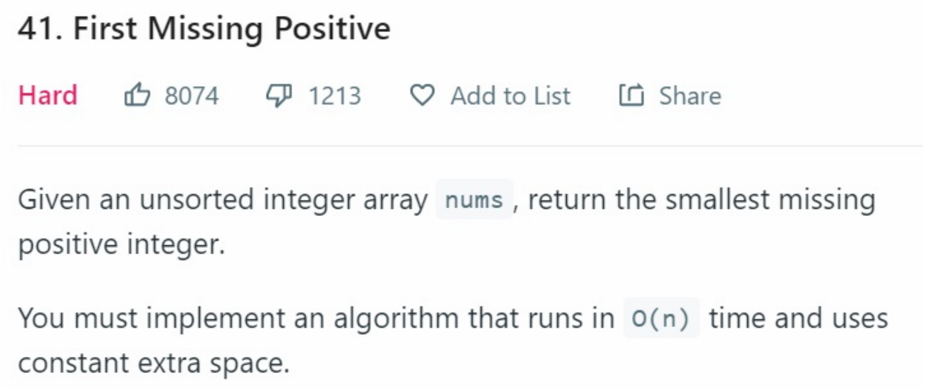


41. First Missing Positive

Given an unsorted integer array `nums`, return the smallest missing positive integer.

You must implement an algorithm that runs in `O(n)` time and uses constant extra space.

smallest positive no. -> 1

time : O(n)

space : O(1)

① segregate +ve, -ve

② marking in +ve area

③ travel and find first index in +ve area, having a +ve value

10	-4	3	1	6	-2
0	1	2	3	4	5

-10	3	-1	6	-4	-2
0	1	2	3	4	5
↓	↓	↓	↓		
1	2	3	4		

n length

> 1 to n+1

① segregate +ve, -ve

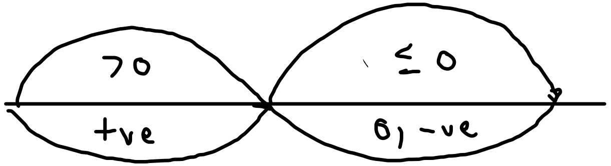
② marking in +ve area

③ travel and find first index in +ve area, having a +ve value

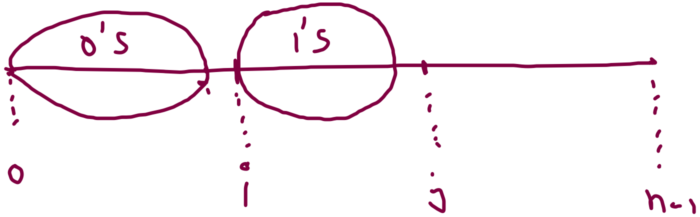
10	-8	5	2	13	4	-6	3	7	1
0	1	2	3	4	5	6	7	8	9

-10	-5	-2	-13	-4	3	-7	1	-8	-6
0	1	2	3	4	5	6	7	8	9
↓	↓	↓	↓	↓	↓	↓	↓		
1	2	3	4	5	6	7	8		

