HOMEWORK 1

COMP3121 - ALGORITHM DESIGN

QUESTION 2

PUNEETH KAMBHAMPATI

z5164647

Pseudo-Code:

```
hashmap PAIRS maps Nut-> Bolt
DEF quickPair(nuts, bolts, pairs):

    # Base case for recursion
    IF nuts and bolts only have one element left in them:
        hashmap[nuts[0]] = bolts[0]

    pivotNut = nuts[0]

    # This loop will be O(n) as we only iterate through bolts once
    FOR bolt in bolts:
        IF bolt is smaller than pivotNut:
            add bolt to smaller

        ELSE IF bolt is bigger than pivotNut:
            add bolt to bigger

        ELSE IF bolt fits pivot nut:
            pivotBolt = bolt
            hashmap[pivotNut] = pivotBolt

    # This loop will be O(n) as we only iterate through nuts once
    FOR nut in nuts:
        IF doesn't fit in pivotBolt:
            Add nut to smallerNuts
        ELSE:
            Add nut to biggerNuts


    quickPair(smallerNuts, smallerBolts, hashmap)
    quickPair(biggerNuts, biggerBolts, hashmap)

    RETURN hashmap
```

EXPLANATION:

The main idea of the solution is to sort the bolts and nuts and match the corresponding pieces together. However, we can't compare the nuts to each other or the bolts to each other and hence merge sort will not be useful in sorting a randomly divided segment of the array. Hence, we utilise quick sort in splitting the nuts and bolts into segments based on their size.

We see that a single call to this function is

O(n) + O(n) = O(2n) = O(n)

We essentially divide the array approximately log(n) number of times and each segment takes O(n) so we get,

O(logn) * O(n) = O(n*logn)

This function is also derived from the quicksort algorithm with no extra overhead so that adds to the proof that this function will run in expected time of **O(nlogn)**.

However, in the worst case (if nuts and bolts are already ordered ascending or descending) it will run in **O(n^2)**.