1> [1, n], return an array of all the integers in the range [1, n] that do not appear in nums

-> int* findDisappearedNumbers(int* nums, int numSize, int* returnSize)

```
{
    int temp=0;
    for(int index = 0; index < numsSize; ++index)
    {
        temp = abs(nums[index])-1);
        nums[temp] = abs(nums[temp]) * -1;
    }
    int insert_index = 0;
    *returnSize =0;
    for (int index = 0; index < numsSize; ++index
    {
        if(nums[index] >0)
        {
            ++ *returnSize;
            nums[insert_index++] = index + 1;
        }
    }
    return nums;
}
```

o/p:- case1
nums = [4,3,2,7,8,2,3,1]
→ [5,6]

case2
nums = [1,1]
→ [2]

9/5/24

9/5/24

Lab-2:

Binary Tree Zigzag level order traversal

Given the root of a binary tree, return the zigzag level order traversal of its nodes values. (i.e., from left to right then right to left for the next level & alternate between)

->

→ int ** zigzagLevelOrder(struct TreeNode * root, int * returnSize,
int** returnColumnSizes){

```
int **ans = malloc(2000 * sizeof(int *));
*returnColumnSizes = malloc(2000 * sizeof(int));
*returnSize = 0;
struct TreeNode *tmp[2000] = {0};
int top = -1, start = 0;
tmp[++top] = root;
while(tmp[start])
{
  int tmp_top = top;
  ans[(*returnSize)] = malloc((top - start + 1) * sizeof(int));
  (*returnColumnSizes)[(*returnSize)] = (top - start + 1);
  int idx = (*returnSize) % 2 ? (top-start+1)-1:0;
  int step = (*returnSize) % 2 ? -1:1;
  while (start <= tmp_top)
  {
    ans[(*returnSize)][idx] = tmp[start]->val;
    if(tmp[start]->left)
      tmp[++top] = tmp[start]->left;
    if(tmp[start]->right)
      tmp[++top] = tmp[start]->right;
    start++;
    idx += step;
  }
  (*returnSize)++;
}
return ans;
}
```

9/5/24

Case 1:

root = [3,9,20, null, null, 15, 7]

o/p:

[[3], [20,9], [15,7]]

Case 2:

root = [1]

o/p:

[[1]]

Case 3:

root = [ ]

o/p: [ ]