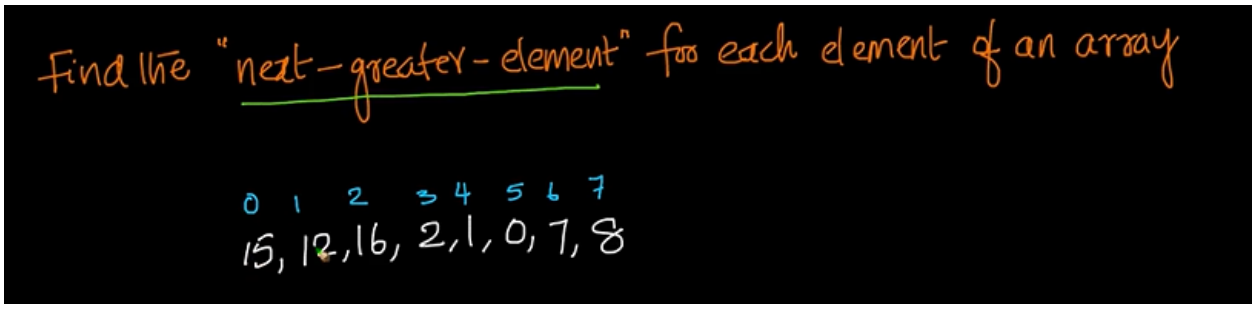
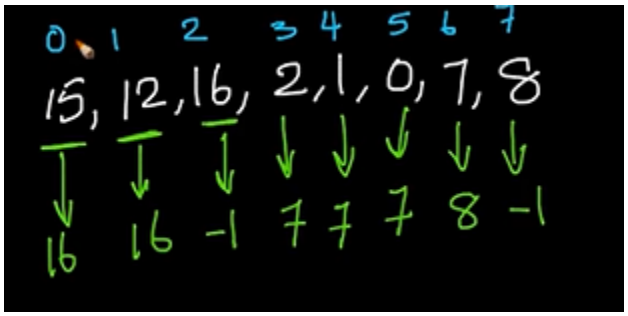
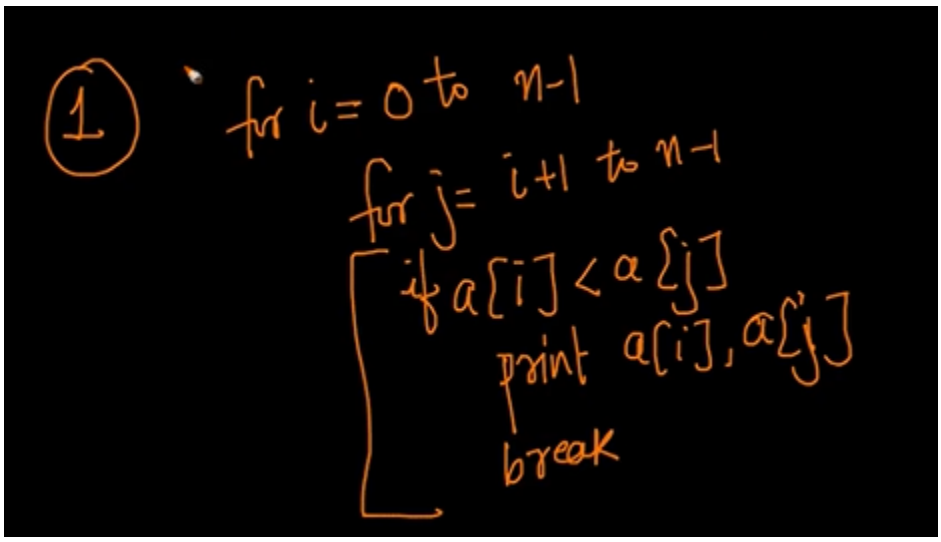
Next Greater Element



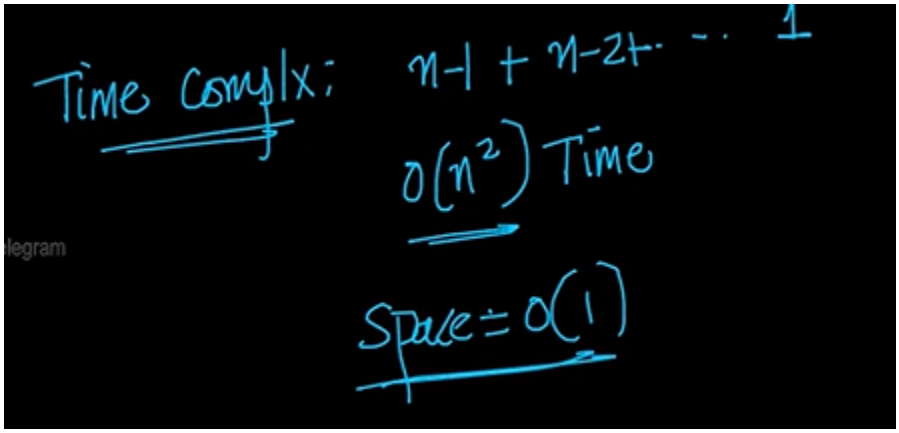
1. Here we have found the next greater element for each element



