Bài 1:

Một số đặc điểm của bài toán brute force: trực quan, trực tiếp và đơn giản, cần phải liệt kê tất cả các cách có thể hoặc tất cả các giải pháp khả thi nhằm giải quyết vấn đề và có vẻ như không tồn tại cách giải nào tối ưu hơn được nữa.

Bài 2:

```
state = []
ss = 0

def check(x, y):
    #diag
    k = (abs(x[0]-y[0]), abs(x[1]-y[1]))
    if k[0]==k[1]:
        return False
    if k[0]==0:
        return False
    if(k[1]==0):
        return False
    return True

def calc(n, k):
    global ss
    ss = ss + 1
    if n==k:
        return 1
    s = 0
    for i in range(0,k):
        k = (i, n)
        c = True
        for j in state:
            if check(j, k)==False:
                c = False
        if c==True:
            state.append(k)
            s = s + calc(n+1)
            state.pop()
    return s

kk = int(input())
print(calc(n,kk))
```

Bước 1: Nếu đã chọn hết hàng, kiểm tra, nếu đúng tăng biến đếm số nghiệm lên 1.

Bước 2: Chọn 1 cột trong số 8 cột trong hàng n thỏa mãn có thể đặt quân hậu vào

Bước 3: Đặt vào vị trí đó, lưu lại và tiếp tục tăng n lên, quay về bước 1

Bài 3:

```
import functools

def sudoku_solver(table):
    unsolved = []
    def check(game_state):
        for i in range(0, 9):
            temp = sorted((table[i][0], table[i][1], table[i][2], table[i][3], table[i][4],
                           table[i][5], table[i][6], table[i][7], table[i][8]))
            for x in range(1, 9):
                if temp[x]==temp[x-1] and temp[x]!=0:
                    return False

            temp = sorted((table[0][i], table[1][i], table[2][i], table[3][i], table[4][i],
                           table[5][i], table[6][i], table[7][i], table[8][i]))
            for x in range(1, 9):
                if temp[x] == temp[x - 1] and temp[x] != 0:
                    return False

            vertical_reindex = i//3*3
            horizontal_reindex = i%3*3
            temp = sorted((table[0+vertical_reindex][0+horizontal_reindex],
                           table[0+vertical_reindex][1+horizontal_reindex],
                           table[0+vertical_reindex][2+horizontal_reindex],
                           table[1+vertical_reindex][0+horizontal_reindex],
                           table[1+vertical_reindex][1+horizontal_reindex],
                           table[1+vertical_reindex][2+horizontal_reindex],
                           table[2+vertical_reindex][0+horizontal_reindex],
                           table[2+vertical_reindex][1+horizontal_reindex],
                           table[2+vertical_reindex][2+horizontal_reindex]))
            for x in range(1, 9):
                if temp[x] == temp[x - 1] and temp[x] != 0:
                    return False
        return True

    def getunsolved():
        for i in range(0,9):
            for j in range(0,9):
                if table[i][j]==0:
                    unsolved.append((i, j))

    def backtrack(depth=0):
        if depth==len(unsolved):
            return check(table)
        if check(table)==False:
            return False
        for i in range(1,10):
            table[unsolved[depth][0]][unsolved[depth][1]] = i
            if backtrack(depth+1)==True:
                return True
            table[unsolved[depth][0]][unsolved[depth][1]] = 0
        return False

    getunsolved()
    print(unsolved)
    return backtrack(), table

def read_table(table, fi):
    f = open(fi, "r")
    for i in range(0, 9):
        table.append(list(map(int, f.readline().split(' '))))

def main():
    table = []
    read_table(table, "input")
    print(table)
    sve, tbe = sudoku_solver(table)
    if sve == False:
        print("No solution")
        return None
    for row in tbe:
        print(row)

if __name__== "__main__" :
    main()
```