**Cyclic sort:**

* **A special type of sorting used to sort the array when the elements of the array are in a specified interval only like [1,n]**

**LC#41:FIRST MISSING POSITIVE**

**Given an unsorted integer array nums. Return the *smallest positive integer* that is *not present* in nums.**

**You must implement an algorithm that runs in O(n) time and uses O(1) auxiliary space.**

**1:Input: nums = [1,2,0] , Output: 3**

**Explanation: The numbers in the range [1,2] are all in the array.**

**2:Input: nums = [3,4,-1,1] ,Output: 2**

**Explanation: 1 is in the array but 2 is missing.**

**3:Input: nums = [7,8,9,11,12] , Output: 1**

**Explanation: The smallest positive integer 1 is missing.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**APPROACH1:(hash map) 1)using int 2)boolean**

**->if n is positive put in the hashmap**

**->check from 1 to n in another loop if ‘I’ is present or not**

class Solution {

public int firstMissingPositive(int[] nums) {

Map<Integer,Integer> hm = new HashMap();

for(int n : nums)

if(n>0)

hm.put(n,hm.getOrDefault(n,-1)+1); **//hm.put(n,true)**

for(int i= 1 ; i<=nums.length+1;i++)

if(!hm.containsKey(i))

return i;

return 1;//never reaches here but for safety}}

**APPROACH 2:SORTING O(N LOG N)**

->**sort the array and start a variable from 1**

**->if it matches do var++**

**->return the value**

class Solution {

public int firstMissingPositive(int[] nums) {

Arrays.sort(nums); // O(n log n)

int smallest = 1;

for (int num : nums) {

if (num == smallest) {

smallest++;

}

}

return smallest;

}

}

**THE ABOVE PASSES BUT NOT ACCORDING TO THE GIVEN CRITERIA**

**APPROACH 3: CYCLIC SORT**

class Solution {

public int firstMissingPositive(int[] nums) {

int n = nums.length;

for(int i = 0 ; i< n ; i++)

{

while(nums[i]>0 &&nums[i]<=n && nums[nums[i]-1]!=nums[i])

{

int crtIdx = nums[i]-1;//nums[i]=3=> crtIdx = 2 nums[2] = 3

int temp = nums[crtIdx];

nums[crtIdx] =nums[i];

nums[i] = temp;

}

}

for(int i = 0 ; i< n ; i++)

{

if(nums[i]!=i+1)

return i+1;

}

return n+1;

}

}

**LC#268:MISSING NUMBER**

**OTHER APPROACHES ARE THERE IN THE BIT MANIPULATION**

**HERE ONLY CYCLIC SORT**

**APPROACH:**

**Sort the array using cyclic**

class Solution {

    public int missingNumber(int[] nums) {

        int n = nums.length;

        int i = 0;

        while(i<n)

        {

            int idx = nums[i**];//here starts from 0 to n so nums[i] else nums[i]-1**

            if( nums[i]<n && nums[idx]!=nums[i]){

                int temp = nums[idx];

                nums[idx] =nums[i];

                nums[i] = temp;

            }

            else{ i++; }

        }

        for(i = 0 ; i<n;i++)

                    if(i!=nums[i])

                return i;

**return n;//for last n if it is not there in the loop**

    }

}

**LC#448:FIND ALL NUMBERS DISAPPEARED IN AN ARRAY**

**Given  nums[i] is in the range [1, n], return *an array of all the integers in the range* [1, n] *that do not appear in* nums.**

**1:Input: nums = [4,3,2,7,8,2,3,1] ,Output: [5,6]**

**2: Input: nums = [1,1] , Output: [2]**

**CODE:**

class Solution {

    public List<Integer> findDisappearedNumbers(int[] nums) {

        int n = nums.length,i=0;

         List<Integer> res = new ArrayList();

        while(i<n)

        {

            int idx = nums[i]-1;

            if(nums[i]!=i+1 && nums[i]!=nums[idx])**//checking whether it is in the right place if duplicates occur it may not be the correct sorting**

            {

                int temp = nums[idx];

                nums[idx] = nums[i];

                nums[i] =temp;

            }

            Else{i++;}

        }

        for( i = 0 ;i<n;i++)

          if(i+1!=nums[i])

          res.add(i+1);

       return res;}}

**LC#287:FIND DUPLICATES (NOT AS PER THE QUESTION MODIFY THE ARRAY NOT ALLOWED BUT KNOWING IS GOOD)**

class Solution {

public int findDuplicate(int[] nums) {

int i = 0;

while (i < nums.length) {

if (nums[i] != i + 1) {

int correctIndex = nums[i] - 1;

if (nums[i] != nums[correctIndex]) {

int temp = nums[i];

nums[i] = nums[correctIndex];

nums[correctIndex] = temp;

} else {

return nums[i]; // Duplicate found

}

} else {

i++;

}

}

return -1;

}

}

**LC#442:FIND DUPLICATES IN A ARRAY**

**1:Input: nums = [4,3,2,7,8,2,3,1] , Output: [2,3]**

**2:Input: nums = [1,1,2] , Output: [1]**

**CODE:**

class Solution {

    public List<Integer> findDuplicates(int[] nums) {

        int i = 0 , n = nums.length;

        Set<Integer> seen = new HashSet<>();

        while (i < n) {

            if (nums[i] != i + 1) {

                int idx = nums[i] - 1;

                if (nums[i] != nums[idx]) {

                    int temp = nums[idx];

                    nums[idx] = nums[i];

                    nums[i] = temp;

                } else {

                    seen.add(nums[i]);

                    i++;

                }

            } else {

                i++;

            }

        }

        return new ArrayList<>(seen);}}

**WITHOUT HASHSET:**

class Solution {

    public List<Integer> findDuplicates(int[] nums) {

        int i = 0 , n = nums.length;

        List<Integer> res = new ArrayList();

        while(i<n)

        {

            if(nums[i]!=i+1)

            {

                int idx = nums[i]-1;

                if(nums[i]!=nums[idx])

                {

                    int temp = nums[idx];

                    nums[idx] =  nums[i] ;

                    nums[i] = temp;

                }

                else

                {

                    i++;//**if dup also increment else stuck in infinite loop**

                }

            }

            else

            {

                i++;

            }

        }

        for(i = 0;i <n;i++)

        {

            if(nums[i]!=i+1)**the array may not be in order but since they have told the interval then nums[2]=3 like that only all there if nums[4] = 3 , then it will add in res**

                res.add(nums[i]);

        }

        return res;

    }

}