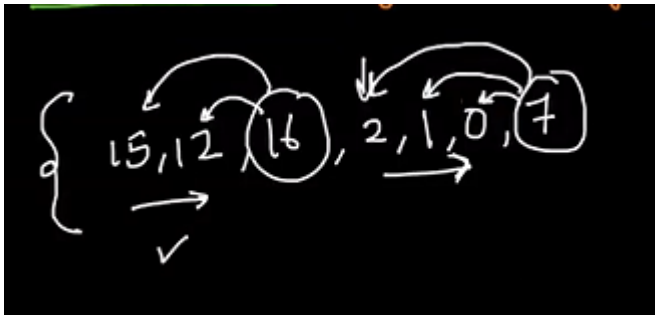
2. In brute force approach ..we follow this

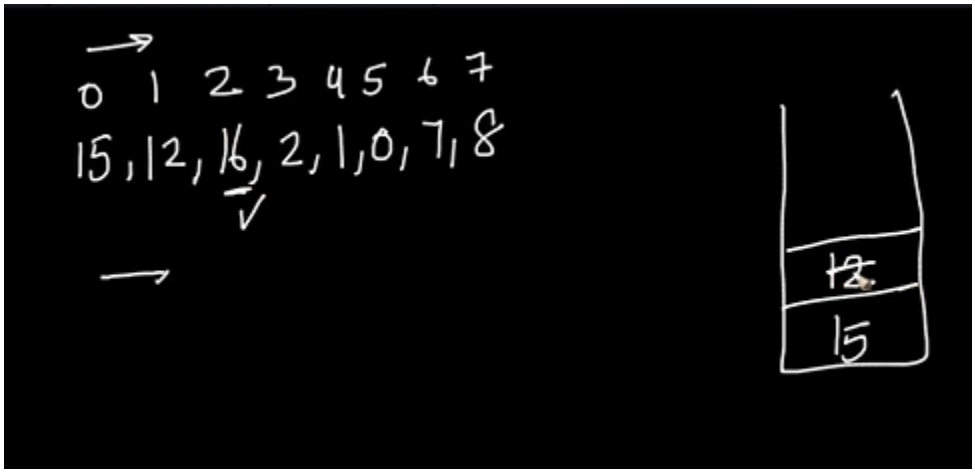


3. This would solve the problem..but the main issue is time complexity



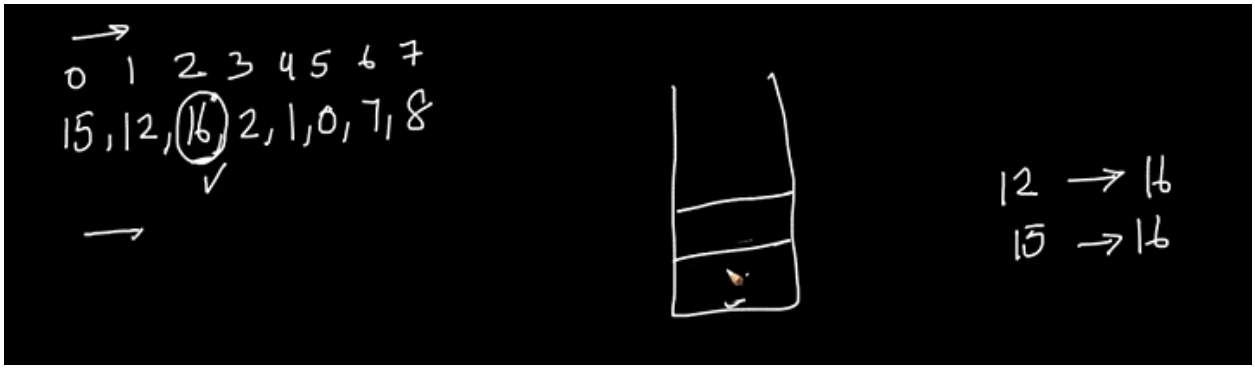
Can we solve this better?

