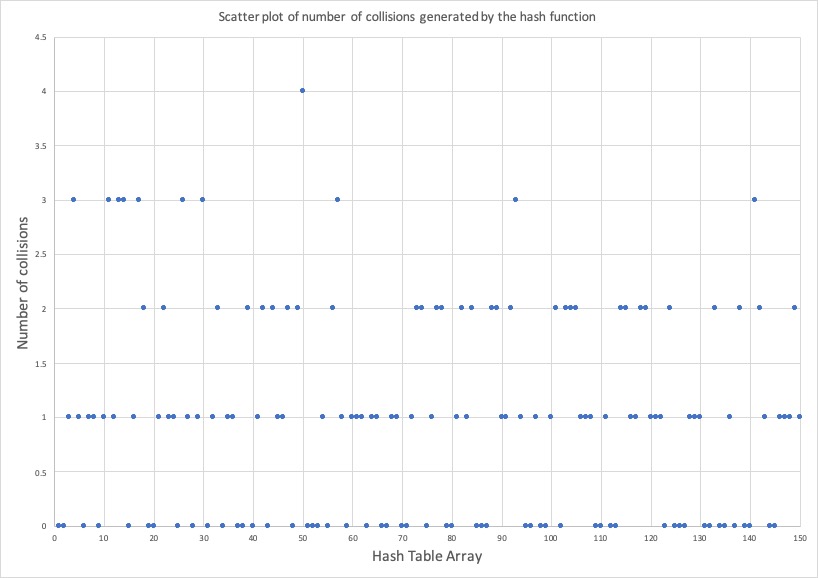
# Part 2 ­ Hash Function Analysis

1. Analyzing the hash table containing the pointers to the all the student objects in the institution object
2. The indexing key used is the student number
3. Student number was chosen because every student will have a unique student number

## Results:

### What does the range of hash table array indices produced by our hash function look like?



### Can we predict where all the other indices (not produced in our experiment) would "land"?

### Does our hash function evenly distribute indexing keys across the range of hash table array indices? Do we obtain a 1­to­1 mapping?

The hash function does a mediocre job distributing the indexing keys across the range of the hash table array. There are 54 empty indices in the hash table array implying that there is no 1-to-1 mapping. The hash function produces a maximum of 4 collisions per index. From the scatter plot above, it is clear that the indexing keys are distributed evenly and the majority of the keys are created only once

### Is our hash function producing collisions? Why?

Yes, the hash function is producing collisions. This is happening because arithmetic on the student number may not be unique. Since we are using a folding technique, the arithmetic can be identical to a student number is that looks different, but because of the arithmetic the hash function performs on it, it produces the same hash code.

### What do synonyms look like? Give some examples of synonyms.

### How easy each version is to compute, by expressing its time efficient using the Big O notation.

The hash function is very easy to compute. It can be computed in O(1) because we are using folding hash functions that does simple arithmetic on parts of the indexing key.