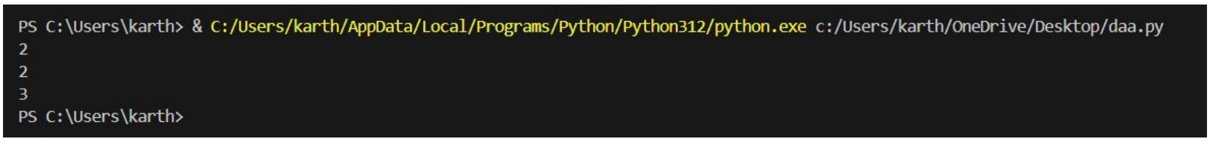
1. Counting Elements Given an integer array arr, count how many elements x there are, such that x + 1 is also in arr. If there are duplicates in arr, count them separately

Code:

```
def count_elements(arr):
   count=0
   elements=set(arr)
   for num in arr:
      if num+1 in elements:
         count+=1
   return count
arr = [1,2,3]
print(count_elements(arr))
arr = [1,1,2,2]
print(count_elements(arr))
arr = [1,3,2,3,5,0]
print(count_elements(arr))
```

output:



```
PS C:\Users\karth> & C:/Users/karth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/karth/OneDrive/Desktop/daa.py
2
2
3
PS C:\Users\karth>
```

Time complexity:

F(n)=o(n)

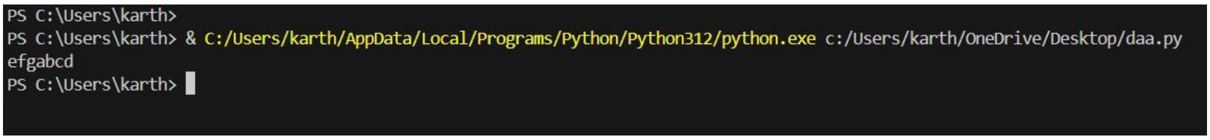
2. Perform String Shifts You are given a string s containing lowercase English letters, and a matrix shift, where shift[i] = [directioni, amounti]: ● directioni can be 0 (for left shift) or 1 (for right shift). ● amounti is the amount by which string s is to be shifted. ● A left shift by 1 means remove the first character of s and append it to the end. ● Similarly, a right shift by 1 means remove the last character of s and add it to the beginning. Return the final string after all operation

Code:

```
def string_shift(s,shift):
   total_shift = 0
   for direction, amount in shift:
      if direction==0:
         total_shift-=amount
```

```
        else:
            total_shift+=amount
    total_shift %= len(s)
    if total_shift < 0:
        return s[-total_shift:]+s[:-total_shift]
    else:
        return s[-total_shift:]+s[:-total_shift]
s = "abcdefg"
shift = [[1,1],[1,1],[0,2],[1,3]]
print(string_shift(s, shift))
```

output:



```
PS C:\Users\karth>
PS C:\Users\karth> & C:/Users/karth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/karth/OneDrive/Desktop/daa.py
efgabcd
PS C:\Users\karth> 
```

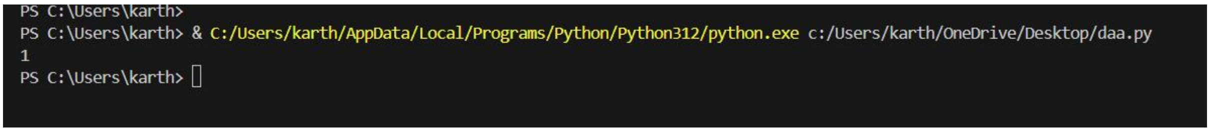
Time complexity:f(n)=n(logn)

3. Leftmost Column with at Least a One A row-sorted binary matrix means that all elements are 0 or 1 and each row of the matrix is sorted in non-decreasing order. Given a row-sorted binary matrix binaryMatrix, return the index (0-indexed) of the leftmost column with a 1 in it. If such an index does not exist, return -1. You can't access the Binary Matrix directly. You may only access the matrix using a BinaryMatrix interface: ● BinaryMatrix.get(row, col) returns the element of the matrix at index (row, col) (0-indexed). ● BinaryMatrix.dimensions() returns the dimensions of the matrix as a list of 2 elements [rows, cols], which means the matrix is rows x cols. Submissions making more than 1000 calls to BinaryMatrix.get will be judged Wrong Answer. Also, any solutions that attempt to circumvent the judge will result in disqualification. For custom testing purposes, the input will be the entire binary matrix mat. You will not have access to the binary matrix directly.

