Given an unsorted array, find	the smallest missing	positive integer			
put: [3,4,-1,1] input:[1	2,07 inp	ut: [7,8,9,11,12]			
tput: 2 output: 3					
Smallest positive integer the smallest	positive integer is 1	-oThis also tells.	is that I is our lower bound		
보기 있게 보고 나타고 있다. (하는 중요) 공기 (하는 )				why? B/c if we have an input like [	[1,2,3,4] the next smallest missing in
				3	is 5 len(array) +1
ef first MissingPositive (array):					
if 1 not in numes:	base case				
return 1					
for index, value in enumerate (array):	Here we identi	for all the ne	gative numbers and zeros	5	
if value L= D:	and replace the				
array[index]=len(array)+1					
for element in array:	Here we place o	nur Markers, identifyi	ng all relevant elements in	the input array	
num = abs (num)	Treating the inp	ut array as an a	rray of indicies and make elemen	nts present at the given index neg	afive
if num <= lenlamay):				nd negative numbers were replaced	이 맛있다. 보이라는 아이를 보이기 때문이 없다.
amou[n-1]= -1 * abs(amou[n-1])				ins a I ) this is the	
	fte	third element (4) negat	ive b/c the input array conten	as a 3 element in the	to to every
	the	fourth element (6) negati	e b/c the input array contain	ins a 4 evenent in the	e array
for index in rangellenlamy):	Return the first i	ndex that has a posi	Hive element		
if array[i] > 0:		evious example: [-3			
return i+1		a positive element is 1			
	The negotive number				
* remember armys are 0 based					
so we need to adjust and "+1"					
When dealing with arrays					
Brute force Solution					
def first Missing Integer (nums)					
for index in range (1,1	en (nums)+2):				
if index not in no	ums:				
return index					
This solution runs in Oln'	-1) time				
Checks the array at every i					

for index, value in enumerate (nums): $3 <= 0$ $2 <= 0$ $1 <= 0$ $-1 <= 0$ $6 <= 0$ if value $<= 0$ : $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1 <= 0$ $1                                    $	Walk through				
for index value in enumate (nums): 3 <= 0 2 <= 0	input array: [3,2,1,-1,6]				
for index value in enumate (nums): 3 <= 0 2 <= 0	2st iteration				
######################################					
					1 <= 0 -1 2= 0 6 2= 0
len(mon)   = 6		#F	##		
Num	Urme Finger 7 = 150 (um2) + (				
Num= abs(num)  3 < 6	Nums = [3,2,1,6,6]				
if nam <= len(nums):  nums[i] = -1 * abs(nums[num-i])  nums[i] = -1 * abs(nums[i])  nums[i] = -1 * abs(nums[i])  2 = 6 -> nums[i] = -1 * abs(nums[i])  nums[i] = -1 * abs(nums[i])  2 = 6 -> nums[i] = -1 * abs(nums[i])  1 = -1 * abs(num	for num in nums:				
if nam <= len(nums):  nums[i] = -1 * abs(nums[num-i])  nums[i] = -1 * abs(nums[i])  nums[i] = -1 * abs(nums[i])  2 = 6 -> nums[i] = -1 * abs(nums[i])  nums[i] = -1 * abs(nums[i])  2 = 6 -> nums[i] = -1 * abs(nums[i])  1 = -1 * abs(num	01100 - 01-0				
