Class Notes 2

▸ Python

▸ JavaScript

▾ Java

💡 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.

TC: O(n)

SC: O (n)

```
class Solution {
    public int missingNumber(int[] nums) {
        Set<Integer> numSet = new HashSet<Integer>();
        for (int num : nums) numSet.add(num);

        int expectedNumCount = nums.length + 1;
        for (int number = 0; number < expectedNumCount; number++) {
            if (!numSet.contains(number)) {
                return number;
            }
        }
        return -1;
    }
}
```

💡 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]]

Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].

Solution:

TC : O(nlogn)

SC : O (log n)

```
class Solution {
    public int[][] merge(int[][] intervals) {
        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b
[0]));
        LinkedList<int[]> merged = new LinkedList<>();
        for (int[] interval : intervals) {
            // if the list of merged intervals is empty or if the cur
rent
            // interval does not overlap with the previous, simply ap
pend it.
            if (merged.isEmpty() || merged.getLast()[1] < interval
[0]) {
                merged.add(interval);
            }
        // otherwise, there is overlap, so we merge the current and p
revious
            // intervals.
            else {
                merged.getLast()[1] = Math.max(merged.getLast()[1], i
nterval[1]);
            }
        }
        return merged.toArray(new int[merged.size()][]);
    }
}
```

💡 Question 3

You are given two integer arrays nums1 and nums2, sorted in non-decreasing

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]

Explanation: The arrays we are merging are [1,2,3] and [2,5,6].

The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

Solution:

TC : O(nlogn)

SC : O(log n)

```
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        // Make a copy of the first m elements of nums1.
        int[] nums1Copy = new int[m];
        for (int i = 0; i < m; i++) {
            nums1Copy[i] = nums1[i];
        }

        // Read pointers for nums1Copy and nums2 respectively.
        int p1 = 0;
        int p2 = 0;

        // Compare elements from nums1Copy and nums2 and write the sm
allest to nums1.
        for (int p = 0; p < m + n; p++) {
            // We also need to ensure that p1 and p2 aren't over the
boundaries
            // of their respective arrays.
            if (p2 >= n || (p1 < m && nums1Copy[p1] < nums2[p2])) {
                nums1[p] = nums1Copy[p1++];
            } else {
                nums1[p] = nums2[p2++];
            }
        }
    }
}
```

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

Solution:

TC: O(nlogn)

SC : O(logn)

```
class Solution {
    public int majorityElement(int[] nums) {
        Arrays.sort(nums);
        return nums[nums.length/2];
    }
}
```

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

TC : O(n)

SC : O(n)

```
class Solution {
```

```
class Solution {
    public int findDuplicate(int[] nums) {
        Set<Integer> seen = new HashSet<Integer>();
        for (int num : nums) {
            if (seen.contains(num))
                return num;
            seen.add(num);
        }
        return -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

Solution:

TC : O (n)

SC : O(n)

```
import java.lang.*;

class Main {

    // Driver code
    public static void main(String[] args)
    {
        int a = 60, b = 70;
        int size = Math.abs(b - a) + 1;
        int array[] = new int[size];

        // Iterate through a to b, If
        // it is a multiple of 2 or 5
        // Mark index in array as 1
        for (int i = a; i <= b; i++)
            if (i % 2 == 0 || i % 5 == 0)
                array[i - a] = 1;


