Question 1.

The idea for both parts is to use two for loops and compute the square sum of every possible pair. We then want to store these sums each iteration. If later we produce a sum that we have previously made, this means that we have found two different pairs that make this sum. Our second loop will be i = 0..len(A) and our second loop will be between j = i + 1 .. len(A) to avoid counting the same pair twice. Since the numbers in A are distinct, we can be sure that we won't have duplicate pairs. This is done in o(n^2) time.

<u>PART A</u>
For n^2 * logn runtime, our method of storage will be to use a binary search tree. Each iteration we must perform an insertion and compute whether a sum is already in the tree. Both functions run in logn time and we do this n^2 times.

<u>PART B</u>
For n^2 runtime, we will use a hashmap rather than a binary search tree. Similarly, each iteration we perform and insertion and contains. Both these functions run in constant time and we do this n^2 times.

SAMPLE CODE

```
#!/usr/bin/python3
def main():
	# TESTING

	# 1 + 64 = 16 + 49
	arr = [1,8,4,7]
	print(q1a(arr))
	print(q1b(arr))

	# 0 .. 7 has no such pairs
	arr = [i + 1 for i in range(7)]
	print(q1a(arr))
	print(q1b(arr))


def q1a(arr):
	class Node: # bst node
		def __init__(self, val):
			self.val = val
			self.left = None
			self.right = None

	class BinarySearchTree: # bst data structure
		def __init__(self):
			self.head = None

		# insert value into bst
		def insert(self, val):
			if self.head == None:
				self.head = Node(val)
				return

			node = self.head

			while True:
				if node.val == val:
					return

				if node.val > val:
					nextNode = node.right
				else:
					nextNode = node.left

				if nextNode == None:
					break

				node = nextNode

			if node.val > val:
				node.right = Node(val)
			else:
				node.left = Node(val)
```

```
        # if bst contains value
        def contains(self, val):
            def recurse(node, val):
                if not node:
                    return False

                if node.val == val:
                    return True

                return recurse(node.left, val) or recurse(node.right, val)

            return recurse(self.head, val)


        def showTree(self):
            def recurse(node):
                if not node:
                    return

                print(node.val)

                recurse(node.left)
                recurse(node.right)

            recurse(self.head)

    # binary search tree implementation to track visited. Insertion and contains done in logn
time
    bst = BinarySearchTree()

    for i, num1 in enumerate(arr):
        for j, num2 in enumerate(arr[i + 1:]):
            total = num1*num1 + num2*num2

            if bst.contains(total): # if we have previously made the sum, we have satisfied
condition
                return True

            bst.insert(total) # insert into bst

    return False


def q1b(arr):
    seen = {} # uses a hashmap

    # n^2 loop through array
    for i, num1 in enumerate(arr):
        for j, num2 in enumerate(arr[i + 1:]):
            total = num1*num1 + num2*num2

            if total in seen: # if we have previously made the sum, we have satisfied
condition
                return True

            seen[total] = (num1, num2) # add to hashmap

    return False


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