Code:

```
class BinaryMatrix:
    def __init__(self,mat):
        self.mat=mat
    def get(self,row,col):
        return self.mat[row][col]
    def dimensions(self):
        return [len(self.mat), len(self.mat[0])]
def leftMostColumnWithOne(binaryMatrix):
    rows, cols = binaryMatrix.dimensions()
    current_row=0
    current_col=cols-1
    result=-1
    while current_row < rows and current_col >= 0:
        if binaryMatrix.get(current_row, current_col)==1:
            result = current_col
            current_col-=1
        else:
            current_row+=1
    return result
mat = [[0, 0, 0, 1],
    [0, 0, 1, 1],
    [0, 1, 1, 1]]
binaryMatrix = BinaryMatrix(mat)
print(leftMostColumnWithOne(binaryMatrix))
```

output:



```
PS C:\Users\karth>
PS C:\Users\karth> & C:/Users/karth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/karth/OneDrive/Desktop/daa.py
1
PS C:\Users\karth> 
```

Time complexity:

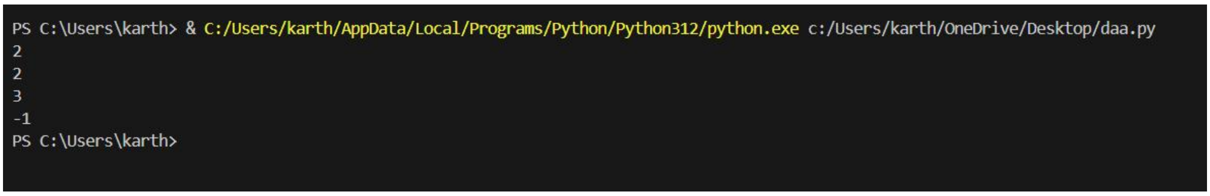
F(n)=o(n)

4. First Unique Number You have a queue of integers, you need to retrieve the first unique integer in the queue. Implement the FirstUnique class: ● FirstUnique(int[] nums) Initializes the object with the numbers in the queue. ● int showFirstUnique() returns the value of the first unique integer of the queue, and returns -1 if there is no such integer. ● void add(int value) insert value to the queue

Code:

```
from collections import deque, Counter
class FirstUnique:
    def __init__(self, nums):
        self.queue = deque(nums)
        self.count = Counter(nums)
    def showFirstUnique(self):
        while self.queue and self.count[self.queue[0]] > 1:
            self.queue.popleft()
        if not self.queue:
            return -1
        return self.queue[0]
    def add(self, value):
        self.queue.append(value)
        self.count[value] += 1
first_unique = FirstUnique([2, 3, 5])
print(first_unique.showFirstUnique())
first_unique.add(5)
print(first_unique.showFirstUnique())
first_unique.add(2)
print(first_unique.showFirstUnique())
first_unique.add(3)
print(first_unique.showFirstUnique())
```

output:



```
PS C:\Users\karth> & C:/Users/karth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/karth/OneDrive/Desktop/daa.py
2
2
3
-1
PS C:\Users\karth>
```

Time complexity:

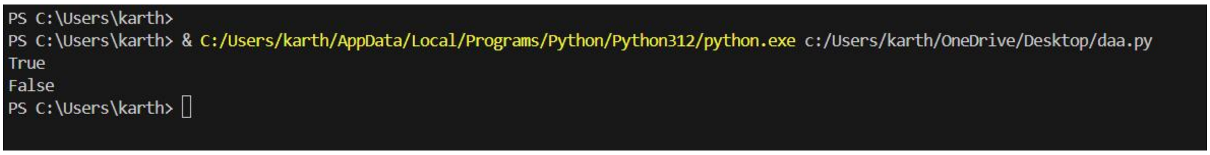
F(n)=o(n)

5. Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree Given a binary tree where each path going from the root to any leaf form a valid sequence, check if a given string is a valid sequence in such binary tree. We get the given string from the concatenation of an array of integers arr and the concatenation of all values of the nodes along a path results in a sequence in the given binary tree

Code:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
class Solution:
    def isValidSequence(self, root: TreeNode, arr: list[int]) -> bool:
        def dfs(node, index):
            if not node:
                return False
            if index >= len(arr) or node.val != arr[index]:
                return False
            if not node.left and not node.right:
                return index == len(arr) - 1
            return dfs(node.left, index + 1) or dfs(node.right, index + 1)
        return dfs(root, 0)
root = TreeNode(0)
root.left = TreeNode(1)
root.right = TreeNode(0)
root.left.left = TreeNode(0)
root.left.right = TreeNode(1)
root.right.left = TreeNode(0)
root.right.right = TreeNode(0)
root.left.left.left = TreeNode(1)
root.left.right.left = TreeNode(0)
root.right.left.left = TreeNode(0)
sol = Solution()
print(sol.isValidSequence(root, [0, 1, 0, 1]))
print(sol.isValidSequence(root, [0, 0, 1]))
```

output:



