Question 1

Given an array nums containing n distinct numbers in the range [0, n], return the only number in the range that is missing from the array.

Example 1:

Input: nums = [3,0,1]

Output: 2

Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

Hashset will be used

Algorithm: 1. Maintain a hashset of element

```
2.No of element = n+1

    2.1. Check if no from 0 to n+1 are present in hashset or not
    if not: return the missing no.
```

In [9]:
```
def find_missing_num(list_ : list)->int:
    n =len(list_)+1
    list1 = [i for i in range(0,n)]
    for i in list1:
        if i not in list_:
            missing_num = i
    return missing_num
```

In [10]:
```
find_missing_num([3,0,1])
```

Out[10]:
```
2
```

Time complexity: O(n)

Space Complexity : O(n)

In [12]:
```
def find_missing_num(list_: list) -> int:
    n = len(list_) + 1
    num_set = set(list_)
    print(num_set)
    for i in range(n):
        if i not in num_set:
            return i
```

In [13]:
```
find_missing_num([3,0,1])
```

```
{0, 1, 3}
```

Out[13]:
```
2
```

Question 2

Given an array of intervals where intervals[i] = [starti, endi], merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

Example 1:

Input: intervals = [[1,3],[2,6],[8,10],[15,18]]

Output: [[1,6],[8,10],[15,18]]

In [37]:
```
def interval_(list_: list)->list:
    list_ = sorted(list_)
    merged = []
    for i in list_:
        if not merged or merged[-1][1]<i[0]:
            merged.append(i)
        else:
            merged[-1][1]= max(merged[-1][1], i[1])
    return merged
```

In [38]:
```
interval_([[1,3],[2,6],[8,10],[15,18]])
```

Out[38]:
```
[[1, 6], [8, 10], [15, 18]]
```

Question 3

You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively. Merge nums1 and nums2 into a single array sorted in non-decreasing order. The final sorted array should not be returned by the function, but instead be stored inside the array nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n

Example 1:

Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3

Output: [1,2,2,3,5,6]

In [67]:
```
def sort_array(list1: list, list2: list) -> list:
    temp = list1[:len(list2)]
    p1 = 0
    p2 = 0
    for i in range(len(list1)):
        if p1< len(temp) and (temp[p1]<=list2[p2]):
            list1[i] = temp[p1]
            p1 = p1+1
        else:
            list1[i] = list2[p2]
            p2 = p2+1
    return list1

sort_array( [1,2,3,0,0,0], [2,5,6])
```

Out[67]:
```
[1, 2, 2, 3, 5, 6]
```

Time complexity: O(n)

Space Complexity: O(1)

Question 4

Given an array nums of size n, return the majority element. The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Example 1:

Input: nums = [3,2,3]

Output: 3

In [93]:
```
def majority(list_ : list)-> list:
    length = len(list_)
    count = []
    for i in range(len(list_)):
        if list_.count(list_[i]) >= length/2:
            return list_[i]
```

In [94]:
```
majority([3,2,3])
```

Out[94]:
```
3
```

Time complexity: O(n^2)

better approach.. hashmap

In [99]:
```
def majority(list_: list)->list:
    length = len(list_)
    dict_ = dict()
    for i in list_:
        if i not in dict_:
            dict_[i] = 1
        else:
            dict_[i]= dict_[i]+1
            if dict_[i] > length/2:
                return i
```

In [100]:
```
majority([3,2,3])
```

Out[100]:
```
3
```

Time Complexity: O(n)

Space Complexity: O(n)

Question 5

Given an array of integers nums containing n + 1 integers where each integer is in the range [1, n] inclusive.

There is only one repeated number in nums, return this repeated number. You must solve the problem without modifying the array nums and uses only constant extra space.

Example 1:

Input: nums = [1,3,4,2,2]

Output: 2

In [110]:
```python
def repeated(list_: list):
    dict_ = dict()
    for i in list_:
        if i not in dict_:
            dict_[i] = 1
        else:
            return i
    return -1
```

In [111]:
```python
repeated([1,3,4,2,2])
```

Out[111]:
```
2
```

In [112]:
```python
repeated([1])
```

Out[112]:
```
-1
```

Question 6

There are many situations where we use integer values as index in array to see presence or absence. We can use bit manipulations to optimize space in such problems.

Let us consider the below problem as an example. Given two numbers say a and b, mark the multiples of 2 and 5 between a and b and output each of the multiples. Note : We have to mark the multiples i.e save (key, value) pairs in memory such that each key either has a value as 1 or 0 representing a multiple of 2 or 5 or not respectively.

Examples

Input : 2 10

Output : 2 4 5 6 8 10

Input: 60 95

Output: 60 62 64 65 66 68 70 72 74 75 76 78 80 82 84 85 86 88 90 92 94 95

In [108]:
```
def multiple_2_and_5(a: int, b: int)->list:
    list_ =[]
    for i in range(a,b+1):
        if i%2 ==0 or i%5==0:
            list_.append(i)
    return list_
```

```
In [109]: multiple_2_and_5(60,95)
```

```
Out[109]: [60,
           62,
           64,
           65,
           66,
           68,
           70,
           72,
           74,
           75,
           76,
           78,
           80,
           82,
           84,
           85,
           86,
           88,
           90,
           92,
           94,
           95]
```

But we are supposed to use bit manipulation

Agorithm:

```
1. Creating an array of size b-1+1

2. For each elememt we are checking if it is divisible by 2 or 5

3. return all numbers marked as 1
```

TC: O(b-a)

In [116]:
```
def main():
    a = 60
    b = 70
    size = abs(b - a) + 1
    array = [0] * size

    for i in range(a, b + 1):
        if i % 2 == 0 or i % 5 == 0:
            array[i - a] = 1

    for i in range(a, b + 1):
        if array[i - a] == 1:
            print(i, end=' ')
```

In [117]:
```
main()
```

```
60 62 64 65 66 68 70
```

In [118]:
```
x = [0]*4
```

Question 7

Given an array of positive integers. We need to make the given array a ‘Palindrome’. The only allowed operation is”merging” (of two adjacent elements). Merging two adjacent elements means replacing them with their sum. The task is to find the minimum number of merge operations required to make the given array a ‘Palindrome’.

To make any array a palindrome, we can simply apply merge operation n-1 times where n is the size of the array (because a single-element array is always palindromic, similar to a single-character string). In that case, the size of array will be reduced to 1. But in this problem, we are asked to do it in the minimum number of operations.

Example :

Input : arr[] = {15, 4, 15}

Output : 0

Array is already a palindrome. So we do not need any merge operation.

Input : arr[] = {1, 4, 5, 1}

Output : 1

We can make given array palindrome with minimum one merging (merging 4 and 5 to make 9)

Input : arr[] = {11, 14, 15, 99}

Output : 3

We need to merge all elements to make a palindrome.

In [125]:
```
def make_palindrome(array):
    length = len(array)
    low = 0
    high = length -1
    count = 0
    while low<=high:
        if array[low]==array[high]:
            low = low + 1
            high = high -1

        elif array[low]> array[high]:
            high = high-1
            array[high]= array[high]+ array[high+1]
            count = count+1
        else:
            low = low +1
            array[low] = array[low]+array[low-1]
            count = count +1
    return count



make_palindrome([1, 4, 5, 9, 1])
```

Out[125]:
```
1
```