Andrew Ha
COMP 352

Class Specifications

Class	Abstract	Implements Abstract	Methods	Method Description
HashTable	Yes	No	HashTable Input – int N Size to initialize array to	Constructor, for the child classes. Takes in a number to create the hash table size
			Size Input: None Output: Size of hash table	Returns the current size (number of elements) in the hash table
			Get Input K: Key is to be searched for in hash table Output: MapElement object if a matching key is found. Null otherwise	Abstract method for other child classes to inherit. Given a key it will attempt to look for the element
			Get Input K: Key is to be inputted for in hash table Output: MapElement object if a matching key is found. Null otherwise	Abstract method for other child classes to inherit. Will attempt to place an element given a key

			Remove Input K: Key to be searched for in the table Output: String Value of element that is be removed	Abstract Method for other child classes to inherit. Will attempt to remove an element
HashLinear	No	Yes	HashLinear Input – int N Size to initialize array to	Constructor, HashLinear class. Calls the super constructer of HashTable
			Size Input: None Output: Size of hash table	Returns the current size (number of elements) in the hash table. Called from HashTable
			Get Input K: Key is to be searched for in hash table Output: MapElement object if a matching key is found. Null otherwise	Given a key it will attempt to look for the element. Implements linear probing
			Get Input K: Key is to be inputted for in hash table Output: MapElement object if a matching key is found. Null otherwise	Will attempt to place an element given a key Implements linear probing
			Remove Input K: Key to be	Will attempt to remove an

			searched for in the table Output: String Value of element that is be removed	Element. Implements linear probing
HashQuadratic	No	Yes	HashQuadratic Input – int N Size to initialize array to	Constructor, HashLinear class. Calls the super constructer of HashTable
			Size Input: None Output: Size of hash table	Returns the current size (number of elements) in the hash table. Called from HashTable
			Get Input K: Key is to be searched for in hash table Output: MapElement object if a matching key is found. Null otherwise	Given a key it will attempt to look for the element. Implements Quadratic Probing
			Get Input K: Key is to be inputted for in hash table Output: MapElement object if a matching key is found. Null otherwise	Will attempt to place an element given a key Implements Quadratic Probing
			Remove Input K: Key to be searched for in the table	Will attempt to remove an Element.

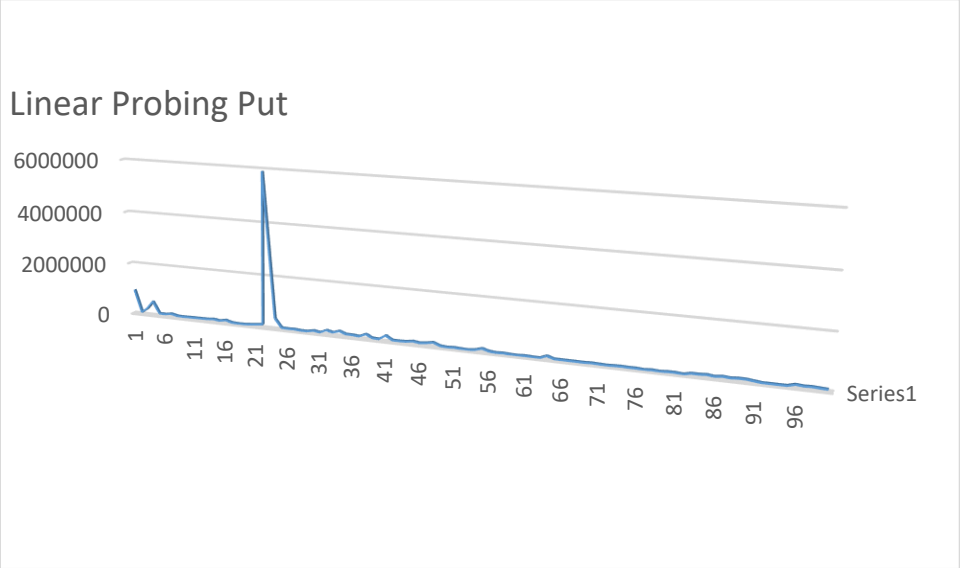
			Output: String Value of element that is be removed	Implements Quadratic Probing
HashChaining	No	Yes	HashQuadratic Input – int N Size to initialize array to	Constructor, HashLinear class. Calls the super constructer of HashTable
			Size Input: None Output: Size of hash table	Returns the current size (number of elements) in the hash table. Called from HashTable
			Get Input K: Key is to be searched for in hash table Output: MapElement object if a matching key is found. Null otherwise	Given a key it will attempt to look for the element. Implements Hash Chaining
			Get Input K: Key is to be inputted for in hash table Output: MapElement object if a matching key is found. Null otherwise	Will attempt to place an element given a key Implements Hash Chaining
			Remove Input K: Key to be searched for in the table Output: String	Will attempt to remove an Element. Implements Hash Chaining