1. Here if we observe  ..16 ..will replace 15,12…and similarly 7 will replace 2,1,0…we'll use this observation and solve it in O(n)
2. Lets take a stack and insert elements ..until a next greater element is found

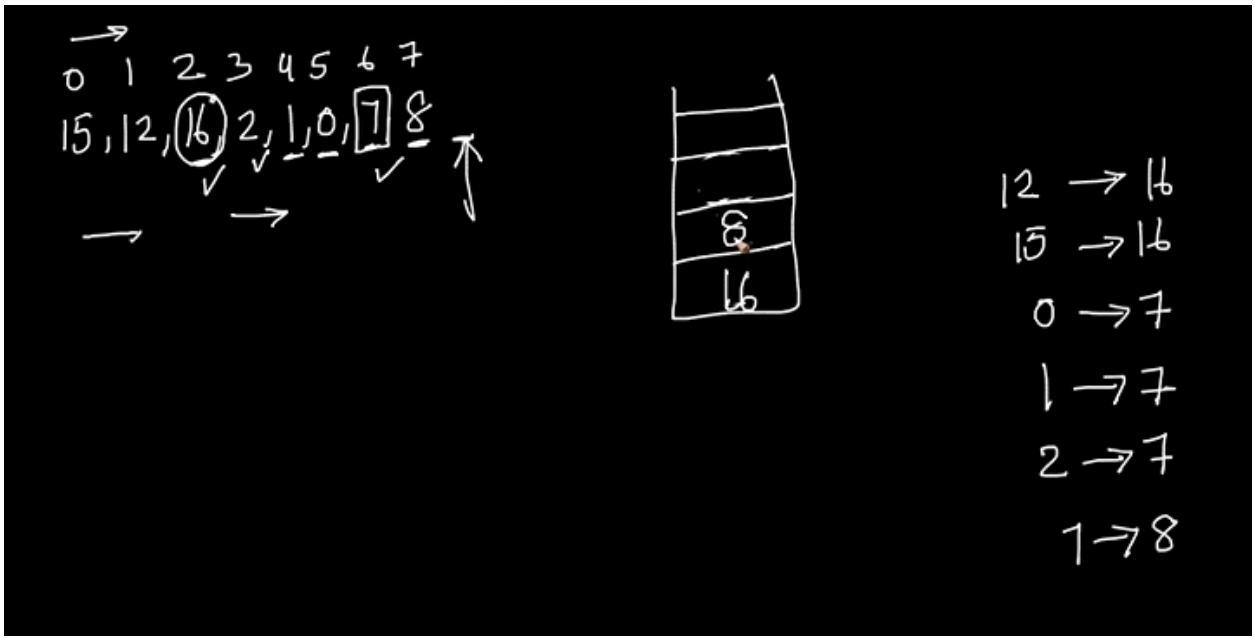


..so here we

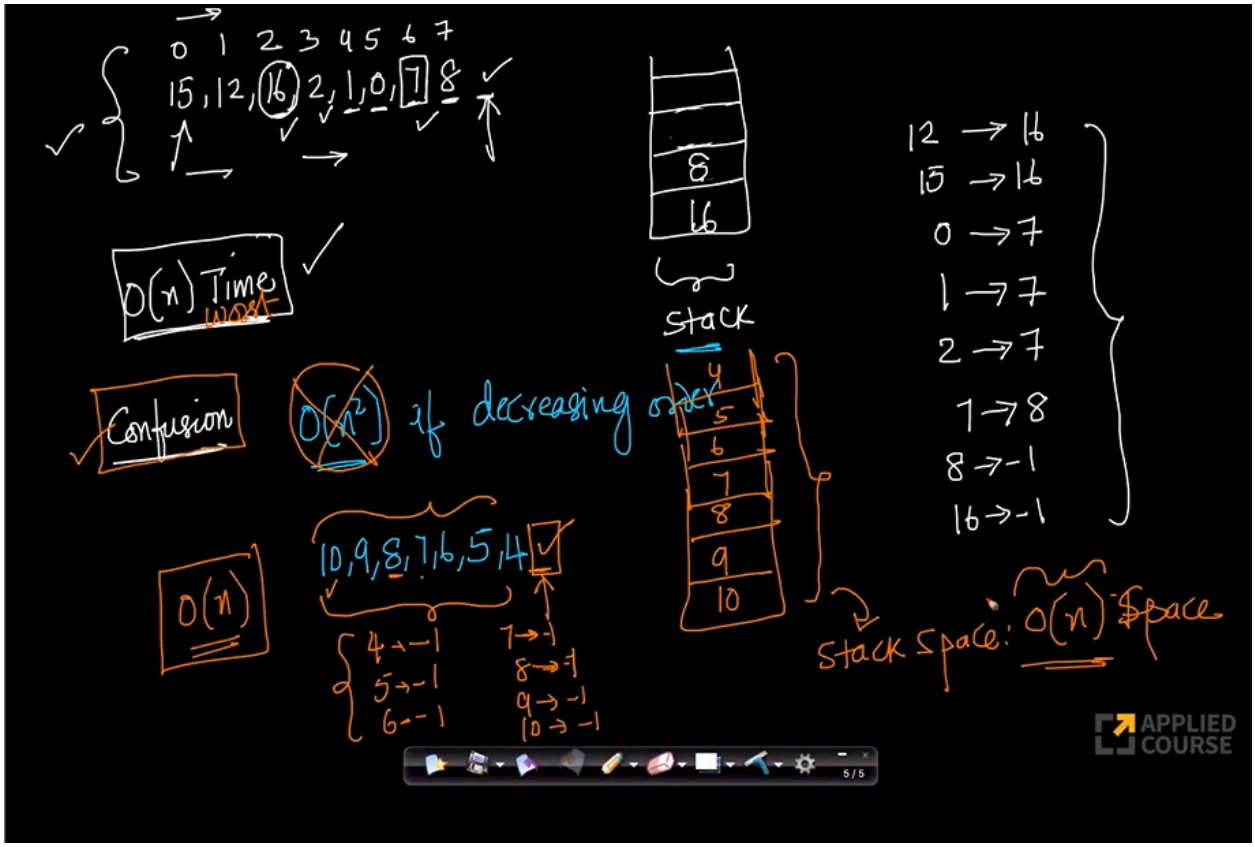
insert 15,12 and …16 is greater …so pop the stack and replace 15 and 12 with 16