↳ is absent



0 1 0 0 1 0

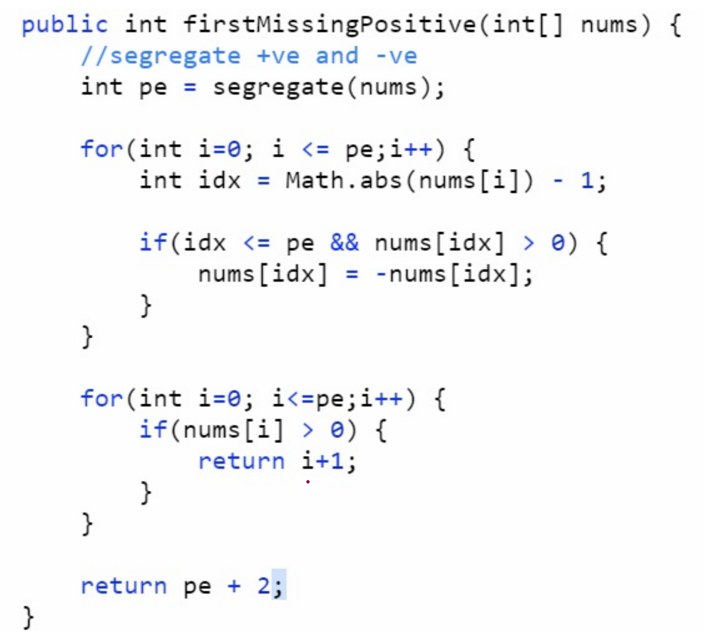


0 to i-1 -> 0's

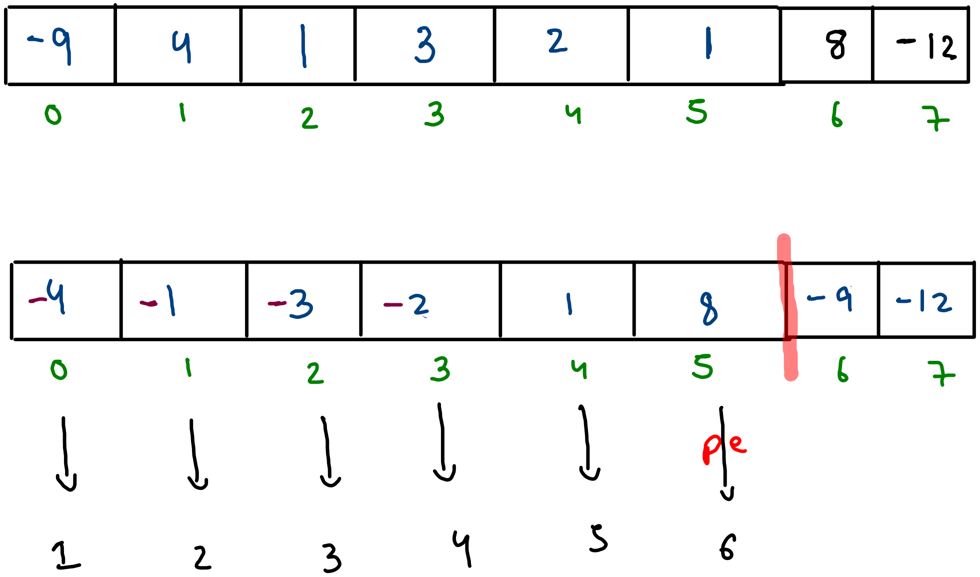
i to j-1 -> 1's

j to n-1 -> uk

```
if(arr[j] == 1) {
    j++;
}

else {
    swap(arr, i, j)
    j++; i++;
}
```

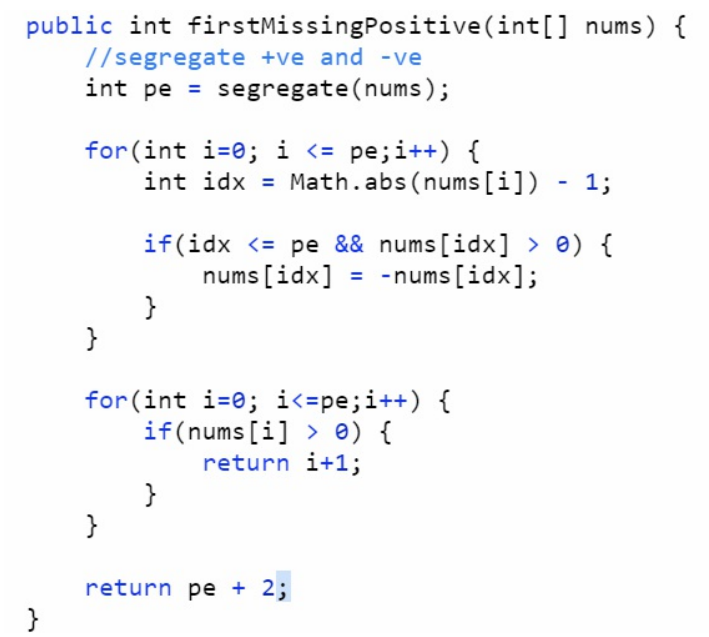


```
public int firstMissingPositive(int[] nums) {
    //segregate +ve and -ve
    int pe = segregate(nums);

    for(int i=0; i <= pe;i++) {
        int idx = Math.abs(nums[i]) - 1;

        if(idx <= pe && nums[idx] > 0) {
            nums[idx] = -nums[idx];
        }
    }

    for(int i=0; i<=pe;i++) {
        if(nums[i] > 0) {
            return i+1;
        }
    }

    return pe + 2;
}
```

