Doubt Class

Created	@July 3, 2022 10:08 AM
Class	
Type	
Materials	
Reviewed	☐

Problem

Given an array of integers, of length n , modify the array such that all the negative numbers are shifted to the last.

Example: `[-1,2,3,-5,6]` → `[2,3,6,-5,-1]`

Note: Try to solve without extra space and in the most efficient manner possible.

Solution

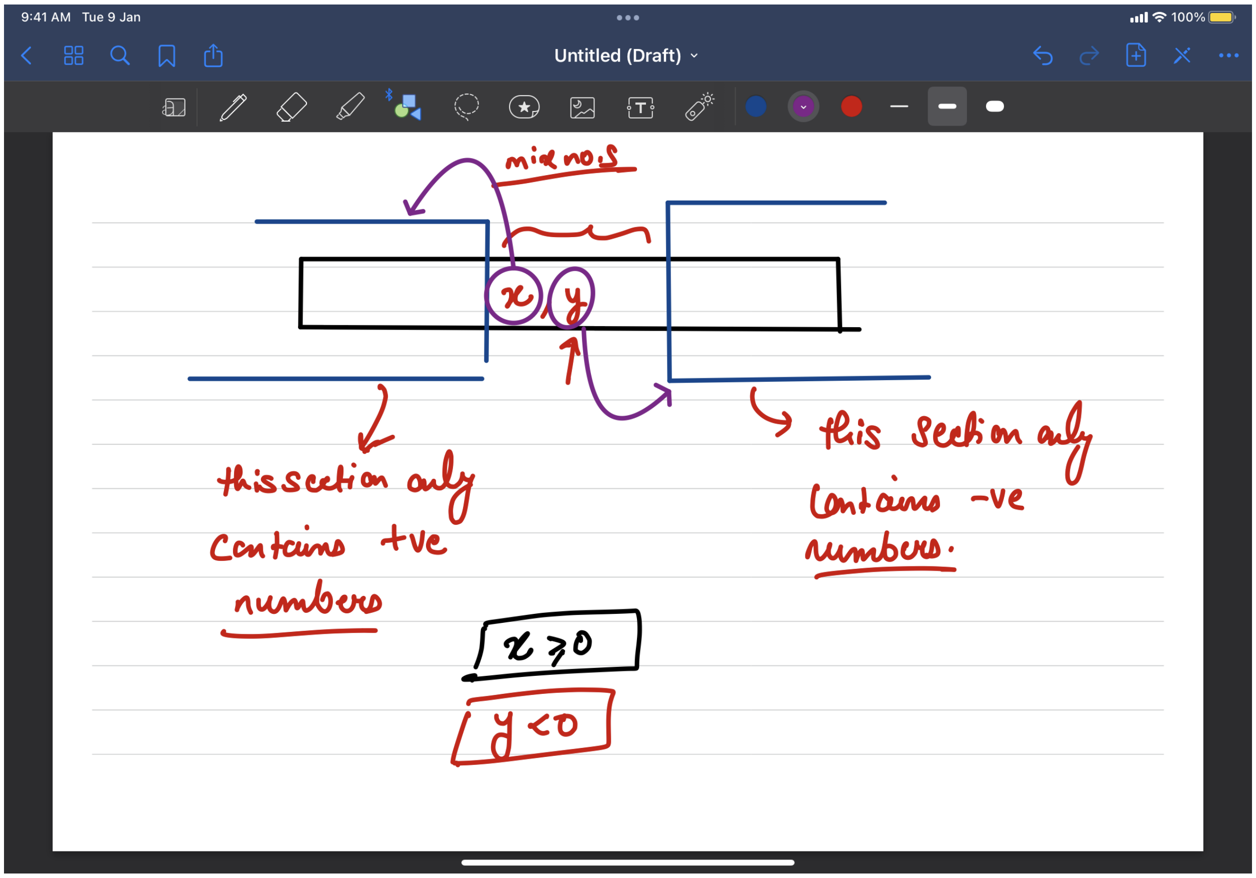
One possible solution is to sort the array in descending order. But there are a lot of concerns for a sorting based solution. If we use something like `merge sort` or `quick sort` then the time will `` `O(nlogn)` ``

But they will take extra space, merge sort will take O(n) space and quick sort will take O(logn) space.

There are other sorting algorithms as well like `insertion` `bubble` and `selection` but there time complexity is very bad `O(n^2)`

