Dheeraj Satya Sushant Viswanadham
(z5204820)

COMP3121 Assignment 1 – Question 1

In order to determine if a number 'S' exists that satisfies the sum of the squares of two distinct numbers 'X' and 'Y' in two different ways, we first need to go through all of **[n*(n-1)]/2** possible pairs of distinct integers in array A and compute the sum of the squares which will then be stored in array B of size **[n*(n-1)]/2**. From there, we can then observe that if S pops up again from another two distinct numbers.

1A) To get an algorithm that has at most has worst case performance of n^2*log(n) we can have the below implementation:

First implement two for loops where:

loop x = 0 through to the length of array A

loop y = (x + 1) through to the length of array A

In this way, we can avoid any possible duplicate pairs as 5^2 + 11^2 = 11^2 + 5^2, regardless of the order.

We can then use a simple AVL tree (Adelson-Velsky and Landis) for insertion and searching if the sum S already exists in the tree for each iteration. These functions have worst-case time complexity of O(log(n)) and as this needs to be done each time the arithmetic sum would be n + (n - 1) + (n - 2) + … 1 = O(n^2) for the nested loops and hence, it will take **n^2 * log(n) in the worst-case scenario**.

Example Pseudocode:

```
class AVLnode {
    # … initialise …
}
class AVLtree {
    private AVLnode key;
    # insert constructor
    # code to check if tree is empty
    # code to make the tree empty (to reset it)
    # code to insert data into the tree
    # code for getting the height of the node
    # code for LH and RH nodes
    # insert data recursively
    # code to rotate the tree with either left or right children
    # code to search given number/element
    # code for inorder/preorder/postorder traversal
}
```

Dheeraj Satya Sushant Viswanadham
(z5204820)

```
testAVL = AVLtree();

# insert nested for loops e.g. as below:

for x, X in array:
        for y, Y in array[x + 1]:
                sum = (X * X) + (Y * Y);
                # check if the sum already exists in the AVL tree
                if testAVL.contains(sum) already then:
                        return true;
                # otherwise proceed with inserting the value into the AVL tree
                testAVL.insert(sum);
return false;
# End
```

1B) We will follow the same approach as given in (a), however, we will now instead use a hash table or 'hash map' instead of an AVL tree as the hash table is a more efficient data structure operation in terms of average time complexity as each insertion and lookup takes O(1) expected time as compared to the AVL tree's slower O(log(n)) expected time. Since we still have the nested loops to get our values, and which have a time complexity of O(n^2), then the **final expected time will evidently be O(n^2) * O(1) = O(n^2).**

Example Pseudocode:

```
# create HashMap using java.util* library for example.

HashMap <Integer, String> hm = new HashMap<Integer, String>

for x, X in array:
        for y, Y in array[x + 1]:
                sum = (X * X) + (Y * Y);
                # check if the sum already exists in the HashMap
                if sum in hm already then:
                        return true;
                # otherwise proceed with inserting the value into the HashMap
                hm[sum] = (X, Y)
return false;
# End
```