Reverse in pairs

```
a = 2345
def reverse_in_pairs(n):
  n_str = str(n) if n>0 else str(-n)
  res = []
  for i in range(0,len(n_str), 2):
    if i+1 < len(n_str):
      res.append(n_str[i+1])
    res.append(n_str[i])
  ans = ''.join(res)
  return '-'+ans if n<0 else ans

print(reverse_in_pairs(a))
```

Divisor Substrings

```
n = 2345
k = 1

def divisor_substrings(n, k):
  nstr = str(n) if n>0 else str(-n)
  divisors = set()
  res = 0
  for i in range(0, len(nstr)-k+1):
    divisor = int(nstr[i:i+k])
    if divisor not in divisors and divisor != 0:
      divisors.add(divisor)
      if n%divisor == 0:
        res += 1
  return res

print(divisor_substrings(n, k))
```

Longest equal subarray

```
arr = [0,1,1,1,1,1,0,0]
def longest_equal_subarray(a):
  for i in range(len(a)):
    if a[i] == 0:
      a[i] = -1

  m = {0:-1}
  res = nsum = 0

  for i in range(len(a)):
    nsum += a[i]
    if nsum in m:
      res = max(i-m[nsum], res)
    else:
      m[nsum] = i

  return res

print(longest_equal_subarray(arr))
```

Max queries

```
ar = [1,1,2,3,2]
qs = [[1,2,1],
      [2,4,2],
      [0,3,1]]

def binary_search(a, target):
  lo, hi = 0, len(a)
  while lo<hi:
    m = lo + (hi-lo)//2
    if a[m] == target:
      return m, True
    if a[m] > target:
      hi = m
    else:
      lo = m+1
  return lo, False

def max_queries(arr, queries):
  from collections import defaultdict
  m = defaultdict(list)
  for i in range(len(arr)):
    m[arr[i]].append(i)

  res = 0
  for left, right, x in queries:
    if x not in m:
      continue
    index_arr = m[x]
    if right < index_arr[0] or left > index_arr[-1]:
      continue

    a, _ = binary_search(index_arr, left)
    b, is_found = binary_search(index_arr, right)
    if not is_found: b-=1
    res += (b-a)+1
  return res

print(max_queries(ar, qs))
```

Diagonal Sort

```
mat = [[1,2,3],
       [8,5,7],
       [7,3,10]]

def diagonal_sort(arr):
  row, col = len(arr)-1, 0
  is_half_way = False
  while row>=0:
    r,c = row, col
    temp = []
    while c>=0 and r>=0:
      temp.append(arr[r][c])
      r,c = r-1, c-1

    temp.sort()
    r,c = row, col
    while c>=0 and r>=0:
      arr[r][c] = temp.pop()
      r,c = r-1, c-1

    if not is_half_way:
      col +=1
      if col == len(arr[0]):
        col = len(arr[0])-1
        row = len(arr)-2
        is_half_way = True
    else:
      row -= 1
```

```
def mat_printer(arr):
  for row in arr:
    print(row)
  print()

mat_printer(mat)
diagonal_sort(mat)
mat_printer(mat)
```

Rotate matrix k times

```
arr = [[1,2,3,4],
       [5,6,7,8],
       [9,10,11,12],
       [13,14,15,16]]

def reverse_every_row(mat):
  for i in range(len(mat)):
    j, k = 0,len(mat[i])-1
    while j<k:
      mat[i][j], mat[i][k] = mat[i][k], mat[i][j]
      j, k = j+1, k-1

def transpose(mat):
  for i in range(len(mat)):
    for j in range(i, len(mat[0])):
      mat[i][j], mat[j][i] = mat[j][i], mat[i][j]
  return mat

def get_diagonals(mat):
  eles = []
  for i in range(len(mat)):
    eles.append(mat[i][i])
    eles.append(mat[i][len(mat[i])-i-1])
  return eles

def set_diagonals(mat, eles):
  index = 0
  for i in range(len(mat)):
    mat[i][i] = eles[index]
    index+=1
    mat[i][len(mat[i])-i-1] = eles[index]
    index+=1

def rotate_matrix_by_k(mat, k):
  k = k%4
  if k == 0:
    return mat

  diag_eles = get_diagonals(mat)

  if k == 3:
    reverse_every_row(mat)
    transpose(mat)
  else:
    for i in range(k):
      transpose(mat)
      reverse_every_row(mat)
  set_diagonals(mat, diag_eles)

def mat_print():
  for row in arr:
    print(row)
  print()

mat_print()
rotate_matrix_by_k(arr, 1)
mat_print()
rotate_matrix_by_k(arr, 1)
mat_print()
rotate_matrix_by_k(arr, 1)
mat_print()
rotate_matrix_by_k(arr, 1)
mat_print()
```

