Cycle Sort

Cycle Sort:

Key Terms

- Missing/Repeated/Duplicate Numbers
- Unsorted Array
- Permutation/Sequence
- In-place Sorting
- Unique Elements
- Indices/Positions
- Range of Numbers
- Fixed Range/Constant Range
- Modifying Indices/Positions
- Array Shuffling
- Swapping Elements
- Array Partitioning

- **Range of Numbers**: The problem involves an array containing a permutation of numbers from 1 to N, where N is the length of the array or a known upper bound. The numbers in the array are expected to be consecutive integers starting from 1.
- **Missing Numbers or Duplicates**: The problem requires finding missing numbers or duplicates within the given array. Typically, the array may have some missing numbers or duplicates, disrupting the sequence of consecutive integers.
- **No Negative Numbers or Zeroes**: The problem specifies that the array contains only positive integers, excluding zero and negative numbers. Cyclic sort works efficiently with positive integers and relies on the absence of zero or negative values.
- **Linear Time Complexity Requirement**: The problem constraints or requirements indicate the need for a solution with linear time complexity (O(N)), where N is the size of the array. Cyclic sort achieves linear time complexity as it involves iterating through the array once or a few times.
- **In-Place Sorting Requirement**: The problem requires sorting the array in-place without using additional data structures or consuming extra space. Cyclic sort operates by rearranging the elements within the given array, fulfilling the in-place sorting requirement.

Steps

The Cyclic Sort Pattern typically involves a WHILE loop and the following components:

1. **Initialization**: Start by initializing a pointer `i` to 0.

2. **Iterative Sorting**: Repeat the following steps until `i` reaches the end of the array:
    1. Look at the elements in the array to calculate the correct index, and see where the range lies. For example [0,5,4,2,1,3] means it is 0-N while [3,4,5,1,2] means its is 1-N.
        - 0 to N. the correct index for the number `x` at index `i` would be `x`
        - 1 to N, the correct index for the number `x` at index `i` would be `x - 1` because index starts at 0, not 1
    2. Check if the number at index `i` is already in its correct position. If it is, increment `i` to move to the next element.
    3. If the number at index `i` is not in its correct position, swap it with the number at its correct index.
    4. Repeat these steps until numbers are placed in their correct positions.
3. **Termination**: Once `i` reaches the end of the array, the array is sorted.
4. Once this sorted array is created, typically use another array to cycle through to ensure the index matches with the number

```
def cyclic_sort(nums):
    i = 0
    while i < len(nums):
        correct_index = nums[i] - 1  # Correct index for the associated number. Since nums is 1-N, then add -1 to prevent error
        if nums[i] != nums[correct_index]:  # If the number is not at its correct index.
            nums[i], nums[correct_index] = nums[correct_index], nums[i]  # Swap the numbers
        else:
            i += 1  # Evaluate the next number if it's already at the correct index
    return nums
# Example usage:
arr = [3, 1, 5, 4, 2]
print(cyclic_sort(arr))

# Ex. of swaps in the Loop
# [5,1,3,4,2]
# [2,1,3,4,5]
# Output: [1, 2, 3, 4, 5]
```
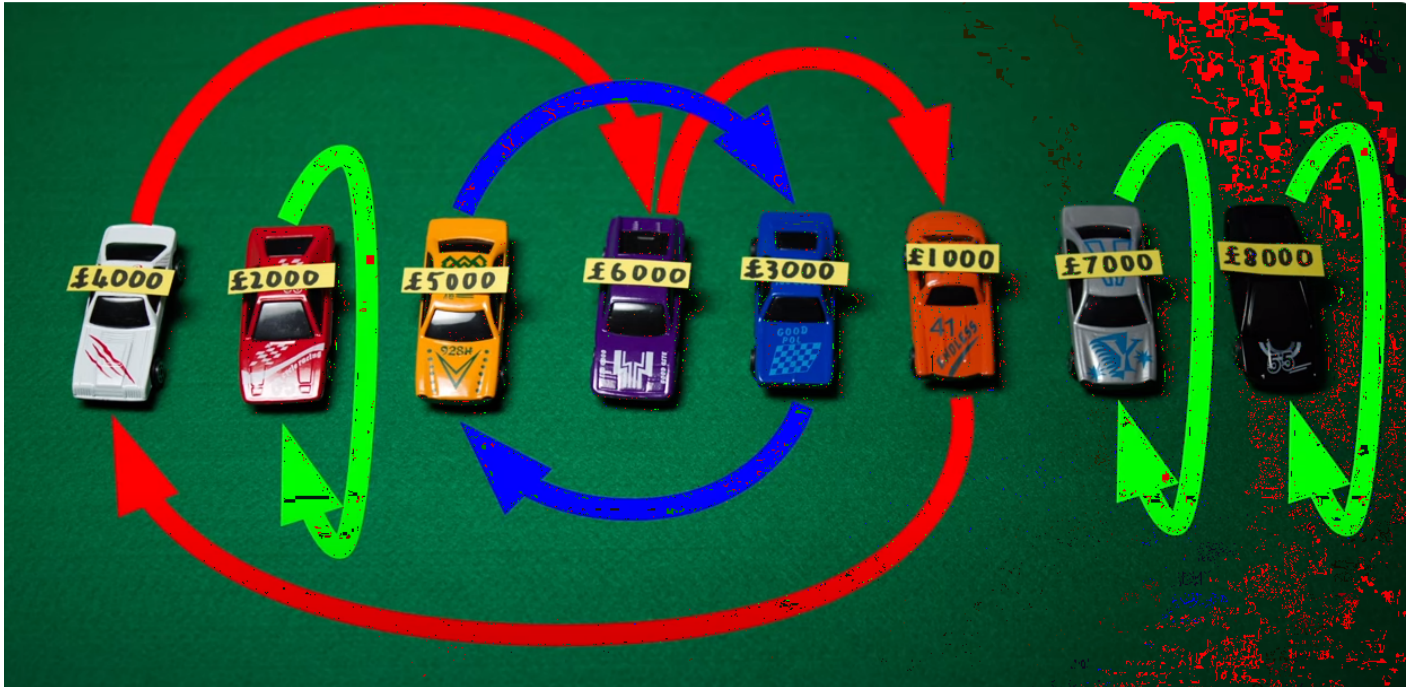
Visualization:

