo
mid
hi





C++:

```
// C++ program to sort an array of 0s, 1s and 2s in a single pass
#include <bits/stdc++.h>
using namespace std;

// Function to sort the input array,
// the array is assumed
// to have values in {0, 1, 2}
void sort012(int a[], int n)
{
    int lo = 0;

    int hi = n - 1;

    int mid = 0;


    // Iterate till all the elements

    // are sorted

    while (mid <= hi) {

        switch (a[mid]) {


        // If the element is 0

        case 0:

            swap(a[lo++], a[mid++]);

            break;


        // If the element is 1 .

        case 1:

            mid++;

            break;


        // If the element is 2

        case 2:

            swap(a[mid], a[hi--]);

            break;

        }

    }

}


// Function to print array arr[]

void printArray(int arr[], int n)

{

    // Iterate and print every element
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

// Driver Code
int main()
{
      // Sample Input
    int arr[] = { 0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);

    sort012(arr, n);

    printArray(arr, n);

    return 0;
}
```

Java:

```
// Java program to sort an array of 0s, 1s and 2s in a
// single pass

class countzot {

    // Sort the input array, the array is assumed to
    // have values in {0, 1, 2}
    static void sort012(int a[], int n)
    {
        int lo = 0;
        int hi = n - 1;
        int mid = 0, temp = 0;
    // Iterate till all the elements
    // are sorted
    while (mid <= hi) {
        switch (a[mid]) {
            // If the element is 0
            case 0: {
                temp = a[lo];
                a[lo] = a[mid];
                a[mid] = temp;
                lo++;
                mid++;
                break;
            }
                // If the element is 1
            case 1:
                mid++;
                break;
                // If the element is 2
            case 2: {
                temp = a[mid];
                a[mid] = a[hi];
                a[hi] = temp;
                hi--;
                break;
            }
        }
    }
}

/* Utility function to print array arr[] */
static void printArray(int arr[], int n)
```

```
    {
        int i;
        for (i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
        System.out.println("");
    }

    /*Driver function to check for above functions*/
    public static void main(String[] args)
    {
        int arr[] = { 0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1 };
        int n = arr.length;
        sort012(arr, n);
        printArray(arr, n);
    }
}
```

Python:

```
# Python program to sort an array with
# 0, 1 and 2 in a single pass

# Function to sort array
def sort012(arr, n):
    lo = 0
    hi = n - 1
    mid = 0
    # Iterate till all the elements
    # are sorted
    while mid <= hi:
      # If the element is 0
      if arr[mid] == 0:
        arr[lo], arr[mid] = arr[mid], arr[lo]
        lo = lo + 1
        mid = mid + 1
        # If the element is 1
      elif arr[mid] == 1:
        mid = mid + 1
        # If the element is 2
      else:
        arr[mid], arr[hi] = arr[hi], arr[mid]
        hi = hi - 1
    return arr

# Function to print array
def printArray(arr):
    for k in arr:
        print(k, end=' ')


# Driver Program
arr = [0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1]
n = len(arr)
arr = sort012(arr, n)
printArray(arr)
```