**COMP1201 Assignment 2**

Q1:

*What we are given:*

* Hash function with separate chaining.
* Takes an integer **(n < 100 000)**.
* Calculates the value as **(2d1 + 3d2 + 5d3 + 7d4 + 11d5) % 47**, where d1 is the most significant digit and d5 is the least significant digit.
* We enter **2000** numbers.

*What we need to prove:*

* Prove that there exists a number **x** **(0 <= x <= 46)** for which we can find **at least 43** numbers among the **given** **2000**, whose **hash** **value** is exactly **x**.

*Proof:*

First, when a hash table has separate chaining, it mean that we build a singly-linked list at each table entry. If the keys for two elements are the same, then both of the elements are added to the corresponding singly-linked list.

The given problem can be easily coded using Java:

//Create the new HashMap, which will be implemented using separate chaining.  
HashMap<Integer, ArrayList<Integer>> myMap = new HashMap<>();  
Random rand = new Random();  
  
int key = 0;  
int value = 0;  
  
//Add 2000 numbers.  
for (int i = 0; i < 2000; i++) {  
 //Generate the 2000 random numbers.  
 value = rand.nextInt(100000);  
  
 //Isolate the different digits, in order to calculate the hashing value.  
 int firstDigit = (value / 10000);  
 int secondDigit = (value / 1000) % 10;  
 int thirdDigit = (value / 100) % 10;  
 int fourthDigit = (value / 10) % 10;  
 int fifthDigit = (value % 10);  
  
 //Add the key and value to the map.  
 key = (2 \* firstDigit +  
 3 \* secondDigit +  
 5 \* thirdDigit +  
 7 \* fourthDigit +  
 11 \* fifthDigit) % 47;  
  
 //Lambda expression to add the integer to its specified ArrayList.  
 myMap.computeIfAbsent(key, k -> new ArrayList<>()).add(value);  
}  
  
//Iterate through the map to see the values.  
int i = 1;  
Integer lastKey = 0;  
ArrayList<Integer> numberOfValues= new ArrayList<>();  
  
for (java.util.Map.Entry<Integer, ArrayList<Integer>> entry : myMap.entrySet()) {  
 Integer myKey = entry.getKey();  
 ArrayList<Integer> myValues = entry.getValue();  
 for (Integer singleValue : myValues) {  
 //Print out everything.  
 //The counter is there so we can easily see the number of elements under each key.  
 if(lastKey == myKey) {  
 //System.out.println("key : " + myKey + " value : " + singleValue + " number : " + i);  
 i++;  
 }  
 else {  
 i = 1;  
 //System.out.println("key : " + myKey + " value : " + singleValue + " number : " + i);  
 lastKey = myKey;  
 i++;  
 }  
 }  
 //System.out.println("Key: " + myKey + " <-> " + "Number of values: " + i);  
 numberOfValues.add(i);  
}  
  
//Display the amount of values under each key, sorted, so we can prove that  
//that there exists a number x between 0 and 46 for which we can find at least 43  
//numbers among the given 2000, whose hash value is exactly x.  
Collections.*sort*(numberOfValues, Collections.*reverseOrder*());  
for(Integer number : numberOfValues) {  
 System.*out*.println(number);  
}

If we run this code, we can see that there is at least one number, **x**, between 0 and 46, for which there are at least 43 numbers, from the 2000, whose hash values is **x**.

Now, if we try to prove this mathematically, we can use the *pigeonhole principle*.

First, given the *hash code*, and the given *maximum value* we can have, the *values* of the *keys* may vary from 0 (included) to 46 (included). Since we input 2000 numbers (pigeons), and we have 47 key slots (pigeonholes), we see that there will be at least one key (pigeonhole) with at a minimum of 43 values (pigeons) under it. We can see that by dividing the pigeons by the pigeonholes (2000 / 47), and we get 42,5531914893617‬. When we the ceiling function (as per pigeonhole principle), we can see that (**⌈**42,5531914893617**⌉** = 43)there is a pigeonhole with at least 43 pigeons in it.