[**4**,2,5,**6**,3,1,7,8]
[**6**,2,5,4,3,**1**,7,8]
[**1**,2,5,4,3,**6**,7,8]
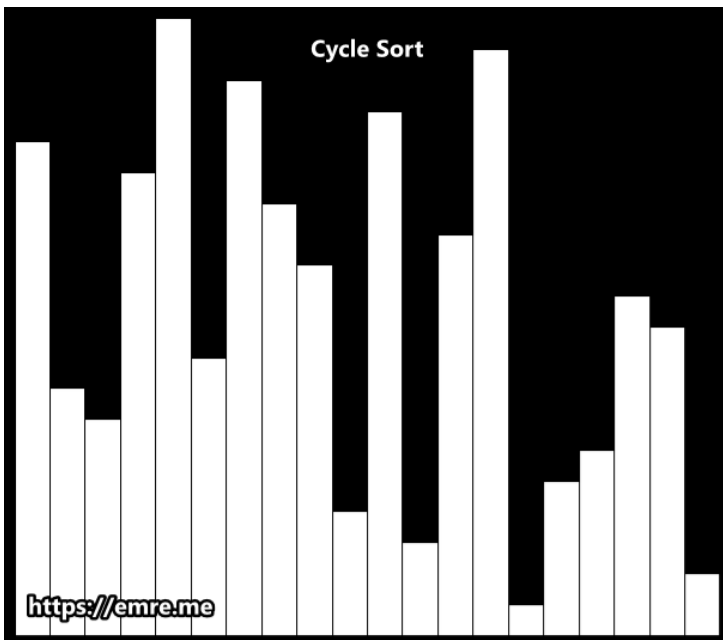next iteration [**1**,2,5,4,3,6,7,8]
next iteration [1,**2**,5,4,3,6,7,8]
[1,2,**3**,4,**5**,6,7,8]

next iteration [1,2,**3**,4,5,6,7,8]
next iteration [1,2,3,**4**,5,6,7,8]
next iteration [1,2,3,4,**5**,6,7,8]
next iteration [1,2,3,4,5,**6**,7,8]
next iteration [1,2,3,4,5,6,**7**,8]
next iteration [1,2,3,4,5,6,7,**8**]

https://youtu.be/jTN7vLqzigc?si=hPJ9mLPanY_d3RTp&t=37

Big O Complexity:

Time: $O(n^2)$ but analyze why this works better over time than other SORTING algorithms
Space: $O(1)$ since it is in-place

One of the advantages of cycle sort is that it has a low memory footprint, as it sorts the array in-place and does not require additional memory for temporary variables or buffers. However, it can be slow in certain situations, particularly when the input array has a large range of values. Nonetheless, cycle sort remains a useful sorting algorithm in certain contexts, such as when sorting small arrays with limited value ranges.

Cycle sort is an in-place sorting Algorithm, unstable sorting algorithm, and a comparison sort that is theoretically optimal in terms of the total number of writes to the original array.

- It is optimal in terms of the number of memory writes. It minimizes the number of memory writes to sort (Each value is either written zero times if it's already in its correct position or written one time to its correct position.)

Revision #16
Created 20 February 2024 05:36:47 by victor
Updated 26 February 2024 19:54:16 by victor