Statement

↳ Given an array, which contains a combination of the following three elements:

- 0 (Representing red)
- 1 (Representing white)
- 2 (Representing blue)

Sort the array in place so that the elements of the same colour are adjacent, and the final order is: red (0), white (1), blue (2).

Naive

↳ Use conventional sorting algorithm, i.e., Merge sort gives $O(n \log n)$.

Optimal

↳ Order elements in single pass

- Initialize pointers
  - start and current: initially point to the first index
  - end: initially point to the last index
- Iterate over array until current pointer is greater than end pointer.
- During iteration, there are 3 conditions:
  - If colours[current] is 0
    - ↳ The current pointer points to red. Swap colours[current] and colours[start]. This places the red element at the start of the array.
    - ↳ Increment both start and current pointers by one. Moving start ensures that the next red element will occupy the correct position.
  - If colours[current] is 1
    - ↳ The white element is already in its correct section, no swapping required.

↳ Increment the current pointer by one.

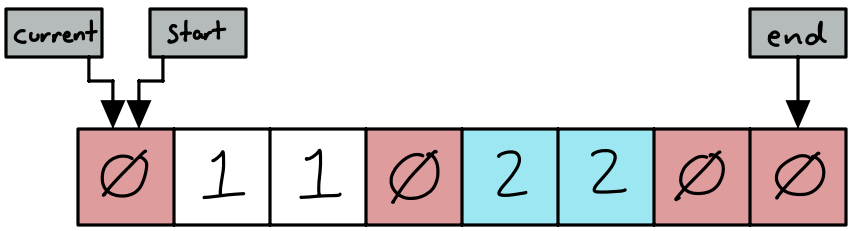
- If colours[current] is 2

↳ The current pointer points to blue. Swap colours[current] and colours[end]. This pushes the blue element to the end of the array.

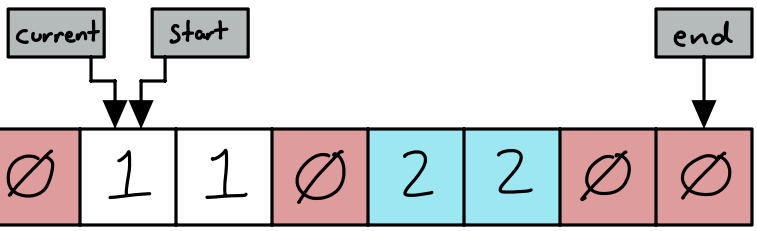
↳ Decrement end pointer by one.

Visualization

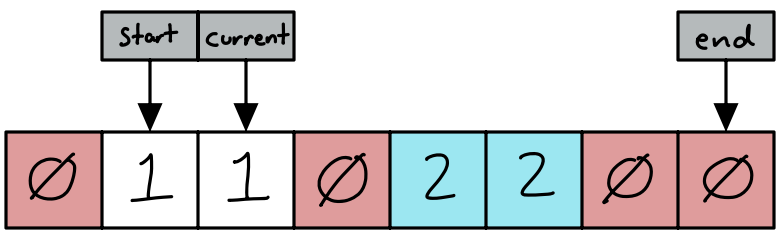
i) colours[current] == Ø so swap the elements of colours[current] and colours[start].
Increment both the current and start pointers.



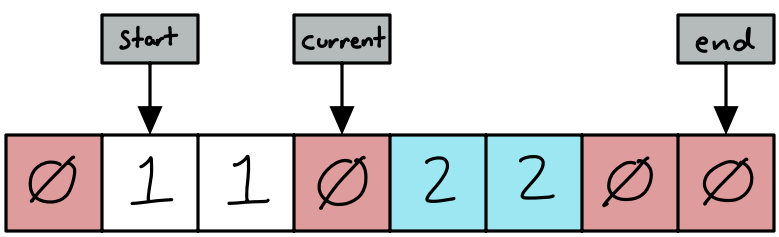
ii) colours[current] == 1
Increment current pointer



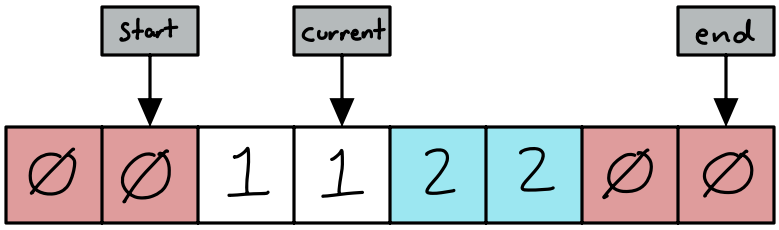
iii) colours[current] == 1
Increment current pointer



iv) colours[current] == Ø
Swap the elements of colours[current] and colours[start].

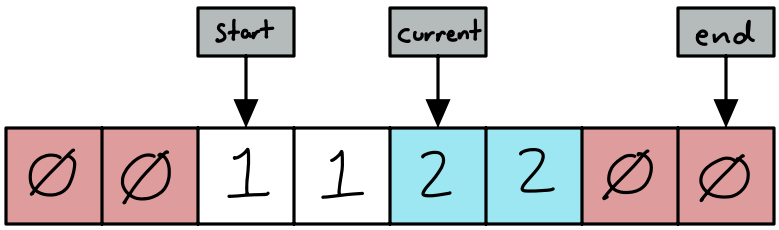


v) Increment both current and start pointers.