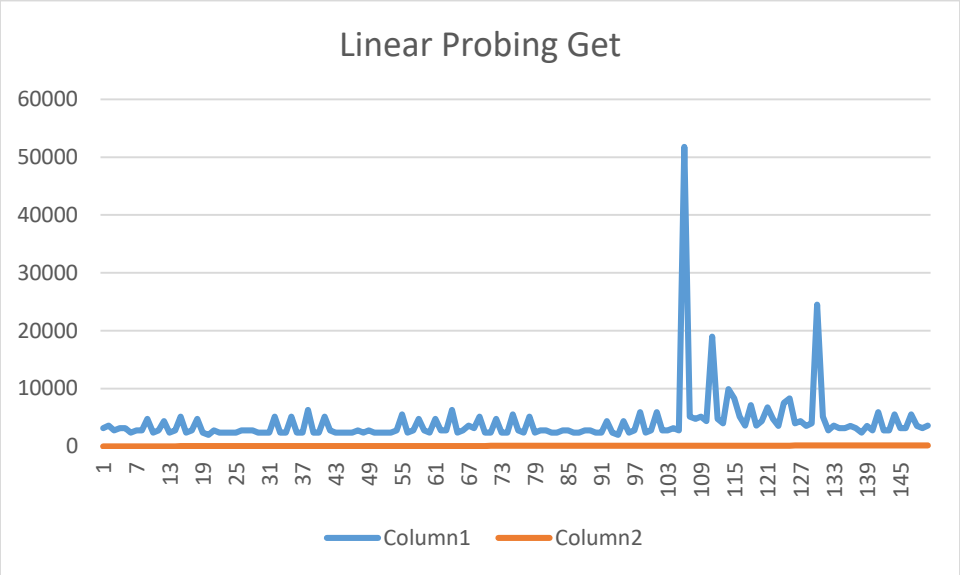
			Value of element that is be removed	
MapElement	No	No	MapElement	Constructor. Randomly generates a key. And sets its value to a string involving the key
MapElement	No	No	HashCode	Overrides HashCode. Using Horners rule to implement a hash code based on its key

