1. Largest element in an array

→ set first element of array as the largest element (max_element)

→ scan through the array, if the current element is greater than max_element, then set max_element to current element.

→ Upon reaching the end of the array, we will obtain the largest element.

2. Second largest element in the array without sorting

→ Find max element of array

→ calculate differences with all elements & max element

→ create a new array of the differences

→ Find the index of largest non zero element in new array

→ This index corresponds to the second largest element in original array

Multiple passes & extra memory

→ if the current element is the largest then the previous element is the second largest.

→ After finding the largest element check if current element is greater than the second largest element.

→ Take no action if current element = largest element.

optimal solution

O(n)

↑ Just one pass of array

3. Check if array is sorted

Assume by sorted, we mean ascending order

Assume by sorted, we mean ascending order

1. For each element in the array check if the previous element is less than or equal. IF NOT array is not sorted.

4. Rotate Array by 1 place (left rotation)

1 2 3 4 5

↓

2 3 4 5 1

→ keep hold of first element
→ shift second to last element by 1 place   O(n)
→ copy first element to last position.

5. Rotate array by 'D' places (left)

0	1	2	3	4
1	2	3	4	5

Rotate by = D % size of array

D ↓

i =	0	1	2	3	4
e.g. Rotate by 3 elements	1	2	3	4	5

new index = i - D

Time: O(N-D)

Space: O(D)

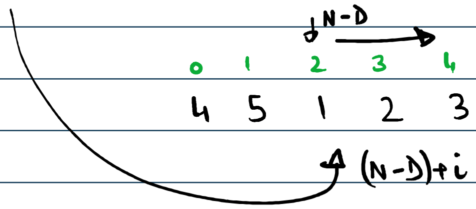
i = 0 1 2

temp[3] = {1,2,3}

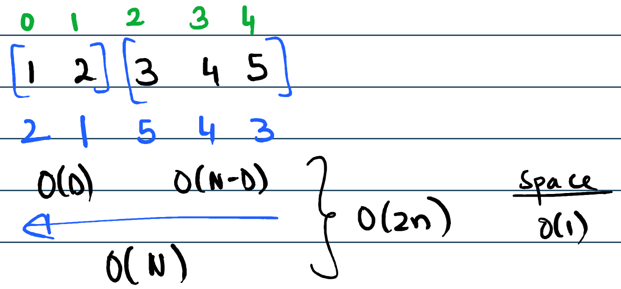
↓N-D →

0	1	2	3	4
4	5	1	2	3

(N-D)+i



e.g. Rotate By 2,



6. Move all zeros to the end of the array

e.g. 1 0 0 3 2 0 0 4 5 1

output: 1 3 2 4 5 1 0 0 0 0

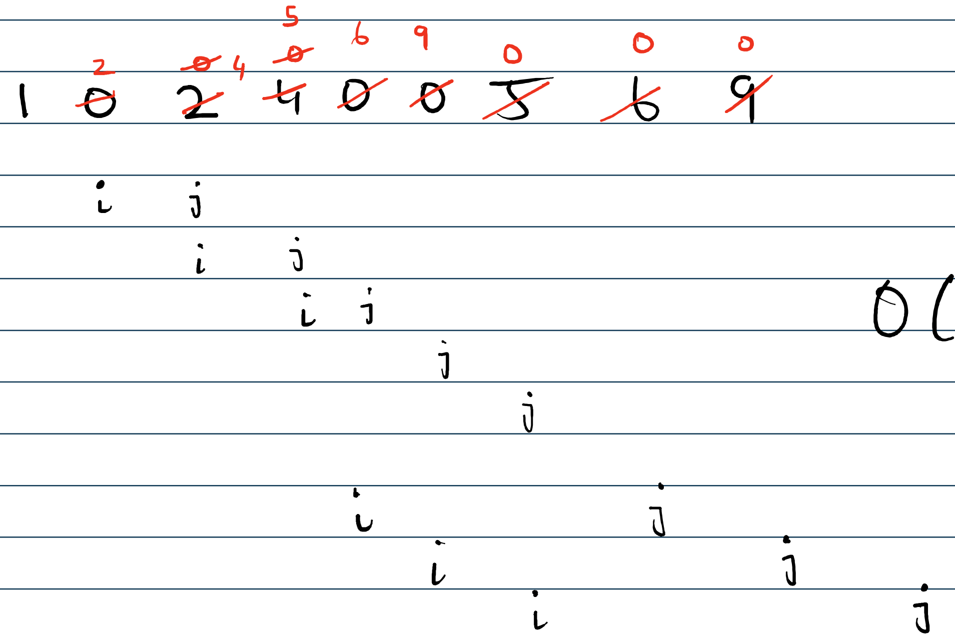
Approach 1:

1. start scanning from the end of the array

2. If we encounter a zero

↳ keep swapping with the next element as long as the next element is not a zero or we reached the end of array $O(n^2)$

Approach 2: Two pointers



O(n)