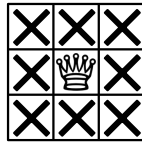
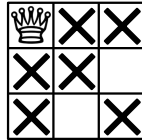


**Question 1.a** – On distingue 3 cas :

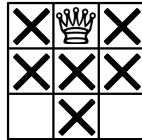
→ Si la première dame est placée au centre de l'échiquier, elle attaque toutes les autres cases et on ne peut pas placer la deuxième dame :



→ Si la première dame est placée dans un coin, il reste deux cases non attaquées. Étant donné que ces deux cases sont sur la même diagonale, il est impossible de placer les deux autres dames :



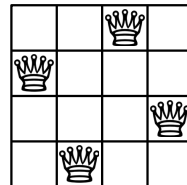
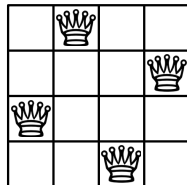
→ Si la première dame est placée sur un bord, il reste deux cases non attaquées. Étant donné que ces deux cases sont sur la même ligne ou la même colonne, il est impossible de placer les deux autres dames :



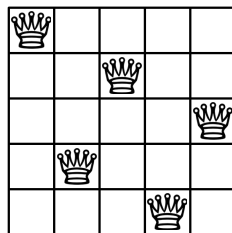
En conclusion :

Il n'y a pas de solution au problème des  $n$  dames pour  $n = 3$ .

**Question 1.b** – Pour  $n = 4$ , il y a deux solutions au problème des  $n$  dames :



**Question 1.c** – Voici une solution possible :



**Question 2** –

```

1 | def testE(Q) :
2 |     n = len(Q)
3 |     for e in Q:
4 |         if e < 0 or e >= n:
5 |             return False
6 |     return True

```

### Question 3.a –

```
1 | def makeC(Q):
2 |     n = len(Q)
3 |     C = [0]*n
4 |     for i in range(n):
5 |         j = Q[i]
6 |         C[j] += 1
7 |     return C
```

### Question 3.b –

```
1 | def testC(Q):
2 |     C = makeC(Q)
3 |     for e in C:
4 |         if e != 1:
5 |             return False
6 |     return True
```

### Question 3.c –

★ La fonction `makeC` est de complexité linéaire en  $n$ . En effet, la création de la ligne 7 s'exécute en temps linéaire en  $n$ . De plus il y a  $n$  tours de boucle et chaque tour s'exécute en temps constant.

★ La fonction `testC` est de complexité linéaire en  $n$ . En effet, l'appel à la fonction `makeC` de la liste ligne 6 s'exécute en temps linéaire en  $n$ . De plus il y a au plus  $n$  tours de boucle et chaque tour s'exécute en temps constant.

### Question 4 –

```
1 | # On utilise un dictionnaire (d: dict[int, NoneType]) pour vérifier si tous les
2 | # (i+j) sont différents
3 | def testAD(Q):
4 |     d = {}
5 |     n = len(Q)
6 |     for i in range(n):
7 |         j = Q[i]
8 |         if i + j in d:
9 |             return False
10 |        else:
11 |            d[i+j] = None
12 |    return True
```

**Question 5.a** – Pour chaque case de coordonnées  $(i, j)$  dans la grille, calculons  $i - j$ . Par exemple, pour  $n = 8$ , on obtient :

0	-1	-2	-3	-4	-5	-6	-7
1	0	-1	-2	-3	-4	-5	-6
2	1	0	-1	-2	-3	-4	-5
3	2	1	0	-1	-2	-3	-4
4	3	2	1	0	-1	-2	-3
5	4	3	2	1	0	-1	-2
6	5	4	3	2	1	0	-1
7	6	5	4	3	2	1	0

On remarque alors que la case de coordonnées  $(i_1, j_1)$  et la case de coordonnées  $(i_2, j_2)$  sont sur la même diagonale si et seulement si  $i_1 - j_1 = i_2 - j_2$ .

**Question 5.b** –

```

1 | # On applique la méthode vue en cours qui permet de traduire une fonction
2 | # itérative en fonction récursive.
3 | def testDAux(Q, d, i):
4 |     if i == len(Q):
5 |         return True
6 |     j = Q[i]
7 |     if i-j in d:
8 |         return False
9 |     else:
10 |         d[i-j] = None
11 |         return testDAux(Q, d, i+1)

1 | def testD(Q):
2 |     return testDAux(Q, {}, 0)

```

**Question 6.a** –

```

1 | def estSol(Q):
2 |     b1 = testC(Q)
3 |     b2 = testAD(Q)
4 |     b3 = testD(Q)
5 |     return b1 and b2 and b3

```

**Question 6.b** – D'après les questions précédentes, l'appel à chacune des trois fonctions intermédiaires s'exécute en temps linéaire en  $n$ . La fonction `estSol` s'exécute donc en temps linéaire en  $n$ .

