```
# ==============Q1============#
def diStringMatch(self, S: str) -> List[int]:
    out = []
    inc = 0
    dec = len(S)
    for i in range(len(S)):
        if S[i] == 'I':
            out.append(inc)
            inc += 1
        elif S[i] == 'D':
            out.append(dec)
            dec -= 1

    if S[-1] == 'D':
        out.append(dec)
    else:
        out.append(inc)

    return out


# ============Q2============#

class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
        for i in range(len(matrix)):
            for j in range(len(matrix[0])):
                if matrix[i][j] == target: return True
        return False

    # =====================Q3=============#
    class Solution(object):
    def validMountainArray(self, arr):
        """
        :type arr: List[int]
        :rtype: bool
        """
        index=len(arr)
        i=0
        for i in range(0,index-1): #index-1 to avoid index out of bounds
```

```
            if arr[i+1]>arr[i]:#strictly increasing
                continue
            else:
                break
        if i==0 or i==index-1:
            return False
        for j in range(index-1,i,-1):
            if arr[j-1]>arr[j]: #strictly decreasing
                continue
            else:
                return False
        return True #control comes here only if both the for loops are
traversed.

    # ============Q4===============#

    class Solution:
    def findMaxLength(self, nums: List[int]) -> int:
        max_length =0
        hash={}
        count=0
        for i in range(len(nums)):
            current=nums[i]
            if current==0:
                count-=1 # decrement our count if our current element is 0
            else:
                # increment our count if current element is 1
             count+=1

            if count==0:
                # if count is 0, we have a new subarray with length+1
                max_length=i+1
            if count in hash:
                max_length=max(max_length,i-hash[count])
            else:
                hash[count]=i
        return max_length


    # ==============Q5============#
    class Solution:
    def minProductSum(self, nums1: List[int], nums2: List[int]) -> int:
        nums1.sort()
        nums2.sort()
        n, res = len(nums1), 0
        for i in range(n):
            res += nums1[i] * nums2[n - i - 1]
        return res

    # ============Q6=============#

    class Solution:
    def findOriginalArray(self, nums: List[int]) -> List[int]:
            ans = []
            #answer storing array
            vacans = []
            #when we need to return vacant array
            if len(nums)%2:
                    return ans
                    #when we will have odd number of integer in our
input(double array can't be in odd number)

            nums.sort()
            #sorting

            temp = Counter(nums)
            #storing the frequencies
            for i in nums:
                if temp[i] == 0:
                #if we have already decreased it's value when we were
checking y/2 value, like 2,4 we will remove 4 also when we will check 2
but our iteration will come again on 4.

                    continue
                else:
                    if temp.get(2*i,0) >= 1:
                    #if we have both y and y*2, store in our ans array
                        ans.append(i)
                        #decrease the frequency of y and y*2
                        temp[2*i] -= 1
                        temp[i] -= 1
                    else:
                        return vacans
            return ans
    # ===================Q7===============#
    def generateMatrix(self, n):
    A = [[0] * n for _ in range(n)]
    i, j, di, dj = 0, 0, 0, 1
    for k in xrange(n*n):
        A[i][j] = k + 1
        if A[(i+di)%n][(j+dj)%n]:
            di, dj = dj, -di
        i += di
        j += dj
    return A

# =============Q8===============#
class Solution:
    def multiply(self, mat1: List[List[int]], mat2: List[List[int]]) ->
List[List[int]]:
        r1, c1, c2 = len(mat1), len(mat1[0]), len(mat2[0])
        res = [[0] * c2 for _ in range(r1)]
        for i in range(r1):
            for j in range(c2):
                for k in range(c1):
                    res[i][j] += mat1[i][k] * mat2[k][j]
        return res
```