## Question: Write a program that takes an array A and an index i into A, and rearranges the elements such that

1.) All elements less than A[i] (the "pivot") appear first

2.) Followed by elements equal to the pivot

3.) Followed by elements greater than the pivot

Example

A = [0, 1, 2, 0, 2, 1, 1] (same array)

Pivot Index = 2

Valid Output: [0, 1, 0, 1, 1, 2, 2]

Valid Output: [0, 0, 1, 1, 1, 2, 2]  (a more perfect output)

This is called Dutch national flag partitioning because the Dutch national flag consists of three horizontal bands, each in a different color.

We will get an input and attempt to form a 3 section partitioning of the array and will always end up with these 3 sections of "lesser", "equal", and "greater" value UNLESS the value at our partition index is the smallest OR greatest value in the array.

Approach 2

Rather than create extra space, we can just do 2 passes, one forward, and then one backward.

During the forward pass for each element in the array, we will look for an element smaller than the element we are at. If it is smaller we would swap ourselves with that element.

During the backward pass move backward and look for an element larger than the element we are at. If it is larger we will swap ourselves with it.

ALSO, we stop moving backward when the element we stand at is less than the pivot because at the point we know that the partitioning is finished.

Complexities:

Time: O(n^2) + O(n^2) = O(n^2)

2 O(n^2) passes. For each element in the array, we do nearly the whole array of comparisons (triangular number) both for the forwards and backward scans.

Space: O(1)

Approach 3

But we don't need to do a new scan for a smaller or larger element FOR EACH element in the array.

We can just have a placement pointer where we swap in "smaller than pivot" values (for the forward iteration) and a "larger than pivot" placement pointer to swap in larger values.

The values equal to the pivot will end up lying in the middle by themselves.

Example

A = [0, 1, 2, 0, 2, 1, 1]

Pivot Index = 1

Value at pivot index is 1.

Forward pass: (looking to place items less than 1)

i = 0 [0, 1, 2, 0, 2, 1, 1] Swap index 0 with index 0

i = 1 [0, 1, 2, 0, 2, 1, 1] (no swap)

i = 2 [0, 1, 2, 0, 2, 1, 1] (no swap)

i = 3 [0, 0, 2, 1, 2, 1, 1] Swap index 1 with index 3

i = 4 [0, 0, 2, 1, 2, 1, 1] (no swap)

i = 5  [0, 0, 2, 1, 2, 1, 1] (no swap)

i = 6  [0, 0, 2, 1, 2, 1, 1] (no swap)

After backwards pass: (looking to place items greater than 1)

i = 6  [0, 0, 2, 1, 2, 1, 1] (no swap)

i = 5  [0, 0, 2, 1, 2, 1, 1] (no swap)

i = 4  [0, 0, 2, 1, 1, 1, 2] Swap index 4 and index 6

i = 3  [0, 0, 2, 1, 1, 1, 2] (no swap)

i = 2  [0, 0, 1, 1, 1, 2, 2] Swap index 2 and index 5

i = 1  [0, 0, 1, 1, 1, 2, 2] (stop*)

*0 is less than the pivot value of 1 so we are now in the region LESS THAN the pivot that we already created in the front of the array

Complexities:

Time: O(n) + O(n) = O(n)

Forward pass with a backwards pass, both running in linear time

Space: O(1)

No auxiliary spaces is created that scales with the input