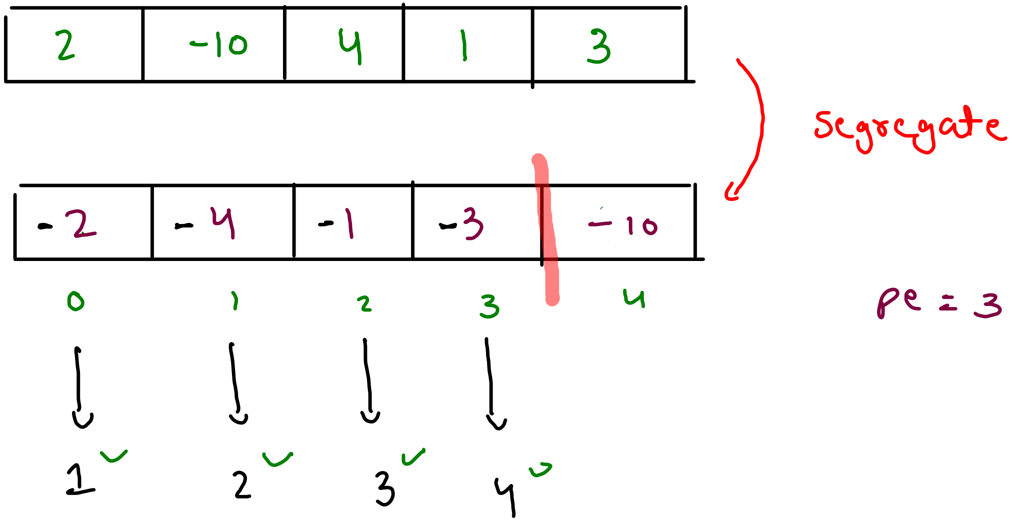


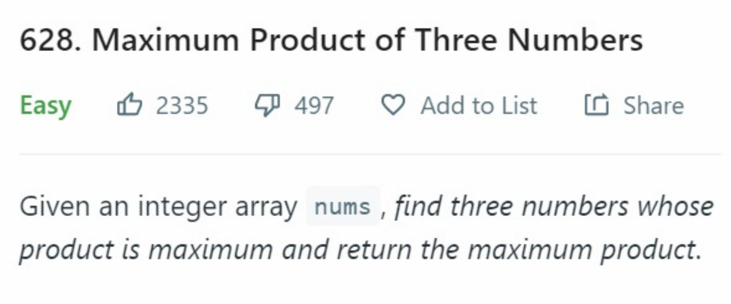
-9	4	1	3	2	1	8	-12
0	1	2	3	4	5	6	7

-4	-1	-3	-2	1	8	-9	-12
0	1	2	3	4	5	6	7



```java
public int firstMissingPositive(int[] nums) {
    //segregate +ve and -ve
    int pe = segregate(nums);

    for(int i=0; i <= pe;i++) {
        int idx = Math.abs(nums[i]) - 1;

        if(idx <= pe && nums[idx] > 0) {
            nums[idx] = -nums[idx];
        }
    }

    for(int i=0; i<=pe;i++) {
        if(nums[i] > 0) {
            return i+1;
        }
    }

    return pe + 2;
}
```