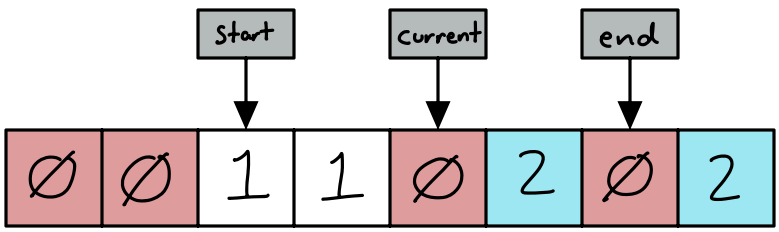
vi) colours[current] == 2

Swap the elements of colours[current] and colours[end] and decrement the end pointer.

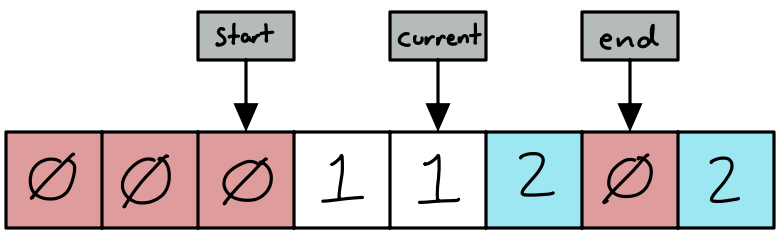


vii) colours[current] == ∅

Swap the elements of colours[current] and colours[start].

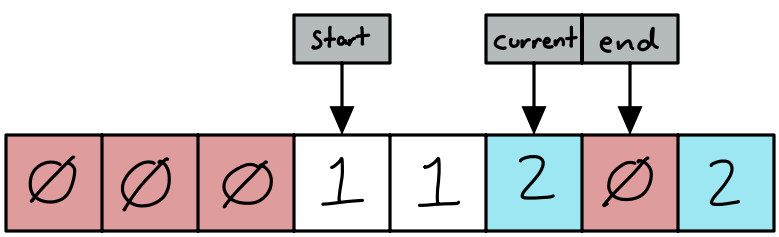


viii) Increment start and current pointers



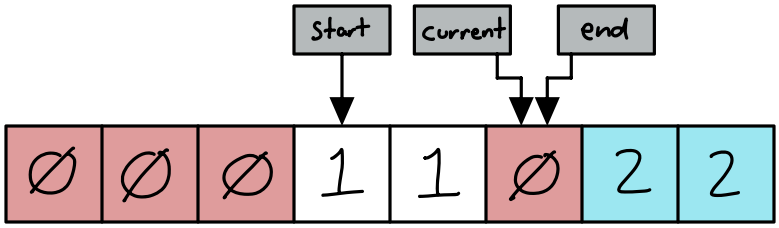
ix) colours[current] == 2

Swap the elements of colours[current] and colours[end] and decrement the end pointer.

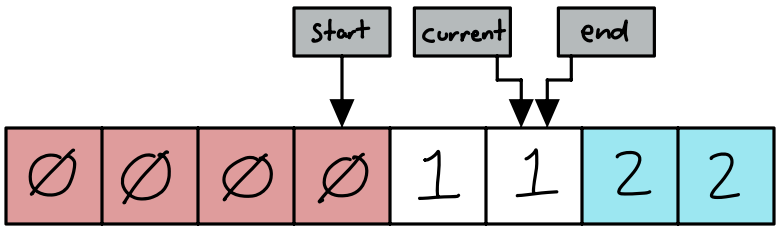


x) colours[current] == ∅

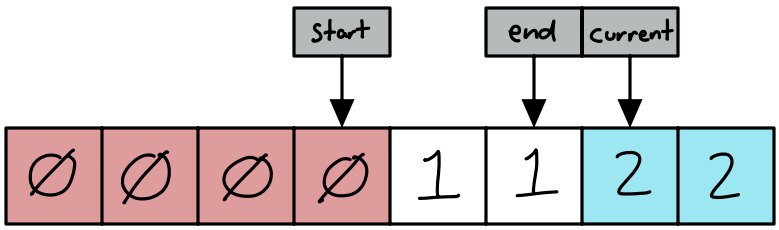
Swap colours[current] and colours[start]



xi) Increment both current and start pointers.



xii) The end pointer is less than the current pointer, no further swapping can be performed, return colours.



Code

```
vector<int> SortColors(vector<int>& colors) {
    int start = 0, current = 0, end = colors.size() - 1;

    while (current <= end) {
        if (colors[current] == 0) {
            swap(colors[start], colors[current]);
            current++;
            start++;
        } else if (colors[current] == 1) {
            current++;
        } else {
            swap(colors[current], colors[end]);
            end--;
        }
    }
    return colors;
}
```

Time Complexity

↳ Only traverse array once, thus O(n).

Space Complexity

↳ No extra space used, thus O(1).