Given an m x n binary matrix mat, return the distance of the nearest 0 for each cell. The distance between two adjacent cells is 1. Input: mat = [[0,0,0],[0,1,0],[0,0,0]] Output: [[0,0,0],[0,1,0],[0,0,0]] Input: mat = [[0,0,0],[0,1,0],[1,1,1]] Output: [[0,0,0],[0,1,0],[1,2,1]]

```
from collections import deque


def updateMatrix(mat):
    rows, cols = len(mat), len(mat[0])
    queue = deque()

    for i in range(rows):
        for j in range(cols):
            if mat[i][j] == 0:
                queue.append((i, j))
            else:
                mat[i][j] = float('inf')

    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    while queue:
        cell = queue.popleft()
        for d in directions:
            new_i, new_j = cell[0] + d[0], cell[1] + d[1]
            if 0 <= new_i < rows and 0 <= new_j < cols and mat[new_i][new_j] > mat[cell[0]][cell[1]] + 1:
                mat[new_i][new_j] = mat[cell[0]][cell[1]] + 1
                queue.append((new_i, new_j))

    return mat

# Test Cases
mat1 = [[0, 0, 0], [0, 1, 0], [0, 0, 0]]
mat2 = [[0, 0, 0], [0, 1, 0], [1, 1, 1]]
```

```
print(updateMatrix(mat1))  # Output: [[0, 0, 0], [0, 1, 0], [0, 0, 0]]
print(updateMatrix(mat2))  # Output: [[0, 0, 0], [0, 1, 0], [1, 2, 1]]
```

Given two integer arrays arr1 and arr2, return the minimum number of operations (possibly zero) needed to make arr1 strictly increasing. In one operation, you can choose two indices 0 <= i < arr1.length and 0 <= j < arr2.length and do the assignment arr1[i] = arr2[j]. If there is no way to make arr1 strictly increasing, return -1. Example 1: Input: arr1 = [1,5,3,6,7], arr2 = [1,3,2,4] Output: 1 Explanation: Replace 5 with 2, then arr1 = [1, 2, 3, 6, 7].

```
def min_operations(arr1, arr2):
    n, m = len(arr1), len(arr2)
    dp = {0: -arr1[0] - 1}
    for i in range(1, n):
        ndp = {}
        for key in dp:
            if arr1[i] > dp[key]:
                ndp[key] = max(ndp.get(key, 0), arr1[i])
            for j in range(m):
                if arr2[j] > dp[key]:
                    ndp[key + 1] = max(ndp.get(key + 1, 0), arr2[j])
        dp = ndp
    if dp:
        return n - max(dp)
    return -1

# Example
arr1 = [1, 5, 3, 6, 7]
arr2 = [1, 3, 2, 4]
print(min_operations(arr1, arr2))  # Output: 1
```

Given two strings a and b, return the minimum number of times you should repeat string a so that string b is a substring of it. If it is impossible for b to be a substring of a after repeating it, return -1. Notice: string "abc" repeated 0 times is "", repeated 1 time is "abc" and repeated 2 times is "abcabc". Example 1: Input: a = "abcd", b = "cdabcdab" Output: 3 Explanation: We return 3 because by repeating a three times "abcdabcdabcd", b is a substring of it.

```
def min_repeats(a, b):
    if b in a:
        return 1
    for i in range(1, len(b)):
        if b.startswith(a[-i:]):
            return i + 1
    return -1

# Example
a = "abcd"
b = "cdabcdab"
print(min_repeats(a, b))  # Output: 3
```

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.

```
def missing_number(nums):
    n = len(nums)
    total_sum = n * (n + 1) // 2
    actual_sum = sum(nums)
    return total_sum - actual_sum

# Example
nums = [3, 0, 1]
print(missing_number(nums))  # Output: 2
```