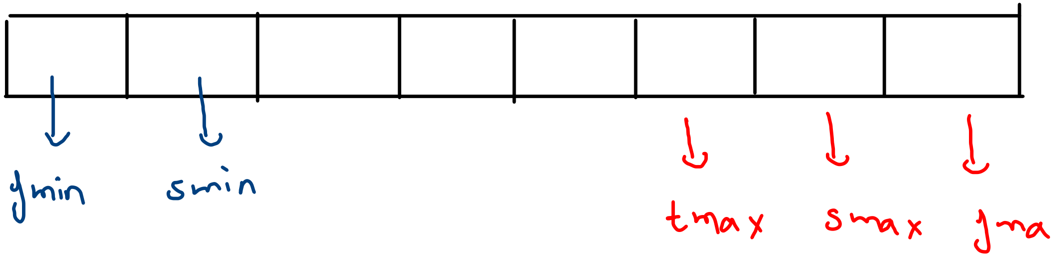




628. Maximum Product of Three Numbers

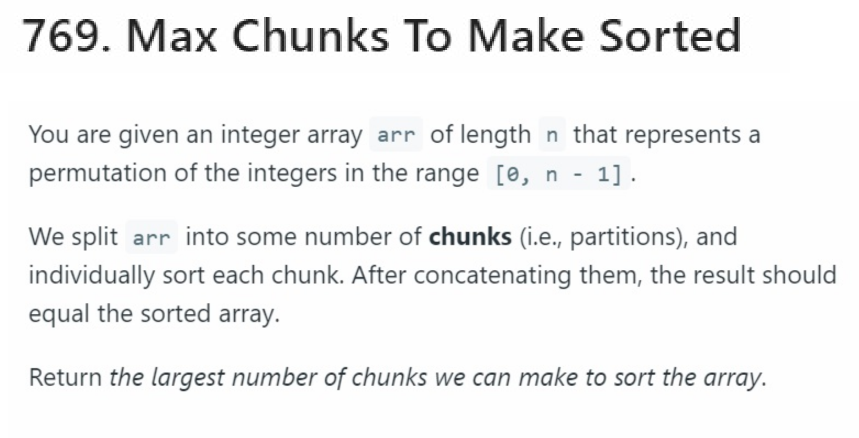
Given an integer array `nums`, *find three numbers whose product is maximum and return the maximum product.*

imagine sorted array



$f1 = fmax * smax * tmax$

$f2 = fmin * smin * fmax$



769. Max Chunks To Make Sorted

You are given an integer array `arr` of length `n` that represents a permutation of the integers in the range `[0, n - 1]`.

We split `arr` into some number of **chunks** (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array.

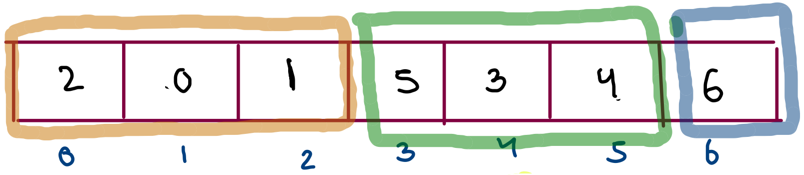
Return *the largest number of chunks we can make to sort the array.*

n = 7

values -> 0 to 6

T : O(n)

S : O(1)