Sort and fill from bottom right

```
from collections import defaultdict

a =  [[5,1,1],
      [1,-4,-4],
      [2,2,5,]]

def custom_sort(arr):
  m = defaultdict(int)
  for m1 in arr:
    for ele in m1:
      m[ele]+=1

  eles = []
  for freq, num in sorted([(freq, num) for num, freq in m.items()]):
    eles.extend([num]*freq)
  index = 0

  # bottom diagonal traversal
  row,col = len(arr)-1, len(arr[0])-1
  is_half_done = False
  while row != -1:
    r, c = row, col
    while 0 <= r < len(arr) and 0 <= c < len(arr[0]):
      arr[r][c] = eles[index]
      index+=1
      r-=1
      c+=1

    if not is_half_done:
      col -= 1
    else:
      row -= 1

    if col == -1:
      row-=1
      col = 0
      is_half_done = True

  return arr

print(custom_sort(a))
```

Smallest element in matrix queries

```
class Min:
  def __init__(self, n):
    self.n = n
    self.m = []
    for i in range(n):
      self.m.append([(i+1)*(j+1) for j in range(n)])
    self.row = self.col = 0
    self.disabled_rows = set()
    self.disabled_cols = set()

  def find_min(self):
    return self.m[self.row][self.col]

  def disable_row(self, r):
    self.disabled_rows.add(r)
    r = self.row
    while self.row in self.disabled_rows:
      self.row+=1

  def disable_col(self, c):
    self.disabled_cols.add(c)
    while self.col in self.disabled_cols:
      self.col+=1

m = Min(5)
print(m.find_min())
m.disable_row(4)
print(m.find_min())
m.disable_col(1)
print(m.find_min())
m.disable_col(0)
print(m.find_min())
m.disable_row(0)
print(m.find_min())
m.disable_row(2)
m.disable_col(2)
print(m.find_min())
m.disable_row(1)
print(m.find_min())
```

Best Squares

```
mat = [[1,2,3,4,5],
       [0,1,10,2,0],
       [1,3,5,7,6],
       [0,1,4,2,3]]

def mat_printer(mat):
    for row in mat:
        print(row)
    print()

def bestSquares(m, k):
    new_mat = []
    for i in range(len(m)+1):
        new_mat.append([0]*(len(m[0])+1))

    for i in range(len(m)):
        so_far_sum = 0
        for j in range(len(m[0])):
            so_far_sum += m[i][j]
            new_mat[i+1][j+1]= new_mat[i][j+1] + so_far_sum

    max_square_sum = 0
    for i in range(len(m)):
        for j in range(len(m[0])):
            if i+k-1 >= len(m) or j+k-1 >= len(m[0]):
                continue
            max_square_sum = max(max_square_sum, new_mat[i+k][j+k] -
new_mat[i+k][j] - new_mat[i][j+k]+ new_mat[i][j])

    uniq_eles = set()
    for i in range(len(m)):
        for j in range(len(m[0])):
            if i+k-1 >= len(m) or j+k-1 >= len(m[0]):
                continue
            cur_sum = new_mat[i+k][j+k] - new_mat[i+k][j] - new_mat[i][j+k]+
new_mat[i][j]
            if cur_sum == max_square_sum:
                for x in range(i,i+k):
                    for y in range(j, j+k):
                        uniq_eles.add(m[x][y])
    return sum(uniq_eles)

print(bestSquares(mat, 2))
```