```
PS C:\Users\karth>
PS C:\Users\karth> & C:/Users/karth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/karth/OneDrive/Desktop/daa.py
True
False
PS C:\Users\karth> 
```

time complexity:

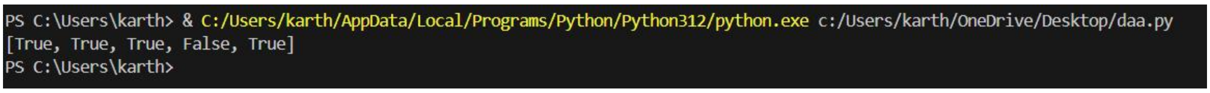
f(n)=o(n)

6. Kids With the Greatest Number of Candies There are n kids with candies. You are given an integer array candies, where each candies[i] represents the number of candies the ith kid has, and an integer extraCandies, denoting the number of extra candies that you have. Return a boolean array result of length n, where result[i] is true if, after giving the ith kid all the extraCandies, they will have the greatest number of candies among all the kids, or false otherwise. Note that multiple kids can have the greatest number of candie

Code:

```
def kidsWithCandies(candies,extraCandies):
   max_candies=max(candies)
   result=[(candy + extraCandies)>=max_candies for candy in candies]
   return result
candies=[2, 3, 5, 1, 3]
extraCandies=3
print(kidsWithCandies(candies,extraCandies))
```

output:



```
PS C:\Users\karth> & C:/Users/karth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/karth/OneDrive/Desktop/daa.py
[True, True, True, False, True]
PS C:\Users\karth>
```

Time complexity:

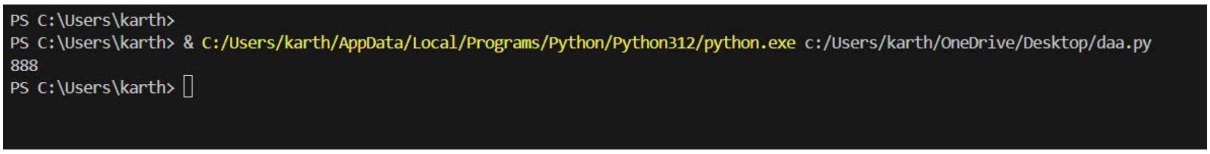
F(n)=n(nlogn)

7. Max Difference You Can Get From Changing an Integer You are given an integer num. You will apply the following steps exactly two times: ● Pick a digit x (0 <= x <= 9). ● Pick another digit y (0 <= y <= 9). The digit y can be equal to x. ● Replace all the occurrences of x in the decimal representation of num by y. ● The new integer cannot have any leading zeros, also the new integer cannot be 0. Let a and b be the results of applying the operations to num the first and second times, respectively. Return the max difference between a and b

Code:

```
def maxDifference(num):
   str_num=str(num)
   def replace_digit(s, x, y):
      return s.replace(x, y)
   max_num = str_num
   for digit in str_num:
      if digit != '9':
            max_num = replace_digit(str_num, digit,'9')

            break

    min_num = str_num

    if str_num[0] != '1':

        min_num = replace_digit(str_num, str_num[0],'1')

    else:

        for digit in str_num[1:]:

            if digit != '0' and digit != '1':

                min_num = replace_digit(str_num, digit,'0')

                break

    return int(max_num)-int(min_num)

num = 555

print(maxDifference(num))
```

output:



```
PS C:\Users\karth>
PS C:\Users\karth> & C:/Users/karth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/karth/OneDrive/Desktop/daa.py
888
PS C:\Users\karth> 
```

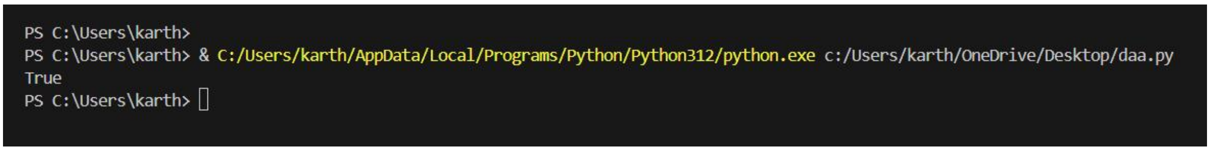
Time complexity:f(n)=0(n)

8. Check If a String Can Break Another String Given two strings: s1 and s2 with the same size, check if some permutation of string s1 can break some permutation of string s2 or vice-versa. In other words s2 can break s1 or vice-versa. A string x can break string y (both of size n) if x[i] >= y[i] (in alphabetical order) for all i between 0 and n-1

Code:

```
def checkIfCanBreak(s1, s2):

    sorted_s1=sorted(s1)

    sorted_s2=sorted(s2)

    can_s1_break_s2 = all(c1 >= c2 for c1, c2 in zip(sorted_s1, sorted_s2))

    can_s2_break_s1 = all(c2 >= c1 for c2, c1 in zip(sorted_s2, sorted_s1))

    return can_s1_break_s2 or can_s2_break_s1

s1="abc"

s2="xya"

print(checkIfCanBreak(s1, s2))
```

output:



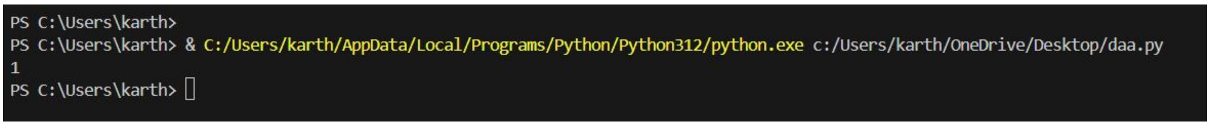
```
PS C:\Users\karth>
PS C:\Users\karth> & C:/Users/karth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/karth/OneDrive/Desktop/daa.py
True
PS C:\Users\karth> 
```

9. Number of Ways to Wear Different Hats to Each Other There are n people and 40 types of hats labeled from 1 to 40. Given a 2D integer array hats, where hats[i] is a list of all hats preferred by the ith person. Return the number of ways that the n people wear different hats to each other. Since the answer may be too large, return it modulo 109 + 7

Code:

```
def numberOfWaysToWearHats(hats):
   MOD=10**9 + 7
   n=len(hats)
   hat_to_people=[[] for _ in range(41)]
   for person in range(n):
      for hat in hats[person]:
         hat_to_people[hat].append(person)
   dp=[0]*(1 << n)
   dp[0]=1
   for hat in range(1,41):
      for mask in range((1 << n) - 1, -1, -1):
         if dp[mask]==0:
            continue
         for person in hat_to_people[hat]:
            if (mask & (1 << person))==0:
               new_mask = mask | (1 << person)
               dp[new_mask]=(dp[new_mask] + dp[mask]) % MOD
   return dp[(1 << n) - 1]
hats=[[3, 4], [4, 5], [5]]
print(numberOfWaysToWearHats(hats))
```

output:



```
PS C:\Users\karth>
PS C:\Users\karth> & C:/Users/karth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/karth/OneDrive/Desktop/daa.py
1
PS C:\Users\karth>
```

Time complexity:

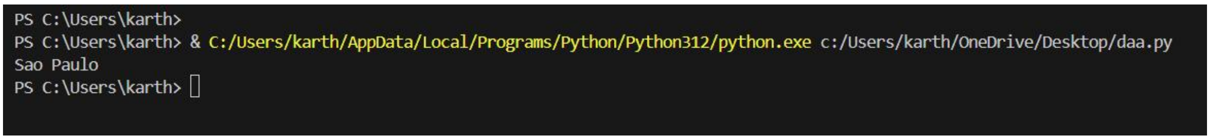
F(n)=0(m+n)

10. . Destination City You are given the array paths, where paths[i] = [cityAi, cityBi] means there exists a direct path going from cityAi to cityBi. Return the destination city, that is, the city without any path outgoing to another city. It is guaranteed that the graph of paths forms a line without any loop, therefore, there will be exactly one destination city

Code:

```
def destCity(paths):
   outgoing=set(cityA for cityA, cityB in paths)
   for cityA,cityB in paths:
      if cityB not in outgoing:
         return cityB
paths=[["London", "New York"],["New York", "Lima"],["Lima", "Sao Paulo"]]
print(destCity(paths))
```

output:



```
PS C:\Users\karth>
PS C:\Users\karth> & C:/Users/karth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/karth/OneDrive/Desktop/daa.py
Sao Paulo
PS C:\Users\karth>
```

Time complexity:

F(n)=o(n)