Task 1:

For task one I made the naive approach by first calculating the total

no of possible moves for the queen and then visiting each obstacle to

find out if it will affect our current calculations

Time Complexity : O(k)

Space Complexity : O(1)

Code :

def queensAttack(n, k, r\_q, c\_q, obstacles):

u = n - r\_q

d = r\_q-1

r = n - c\_q

l = c\_q-1

ru = min(u, r)

rd = min(r,d)

lu = min(l,u)

ld = min(l,d)

for o in obstacles:

if o[1] == c\_q:

if o[0] < r\_q:

d = min(d, r\_q-1-o[0])

else:

u = min(u, o[0]-r\_q-1)

elif o[0] == r\_q:

if o[1] < c\_q: l = min(l, c\_q-1-o[1])

else: r = min(r, o[1]-c\_q-1)

elif abs(o[0]-r\_q) == abs(o[1]-c\_q):

if o[1]>c\_q:

if o[0]>r\_q: ru = min(ru, o[1]-c\_q-1)

else: rd = min(rd, o[1]-c\_q-1)

else:

if o[0]>r\_q: lu = min(lu, c\_q-1-o[1])

else: ld = min(ld, c\_q-1-o[1])

return u + d + r + l + ru + rd + lu + ld

n,k = (input().split())

n=int(n)

k=int(k)

r\_q,c\_q = (input().split())

r\_q=int(r\_q)

c\_q=int(c\_q)

obstacles=[]

for i in range(k):

obstacles.append([int(j) for j in input().split()])

print(queensAttack(n,k,r\_q,c\_q,obstacles))

Test Case : 4 0

4 4

Output : 9

--------------------------------------------------------------------------------------------------------------------------------------

Task 2:

For this task I took the input in a list and then sorted the list.

After sorting the list I swapped elements from index 1 to n-2 with

the subsequent element to it. Important point was to note that doing

this for each pair the swap was performed only once and hence i was

incremented by a step of 2 units.

Time Complexity : O(nlogn+n)= O(nlogn)

Space Complexity : O(n)

Code :

def swap(self, i, j):

self[i], self[j] = self[j], self[i]

n = int(input())

arr = input()

l = list(map(int,arr.split(' ')))

l.sort()

for i in range(1,n-1,2):

swap(l,i,i+1)

for i in range(0,n):

print(l[i],end=" ")

Test Case : 10

10 14 6 8 2 12 4 16 20 18

Output : 2 6 4 10 8 14 12 18 16 20