**Question 7** –

```

1 | def FY(n):
2 |     Q = [i for i in range(n)]
3 |     for i in range(n-1, 0, -1):
4 |         j = random.randint(0,i)
5 |         tmp = Q[i]
6 |         Q[i] = Q[j]
7 |         Q[j] = tmp
8 |     return Q

```

Question 8 –

```
1 def solAlea(n):
2     assert n == 0 or n == 1 or n >= 4
3     while True:
4         Q = FY(n)
5         if estSol(Q):
6             return Q
```

Question 9.a –

```
1 # Suppose qu'au moins une dame est attaquée.
2 def dameABouger(Q, A):
3     n = len(Q)
4     while True:
5         i = random.randint(0,n-1)
6         j = Q[i]
7         if A[i][j] != 1:
8             return i
```

Question 9.b –

```
1 def nvPos(i, A):
2     L = A[i]
3     n = len(L)
4     if L[0] <= L[1]:
5         j1 = 0; j2 = 1
6     else:
7         j1 = 1; j2 = 0
8     for j in range(2,n):
9         if L[j] < L[j1]:
10            j2 = j1; j1 = j
11        elif L[j] < L[j2]:
12            j2 = j
13    return j1, j2
```

Question 9.c –

```
1 # Attention: modifie la liste donnée en entrée
2 def etape2b(Q, A):
3     i = dameABouger(Q, A)
4     j1,j2 = nvPos(i, A)
5     if j1 == Q[i]:
6         Q[i] = j2
7     else:
8         Q[i] = j1
9     return Q
```

### Question 10 –

```
1 def makeA(Q):
2     n = len(Q)
3     attLignes = {i: 0 for i in range(n)}
4     attCol = {j: 0 for j in range(n)}
5     attAntiDiag = {k: 0 for k in range(2*n-1)}
6     attDiag = {k: 0 for k in range(-n+1, n)}
7     for i in range(n):
8         j = Q[i]
9         attLignes[i] += 1
10        attCol[j] += 1
11        attAntiDiag[i+j] += 1
12        attDiag[i-j] += 1
13    A = [[attLignes[i] + attCol[j] + attAntiDiag[i+j] + attDiag[i-j]
14          for j in range(n)] for i in range(n)]
15    ### La dame (i,j) n'attaque qu'une fois la case (i,j)
16    for i in range(n):
17        A[i][Q[i]] -= 3
18    return A
```

### Question 11 –

```
1 def solRI(n):
2     while True:
3         Q = FY(n)
4         # print("restart")
5         if estSol(Q):
6             return Q
7         for _ in range(1000):
8             A = makeA(Q)
9             Q = etape2b(Q, A)
10            if estSol(Q):
11                return Q
```

### Question 12 –

```
1 # Indique si la case de coordonnées (i,j) est dans la grille de taille n x n
2 def estDansGrille(i,j,n):
3     return 0 <= i and i <= n-1 and 0 <= j and j <= n-1
```

```

1 | # Cette fonction renvoie une liste de booléens att telle que att[j] vaut True
2 | # lorsque la case de coordonnées (i,j) est attaquée par une dame d'indice
3 | # i0 < i.
4 | def makeAtt(i, Q):
5 |     n = len(Q)
6 |     att = [False] * n
7 |     for i0 in range(i):
8 |         j0 = Q[i0]
9 |         ### Attaque sur la colonne
10 |         att[j0] = True
11 |         ### Attaque sur l'anti-diagonale
12 |         j = i0 + j0 - i
13 |         if estDansGrille(i,j,n):
14 |             att[j] = True
15 |         ### Attaque sur la diagonale
16 |         j = i - i0 + j0
17 |         if estDansGrille(i,j,n):
18 |             att[j] = True
19 |     return att

```

```

1 | # On suppose que les i premières dames ont été placées.
2 | # Cette fonction place la dame numéro i puis place les autres dames avec des
3 | # appels récursifs.
4 | def nbSolAux(i, Q):
5 |     n = len(Q)
6 |     if i >= n:
7 |         return 1
8 |     att = makeAtt(i, Q)
9 |     L = [j for j in range(n) if not att[j]]
10 |     res = 0
11 |     for j in L:
12 |         Q[i] = j
13 |         res += nbSolAux(i+1, Q)
14 |     return res

```

```

1 | # Hypothèse: n >= 0.
2 | def nbSol(n):
3 |     Q = [0]*n
4 |     return nbSolAux(0, Q)

```