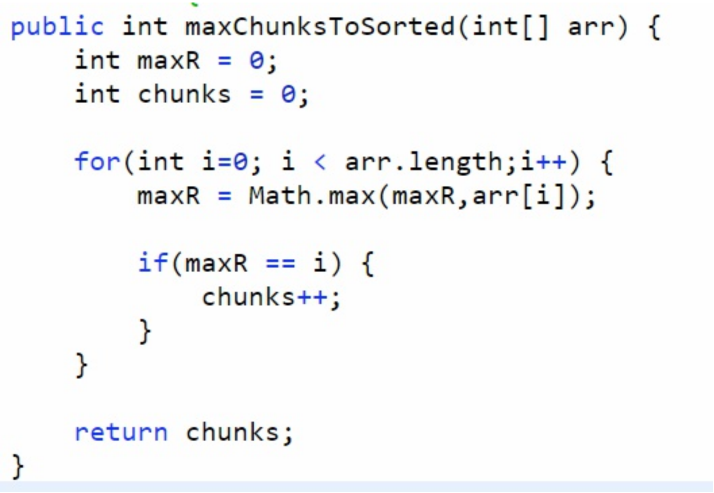
i

range : 1 to n

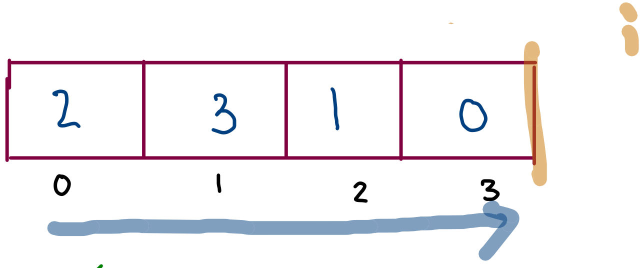
max R ~~2~~ ~~5~~ 6

chunks = ~~0~~ ~~1~~ ~~2~~

3



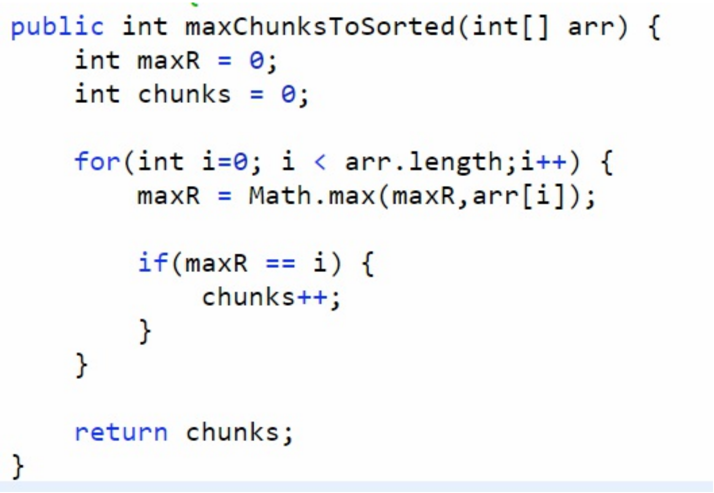
```java
public int maxChunksToSorted(int[] arr) {
    int maxR = 0;
    int chunks = 0;

    for(int i=0; i < arr.length;i++) {
        maxR = Math.max(maxR,arr[i]);

        if(maxR == i) {
            chunks++;
        }
    }

    return chunks;
}
```



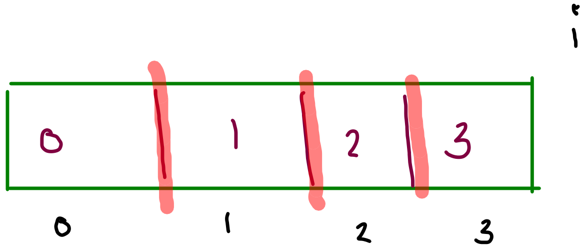
2	3	1	0
0	1	2	3

maxR : ~~2~~
3

chunks = ~~0~~
1



```
public int maxChunksToSorted(int[] arr) {
    int maxR = 0;
    int chunks = 0;

    for(int i=0; i < arr.length;i++) {
        maxR = Math.max(maxR,arr[i]);

        if(maxR == i) {
            chunks++;
        }
    }

    return chunks;
}
```



maxR = ~~0~~

~~1~~

~~2~~

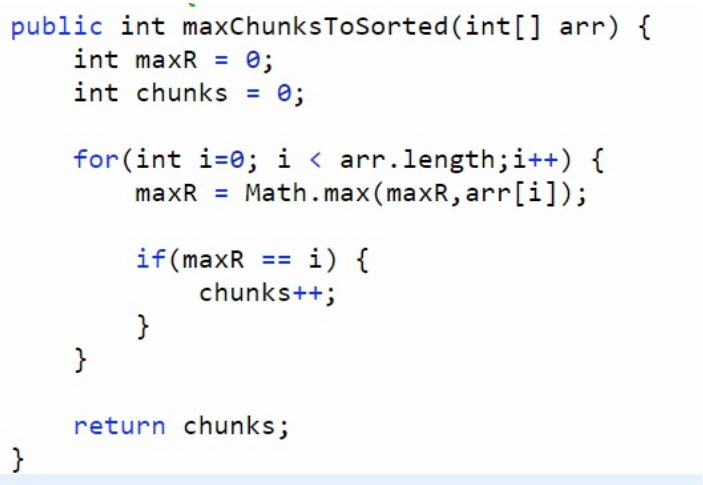
3

chunks = ~~0~~

~~1~~

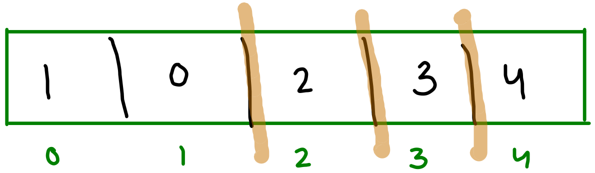
~~2~~

~~3~~ 4



```
public int maxChunksToSorted(int[] arr) {
    int maxR = 0;
    int chunks = 0;

    for(int i=0; i < arr.length;i++) {
        maxR = Math.max(maxR,arr[i]);

        if(maxR == i) {
            chunks++;
        }
    }

    return chunks;
}
```

i



maxR ~~1~~ ~~2~~ ~~3~~ 4

chunks = ~~0~~ ~~1~~ ~~2~~ ~~3~~ 4