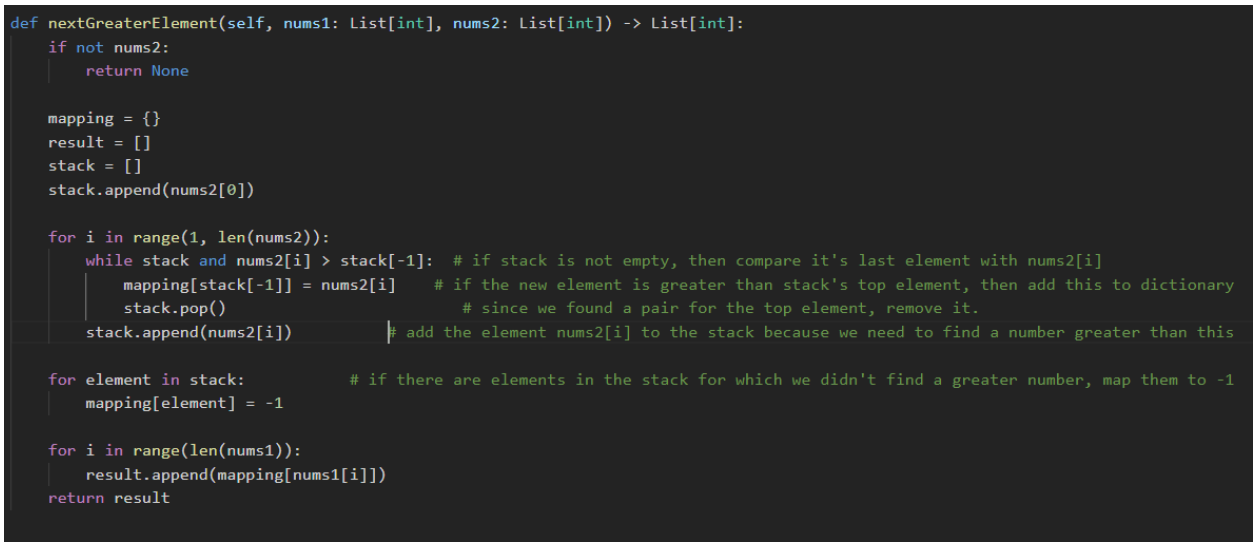
3. And we insert 16 in the stack..
4. Similarly we iterate the entire array…and replace the element with the next greater one



5. Here the space complexity will be O(n)



Python code:

1. 

```python
def nextGreaterElement(self, nums1: List[int], nums2: List[int]) -> List[int]:
    if not nums2:
        return None

    mapping = {}
    result = []
    stack = []
    stack.append(nums2[0])

    for i in range(1, len(nums2)):
        while stack and nums2[i] > stack[-1]:  # if stack is not empty, then compare it's last element with nums2[i]
            mapping[stack[-1]] = nums2[i]    # if the new element is greater than stack's top element, then add this to dictionary
            stack.pop()                      # since we found a pair for the top element, remove it.
        stack.append(nums2[i])          # add the element nums2[i] to the stack because we need to find a number greater than this

    for element in stack:           # if there are elements in the stack for which we didn't find a greater number, map them to -1
        mapping[element] = -1

    for i in range(len(nums1)):
        result.append(mapping[nums1[i]])
    return result
```