        for (int i = a; i <= b; i++)
            if (array[i - a] == 1)
                System.out.printf(i + " ");
    }
}
```

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.

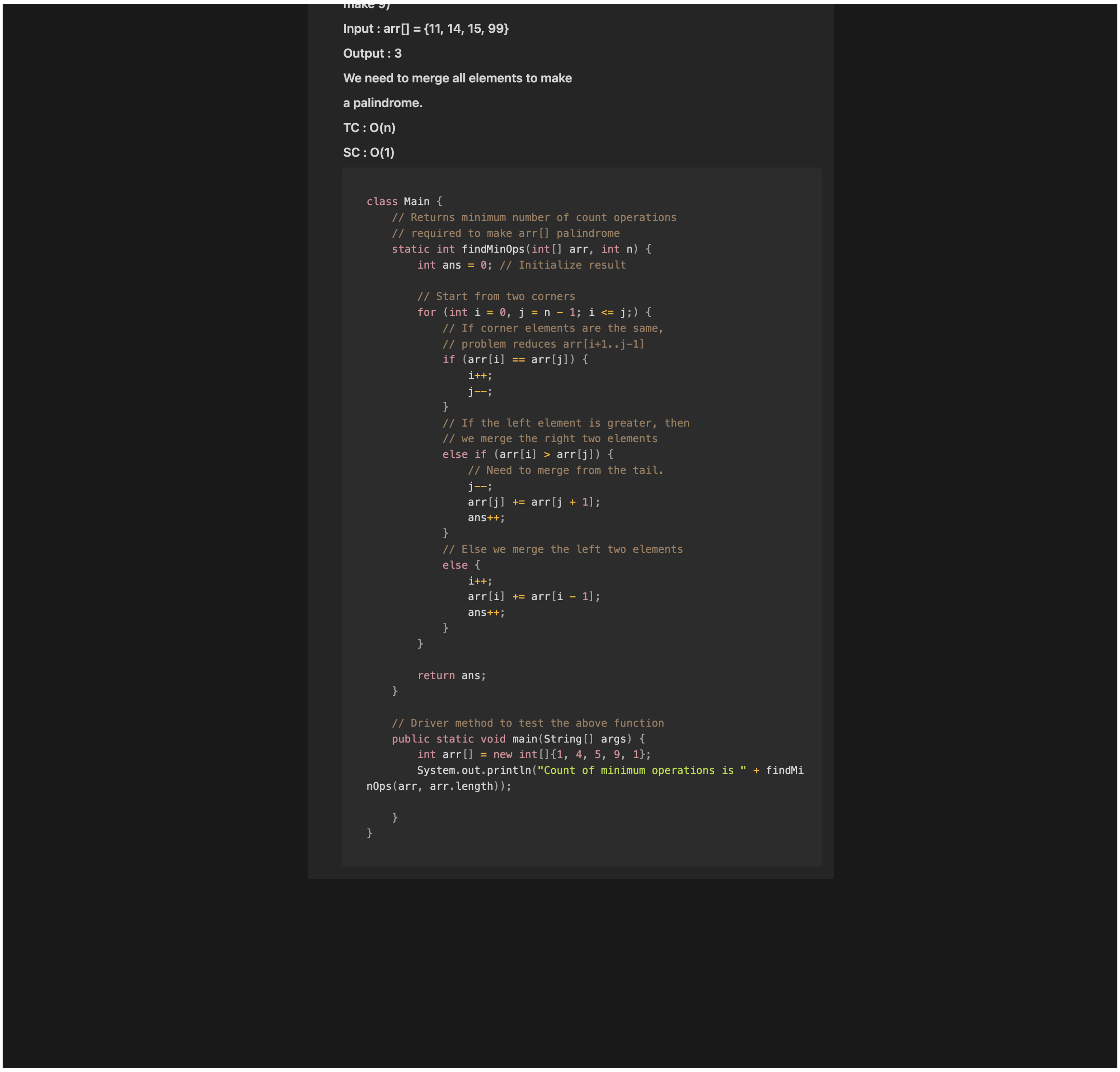
Input : arr[] = {1, 4, 5, 1}

Output : 1

We can make given array palindrome with

minimum one merging (merging 4 and 5 to

make 9)



make 9)

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

Output : 3

We need to merge all elements to make

a palindrome.

TC : O(n)

SC : O(1)

```
class Main {
    // Returns minimum number of count operations
    // required to make arr[] palindrome
    static int findMinOps(int[] arr, int n) {
        int ans = 0; // Initialize result

        // Start from two corners
        for (int i = 0, j = n - 1; i <= j;) {
            // If corner elements are the same,
            // problem reduces arr[i+1..j-1]
            if (arr[i] == arr[j]) {
                i++;
                j--;
            }
            // If the left element is greater, then
            // we merge the right two elements
            else if (arr[i] > arr[j]) {
                // Need to merge from the tail.
                j--;
                arr[j] += arr[j + 1];
                ans++;
            }
            // Else we merge the left two elements
            else {
                i++;
                arr[i] += arr[i - 1];
                ans++;
            }
        }

        return ans;
    }

    // Driver method to test the above function
    public static void main(String[] args) {
        int arr[] = new int[]{1, 4, 5, 9, 1};
        System.out.println("Count of minimum operations is " + findMi
nOps(arr, arr.length));

    }
}
```