#True nums[2] = -1 * abs(nums[2])  nums[1] = -( * abs(nums[num-1])  2 6 -> nums[2] = -1 * ( abs(nums[2])  2 6 -> nums[2] = -1 * ( abs(nums[2])  2 6 -> nums[1] = -1 * abs(nums[2])  nums[1] = -1 * abs(nums[2])  nums[1] = -2  1 6 6 -> nums[1] = -1 * abs(nums[1])  #true nums[0] = -1 * abs(nums[1])  1 * true nums[0] = -1 * abs(nums[2])  1 * true nums[0] = -3  1 * ( 6 -> nums[6]) = -3  1 * ( 6 -> nums[6]) = -3  1 * ( 7 -> nums[6]) = -1 * abs(nums[6])  1 * true nums[5] = -1 * abs(nums[6])  1 * true nums[5] = -1 * ( abs(nums[6]))  1 * true nums[5] = -1 * ( abs(nums[6]))  1 * true nums[5] = -1 * ( abs(nums[6]))  1 * true nums[5] = -1 * ( abs(nums[6]))  1 * true nums[5] = -1 * ( abs(nums[6]))  1 * true nums[5] = -1 * ( abs(nums[6]))  1 * true nums[5] = -1 * ( abs(nums[6]))  1 * true nums[5] = -1 * ( abs(nums[6]))  1 * true nums[6] = -1 * ( abs(nums[6])  1 * true nums[6] = -1 * ( abs(nums[6])  1 * true nums[6] = -1 * ( abs(nums[6])  1 * true nums[6] = -1 * ( abs(nums[6])  1 * true nums[6] = -1 * ( abs(nums[6])  1 * true nums[6] = -1 * ( abs(nums[6])  1 * true nums[6] = -1 * ( abs(nums[6])  1 * tru	11WW- abs(Nwy)		241	-b	nums[3-17=-1 * abs(num:[3-17)
nums[1] = -[ * abs(nums[num-1])  2 \( \) nums[2] = -1 * 1 \\  2 \( \) \text{nums[2] = -1 * 1 \\  2 \( \) \text{nums[2] = -1 * abs(nums[2]) \\  nums[1] = -1 * 2 \\  nums[1] = -1 * 2 \\  nums[1] = -1 * abs(nums[1]) \\  nums[1] = -2 \\  1 \( \) \text{nums[1] = -1 * abs(nums[1]) \\  nums[0] = -1 * abs(nums[1]) \\  nums[0] = -3 \\  nums[0] = -3 \\  \text{nums[0] = -3 * abs(nums[6]) \\  \text{nums[0] = -1 * abs(nums[6]) \\  nums[0] = -3 \\  nums[0] = -1 * abs(nums[6]) \\  nums[0] = -3 \\  nums[0] = -1 * abs(nums[6]) \\  nums[0] = -3 \\  nums[0] = -6 \\  nums[1] > 0 \\  \text{if nums[3] > 0	if nam <= len(nums):				
2 = 6 -> nums[2:] = -1 * abs(nums[2:])  #True nums[1] = -1 * abs(nums[1])  nums[1] = -2    1		um-1])			[조건의] [조건의 [조건의] 1.1 [조건의 [조건의 [조건의 [조건의 [조건의 [조건의 [조건의 [조건의
#True nums[1] = -1 * abs(nums[1])  nums[1] = -1 * 2  nums[1] = -2  146 -> nums[1] = -1 * abs(nums[1])  #true nums[0] = -1 * abs(nums[0])  nums[0] = -1 * 3  nums[0] = -3  646 -> nums[6-1] = -1 * abs(nums[6-1])  #True nums[5] = -1 * abs(nums[6-1])  nums[5] = -1 * abs(nums[6-1])  nums[5] = -1 * 6  nums[5] = -6  nums[5] = -6  nums[5] > 0:  #F  return i + 1  if nums[2] > 0:  #f  return 3 + 1					nums[2] = -[
#True nums[1] = -1 * abs(nums[1])  nums[1] = -1 * 2  nums[1] = -2  146 -> nums[1] = -1 * abs(nums[1])  #true nums[0] = -1 * abs(nums[0])  nums[0] = -1 * 3  nums[0] = -3  646 -> nums[6-1] = -1 * abs(nums[6-1])  #True nums[5] = -1 * abs(nums[6-1])  nums[5] = -1 * abs(nums[6-1])  nums[5] = -1 * 6  nums[5] = -6  nums[5] = -6  nums[5] > 0:  #F  return i + 1  if nums[2] > 0:  #f  return 3 + 1			211	_	Dung [2-1] = -1 * cloch Dung [2-1]
nums[i] = -1 * 2  nums[i] = -2  146 -> nums[i] = -1 * abs(nums[i])  #true nums[o] = -1 * abs(nums[o])  nums[o] = -1 * 3  nums[o] = -3  646 -> nums[i] = -1 * abs(nums[i])  #true nums[i] = -1 * abs(nums[i])  nums[i] = -1					