You are given an n x n integer matrix grid.Generate an integer matrix maxLocal of size (n - 2) x (n - 2) such that: maxLocal[i][j] is equal to the largest value of the 3 x 3 matrix in grid centered around row i + 1 and column j + 1. In other words, we want to find the largest value in every contiguous 3 x 3 matrix in grid. Return the generated matrix. Input: grid = [[9,9,8,1],[5,6,2,6],[8,2,6,4],[6,2,2,2]] Output: [[9,9],[8,6]] Explanation: The diagram above shows the original matrix and the generated matrix. Notice that each value in the generated matrix corresponds to the largest value of a contiguous 3 x 3 matrix in grid.

```
def largestValues(grid):
    n = len(grid)
    maxLocal = [[0] * (n - 2) for _ in range(n - 2)]

    for i in range(n - 2):
        for j in range(n - 2):
            maxLocal[i][j] = max(grid[i][j], grid[i][j + 1], grid[i][j + 2],
                                 grid[i + 1][j], grid[i + 1][j + 1], grid[i + 1][j + 2],
                                 grid[i + 2][j], grid[i + 2][j + 1], grid[i + 2][j + 2])

    return maxLocal

grid = [[9, 9, 8, 1], [5, 6, 2, 6], [8, 2, 6, 4], [6, 2, 2, 2]]
largest_values = largestValues(grid)
print(largest_values)
```

Given the head of a linked list, return the list after sorting it in ascending order. Input: head = [4,2,1,3] Output: [1,2,3,4]

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def sortList(head):
```

```
    if not head or not head.next:
        return head

    mid = get_mid(head)
    left = sortList(head)
    right = sortList(mid)

    return merge(left, right)

def get_mid(head):
    slow = head
    fast = head

    while fast.next and fast.next.next:
        slow = slow.next
        fast = fast.next.next

    mid = slow.next
    slow.next = None

    return mid

def merge(left, right):
    dummy = ListNode()
    curr = dummy

    while left and right:
        if left.val < right.val:
            curr.next = left
            left = left.next
        else:
```

```
            curr.next = right
            right = right.next
        curr = curr.next

    curr.next = left or right

    return dummy.next

# Example
head = ListNode(4)
head.next = ListNode(2)
head.next.next = ListNode(1)
head.next.next.next = ListNode(3)

sorted_head = sortList(head)
result = []
while sorted_head:
    result.append(sorted_head.val)
    sorted_head = sorted_head.next

print(result)
```

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

```
                    from collections import Counter

def majority_element(nums):
    counts = Counter(nums)
    return max(counts, key=counts.get)

# Example
```

```
nums = [3, 2, 3]
print(majority_element(nums))  # Output: 3
```

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)). Example 1: Input: nums1 = [1,3], nums2 = [2] Output: 2.00000 Explanation: merged array = [1,2,3] and median is 2.

```
def findMedianSortedArrays(nums1, nums2):
    nums = sorted(nums1 + nums2)
    n = len(nums)
    if n % 2 == 0:
        return (nums[n // 2 - 1] + nums[n // 2]) / 2
    else:
        return nums[n // 2]

# Example
nums1 = [1, 3]
nums2 = [2]
print(findMedianSortedArrays(nums1, nums2))  # Output: 2.00000
```

Given an array nums of n integers, return an array of all the unique quadruplets [nums[a], nums[b], nums[c], nums[d]] such that: 0 <= a, b, c, d < n a, b, c, and d are distinct. nums[a] + nums[b] + nums[c] + nums[d] == target You may return the answer in any order. Example 1: Input: nums = [1,0,-1,0,-2,2], target = 0 Output: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]] Example 2: Input: nums = [2,2,2,2,2], target = 8 Output: [[2,2,2,2]]

```
def fourSum(nums, target):
    nums.sort()
    result = []
    n = len(nums)
    for i in range(n - 3):
        if i > 0 and nums[i] == nums[i - 1]:
```

```
            continue
        for j in range(i + 1, n - 2):
            if j > i + 1 and nums[j] == nums[j - 1]:
                continue
            left, right = j + 1, n - 1
            while left < right:
                total = nums[i] + nums[j] + nums[left] + nums[right]
                if total == target:
                    result.append([nums[i], nums[j], nums[left], nums[right]])
                    while left < right and nums[left] == nums[left + 1]:
                        left += 1
                    while left < right and nums[right] == nums[right - 1]:
                        right -= 1
                    left += 1
                    right -= 1
                elif total < target:
                    left += 1
                else:
                    right -= 1
    return result

# Example 1
nums1 = [1, 0, -1, 0, -2, 2]
target1 = 0
print(fourSum(nums1, target1))

# Example 2
nums2 = [2, 2, 2, 2, 2]
target2 = 8
print(fourSum(nums2, target2))
```