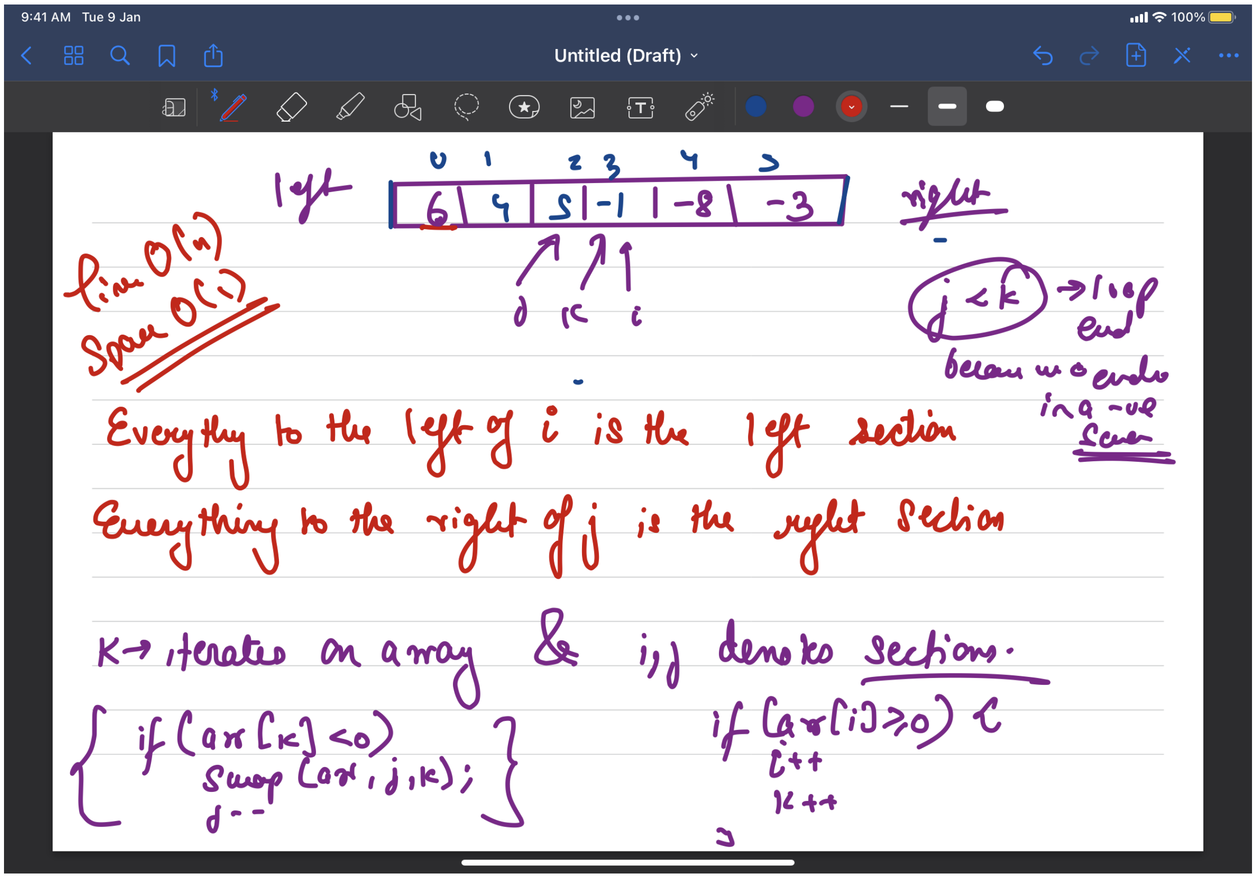
```
function shiftToEnd(arr) {
  arr.sort((a, b) => b-a);
  return arr;
}
```

How can we optimise ?


9:41 AM Tue 9 Jan
100%
Untitled (Draft)
mix nos
x
y
this section only contains +ve numbers
this section only contains -ve numbers.
x ≥ 0
y < 0

Let’s say as per the above image, you somehow got an array such that in the section on the left you’ve only positive numbers and on the section on the right you have only negative. In the middle you’ve mixed numbers. So now for the mixed we want to segregate them in one of the sections, so if an element `x >= 0` we will shift it to the left section, and if there is an element `y<0` we will shift it to the right section.

So if we continue this process, at the last all the negative elements can go to the last.



left

0	1	2	3	4	5
6	4	5	-1	-8	-3

right

j k i

Time O(n)
Space O(1)

$j < k \rightarrow$ loop end
because we ended in a -ve Zone

Everything to the left of i is the left section

Everything to the right of j is the right Section

K→ iterates on array & i,j denotes sections.

```
{ if (arr[k] <0)
    swap (arr, j, k);
    j--
}
```

```
if (arr[i] >= 0) {
    i++
    k++
}
```

So in order to implement the shifting algo we discussed, we can maintain 3 variables. `i,j,k`

`i` denotes the boundary of the left/positive section such that everything to the left of `i` is positive.

`j` denotes the boundary of the right/negative section such that everything to the left of `j` is negative.

`k` is the variable with which we iterate on every element.

Then we will iterate using `k` one by one, if we encounter a negative element we swap it with the element at `j` and reduce. But `k` stays there because due to the swapping on the index `k` we got a new element so we have to judge it again.

If the element is positive we just increment both `i` and `k` because this belongs to the left of `i` and as no swapping was done so new element was there on the old index so we moved `k`

We are touching every element once by k at max, that’s why Time complexity is `O(n)` and space is `O(1)` as no extra space was taken.

`NOTE` these type of algorithms which involves 2-3 pointers denoting boundaries are called as two pointer algorithms.

```
function swap(arr, a, b) {
    let temp = arr[a];
    arr[a] = arr[b];
    arr[b] = temp;
}
function shiftToLast(arr) {
    /**
     * Time: O(n)
     * Space: O(1)
     */
    let i = 0;
    let k = 0;
    let j = arr.length - 1;
    while(k <= j) {
        if(arr[k] < 0) {
            // if element is negative we swap it with the element at j, and negative secti
on increases
            swap(arr, k, j);
            j--;
        } else {
            // if element is positive we just move ahead with i, and positive section incr
eases
            i++;
            k++;
        }
    }
}
const arr = [8];
shiftToLast(arr);
console.log(arr)
```