					Larger   Frank   No. 34   No. 35   No
# true nums [0] = -1 * abs (nums [0])  nums [0] = -3  nums [0] = -3  nums [0] = -1 * abs (nums [6-1])  # True nums [5] = -1 * abs (nums [6-1])  nums [5] = -1 * abs (nums [5])  nums [5] = -1 * abs (nums [6-1])  nums [5] = -1 * abs (nums [6-1])  nums [5] = -1 * abs (nums [6-1])  nums [6] = -1 * abs (nums [6-1])  nums [5] = -1 * abs (nums [6-1])  nums [6] = -1 * abs (nums [6])  nums [6] = -1 * abs (nums [6])  nums [6] = -1 * abs (nums [6])  nums [6] = -1					nums[1] = -2
# true nums [0] = -1 * abs (nums [0])  nums [0] = -3  nums [0] = -3  nums [0] = -1 * abs (nums [6-1])  # True nums [5] = -1 * abs (nums [6-1])  nums [5] = -1 * abs (nums [5])  nums [5] = -1 * abs (nums [6-1])  nums [5] = -1 * abs (nums [6-1])  nums [5] = -1 * abs (nums [6-1])  nums [6] = -1 * abs (nums [6-1])  nums [5] = -1 * abs (nums [6-1])  nums [6] = -1 * abs (nums [6])  nums [6] = -1 * abs (nums [6])  nums [6] = -1 * abs (nums [6])  nums [6] = -1			176	<b>-</b>	2 51-17 = -1 x Glas (25,100 F)=17)
$\begin{array}{cccccccccccccccccccccccccccccccccccc$			# 120		noms [0] = -1 * abs (nums [0])
6 = 6 -> nums [6-1] = -1 * abs (nums [6-1])  # True nums [5] = -1 * abs (nums [5])  nums [5] = -1 * 6  nums [5] = -1 * 6  nums [5] = -6  nums [5] = -6  nums [5] > 0:  if nums [1] > 0:  # F  return i + 1  if nums [2] > 0:  # F  return 3 + 1					nums [0] = -1 * 3
The number of th					nums [0] = -3
The number of th			121	7	nums [6-17 =- ) * abel nums[6-17)
Nums = $[-3, -2, -1, 6, -6]$ Nums = $[-3, -2, -1, 6, -6]$ For i in range (len (nums)): if nums[0] > 0: if nums[1] > 0: -2 > 0  if nums[i] > 0: #F  return i + 1 if nums[2] > 0: if nums[3] > 0: -1 > 0  ##################################			H True		1/WMX 1 5 1 - 1 / CUS 1 NUMS 1 5 ( )
Nums = [-3,-2,-1,6,-6]  For i in range (len (nums)): if nums[0] > 0:  if nums[i] > 0:  #F  return i + 1  if nums[2] > 0:  if nums[3] > 0:  #F  return 3+1			100		nums[5] =-1 * 6
For i in range (len (nums)): if nums[o] >0: if nums[i] >0:  -3 >0  #F  return i + 1  if nums[2] >0:  #f  return 3 +1					nums[5] =-6
if nums[i] > 0: #F  return i + 1  if nums[2] > 0: if nums[3] > 0:  -(>0  #F  #F  return 3+1	Nums = [-3,-2,-1,6,-6]				
if nums[i] > 0: #F  return i + 1  if nums[2] > 0: if nums[3] > 0:  -(>0  #F  #F  return 3+1					
if nums[i] > 0: #F  return i + 1  if nums[2] > 0: if nums[3] > 0:  -(>0  #F  #F  return 3+1	for i in range (len/nums):	if	numstol	>D:	if nums[1] > 0:
return i + 1  if nums[2]>0:  if nums[3]>0:  6>0  # T  return 3+1					
-(>0 #F # T return 3+1	it nums[i] > 0:		₩F		# # # # # # # # # # # # # # # # # # # #
-(>0 #F # T return 3+1	501, 50 ; + 1	:C .	0	0:	[[ n [27] \ n .
#F #T return 3+1	(6101) 1 1				
return 3+1					# 7
I return 4					return 3+1
					return 2