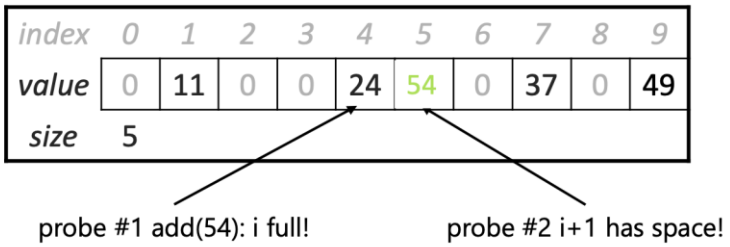
Lab 08

Q1 (Hashing): Hash Functions [6 marks]

Draw the tables (three separate tables for a, b, and c) that result after inserting the following values in the given order: 17, 25, 11, 46, 14 to a table of size N.

Handle collisions using **linear probing (Check lecture15 Slide 12)**

```
set.add(11);        //abs(11) % 10 == 1
set.add(49);        //abs(49) % 10 == 9
set.add(24);        //abs(24) % 10 == 4
set.add(37);        //abs(37) % 10 == 7
set.add(54);        //abs(54) % 10 == 4 collides with 24!
```

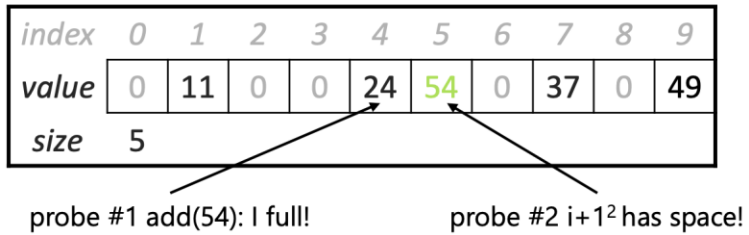


Q1 (Hashing): Hash Functions [6 marks]

Draw the tables (three separate tables for a, b, and c) that result after inserting the following values in the given order: 17, 25, 11, 46, 14 to a table of size N.

Handle collisions using **quadratic probing (Check lecture15 Slide 13)**

```
set.add(11);
set.add(49);
set.add(24);
set.add(37);
set.add(54);  // collides with 24; must probe
```

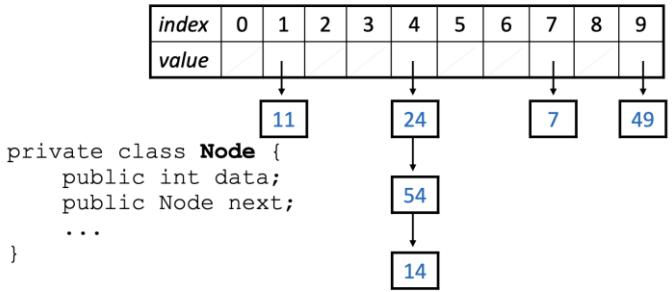


Q1 (Hashing): Hash Functions [6 marks]

Draw the tables (three separate tables for a, b, and c) that result after inserting the following values in the given order: 17, 25, 11, 46, 14 to a table of size N.

Handle collisions using **Separate Chaining (Check lecture15 Slide 28)**

- **separate chaining**: Solving collisions by storing a list at each index.
  - add/contains/remove must traverse lists, but the lists are short
  - impossible to "run out" of indexes, unlike with probing



Q2 (Hashing): Collision Handling

`linearProbing()`

If the hash function indicates that a value should be stored at an already full (non-zero) index of the array

- this function increments the index until an available spot is found
- then inserts the new element at that index.

Check lecture15 – slides 16-29

Q2 (Hashing): Collision Handling

`quadraticProbing()`

This method is similar to linear probing, except instead of incrementing the index by one until an available spot is found, this method increments by a quadratically increasing amount.

- If the initial index returned by the hash function is i, the next index tried is i+1^2, then i+2^2, and so on until an available spot is found.

Check lecture15 – slides 16-29

Q2 (Hashing): Collision Handling

`doubleHashing()`

This final method is slightly more complicated. A secondary hash function is calculated in the case of a collision.
This function often is in the form of d(k) = q – k % q for a given key k, when q is a prime number.

Check lecture15 – slides 16-29

Q3 (Sorting): Comparing performance [4 marks]

- The code in this file is **complete**, but you should examine it anyway.
- An Excel spreadsheet is provided for you: **Results.xlsx**

- Your task is to run the file at least three times and collect the runtime data
  - The average of the trials for each method at each array size should be automatically calculated
- Examine the graph you've created and see if you can guess the Big-O complexities (either O(n^2) and O(n log(n))) of each of these sorting algorithms.
  - Note your hypothesis in the Excel spreadsheet.
  - For full marks, make a text comment in the Excel spreadsheet below your hypothesis, explaining your thinking

Q4 (Minimum Spanning Tree): [6 marks (BONUS)]

`printList()`: This method prints the graph's adjacency list.

`getMST()`: This method creates and returns a new MSTGraph which represents the minimum spanning tree of the original graph