Question 2

Since we can only compare a nut with a bolt, we want to use a modified quicksort algorithm. This allows us to partition our nuts into a bigger and smaller pile using a single bolt and vice versa for bolts. Each iteration of our quicksort, we can just choose a random nut as our pivot point. We will then split our bolts into two piles where one pile is larger than the nut and one pile is smaller. Using the single bolt that matches the nut, we use that as the pivot and split the nuts into a smaller and larger pile. This ensures that each partitioned pile of nuts and bolts have matching pairs. The ordering does not matter at all since all we care about is the size compared to the pivot. Rather than looking through every single remaining nut and bolt, we just need to look through their individual partitioned piles. Since we are halving the pile each time, we will have logn splits and on average, each split will compare n bolts which makes our total runtime nlogn.

SAMPLE CODE

```
#!/usr/bin/python3
import random

def main():
    # TESTING

    nuts, bolts = q2(1000)
    for nut, bolt in zip(nuts, bolts):
        assert(nut.size == bolt.size)
        print(nut.size, bolt.size)

def q2(n):
    # our bolts and nuts with hidden sizes
    class Bolt:
        def __init__(self, size):
            self.size = size

    class Nut:
        def __init__(self, size):
            self.size = size

    # returns if a bolt fits a nut
    # We can only compare a nut and a bolt.
    def compare(nut, bolt):
        return nut.size - bolt.size

    # our quicksort algorithm
    def quickSort(nuts, bolts, left, right):
        if left >= right: # keep going until the subarray is empty
            return

        # since everything is shuffled, just take our pivot to be first in array
        target = bolts[left]

        # find the bolt and nut that matches at pivot and sort the rest
        pivot = move(nuts, left, right, target)
        pivot = move(bolts, left, right, target)

        # sort remaining subarrays
        quickSort(nuts, bolts, left, pivot - 1)
        quickSort(nuts, bolts, pivot + 1, right)


    # this will partition our array.
    # Everything smaller than target goes to the left
    # Everything larger than target goes to right
    # Returns the index of where our match is so we can set our new left and right
    def move(arr, left, right, target):
        idx = left
        smaller = []
        larger = []

        for num in arr[left:right + 1]:
            if compare(num, target) < 0:
                smaller.append(num)

            if compare(num, target) > 0:
                larger.append(num)

        for num in smaller:
            arr[idx] = num
            idx += 1

        pivot = idx
        arr[idx] = target
        idx += 1

        for num in larger:
            arr[idx] = num
            idx += 1

        return pivot

    ####################################################
    # initialise and randomise
    nuts = [Bolt(i + 1) for i in range(n)]
    bolts = [Nut(i + 1) for i in range(n)]
    random.shuffle(nuts)
    random.shuffle(bolts)


    quickSort(nuts, bolts, 0, n - 1)
    return nuts, bolts

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