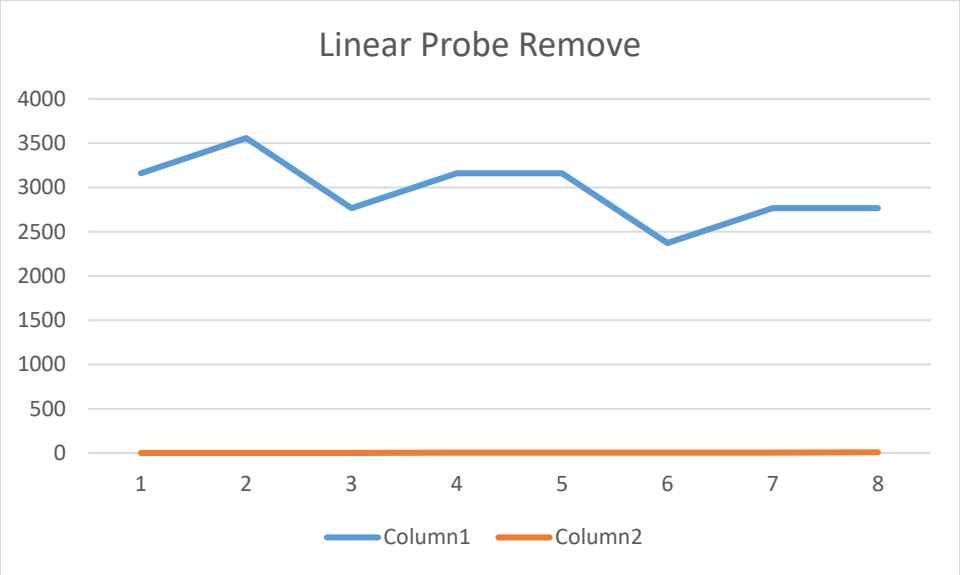
Design Decisions

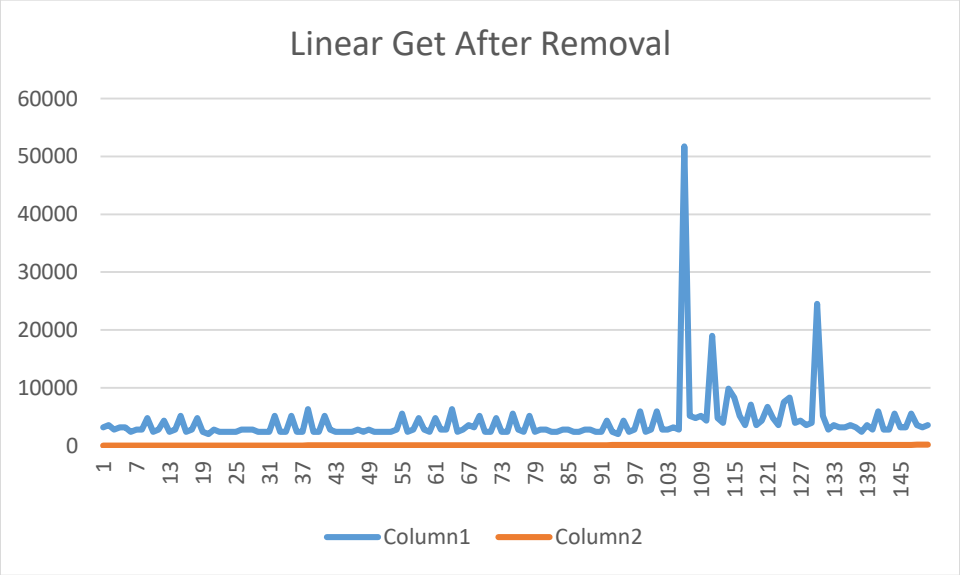
I tried to make my best to make the classes abstract. At first my design isn't the most modular unfortunately, and given the chance, I should've made methods to look for the index in another method, and just reused more methods. Nonetheless, common functionality such as Size and Contructor were kept in the Abstract class as well as the class variables. There were quite a bit of duplicate code between the classes due to lack of time, and not as much foresight into the assignment. I decided to give the linear/quadratic probing and searches approximately the size of N x 5 times before it doesn't have enough time.

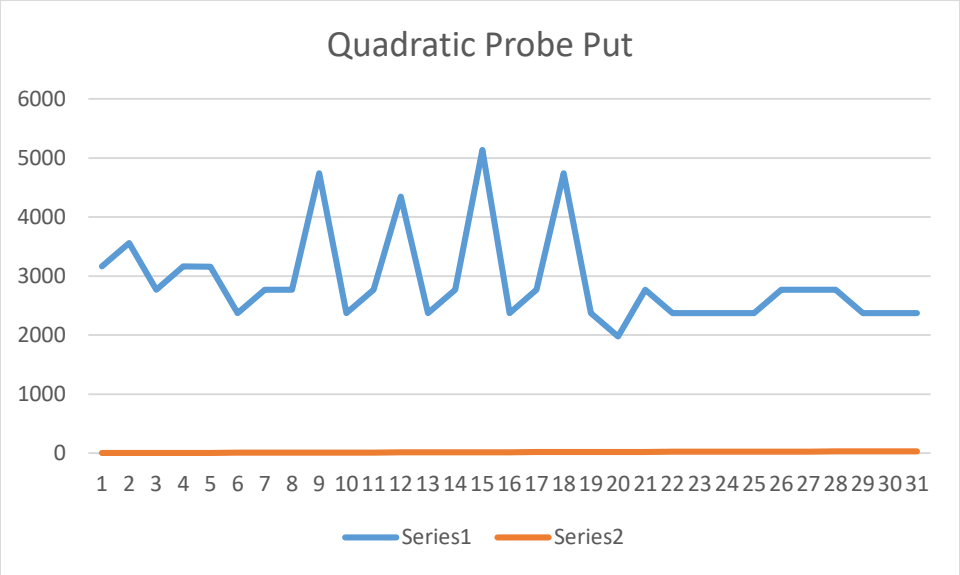
Report Times:

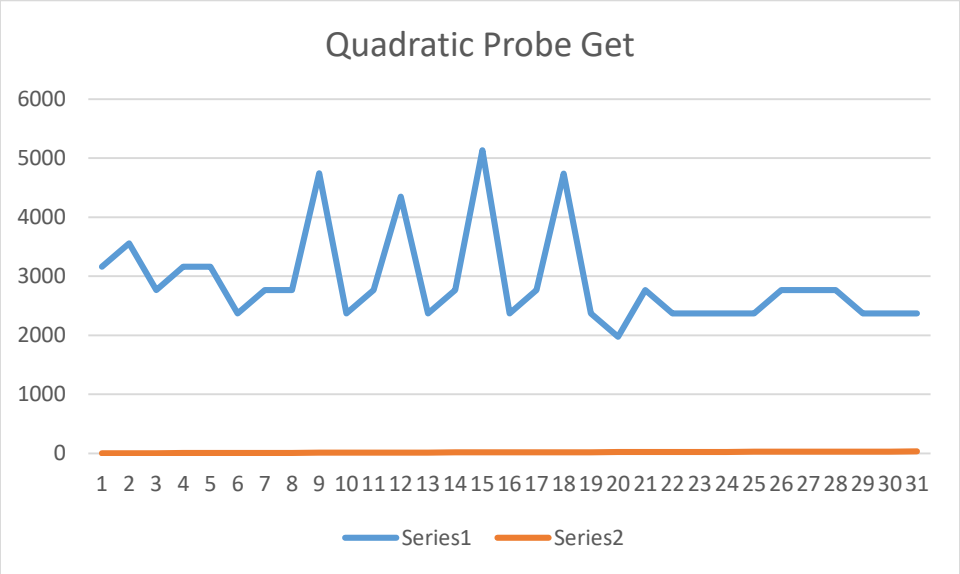


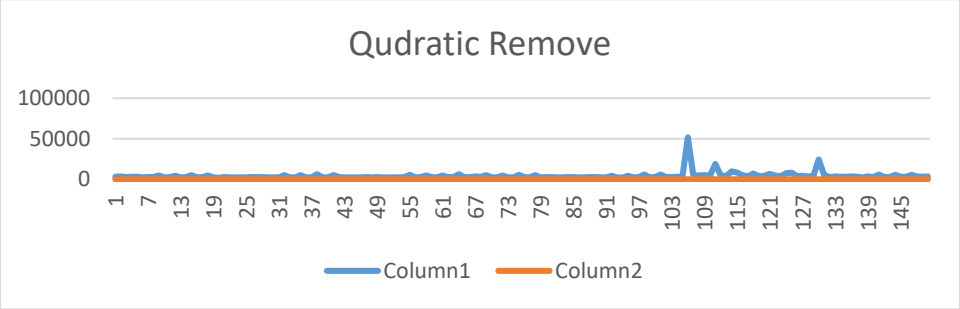


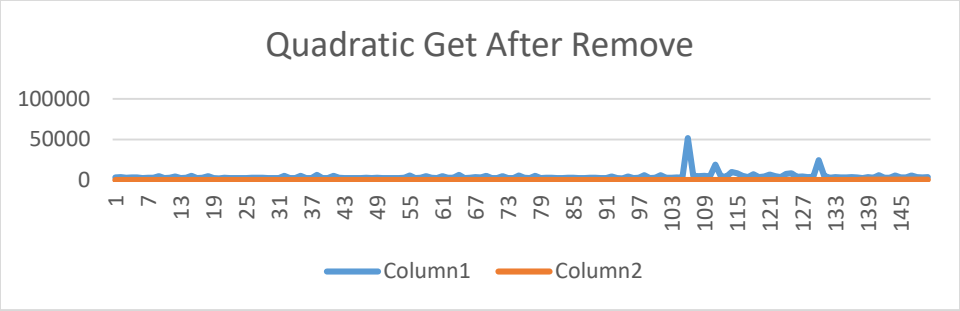

Linear Probe Remove
4000
3500
3000
2500
2000
1500
1000
500
0
1
2
3
4
5
6
7
8
Column1
Column2

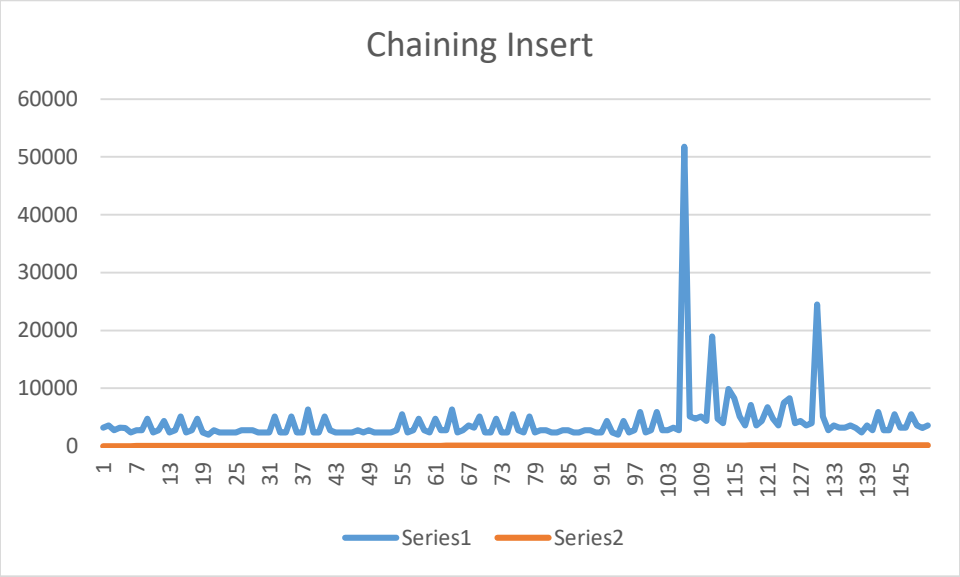

Linear Get After Removal
60000
50000
40000
30000
20000
10000
0
1
7
13
19
25
31
37
43
49
55
61
67
73
79
85
91
97
103
109
115
121
127
133
139
145
Column1
Column2

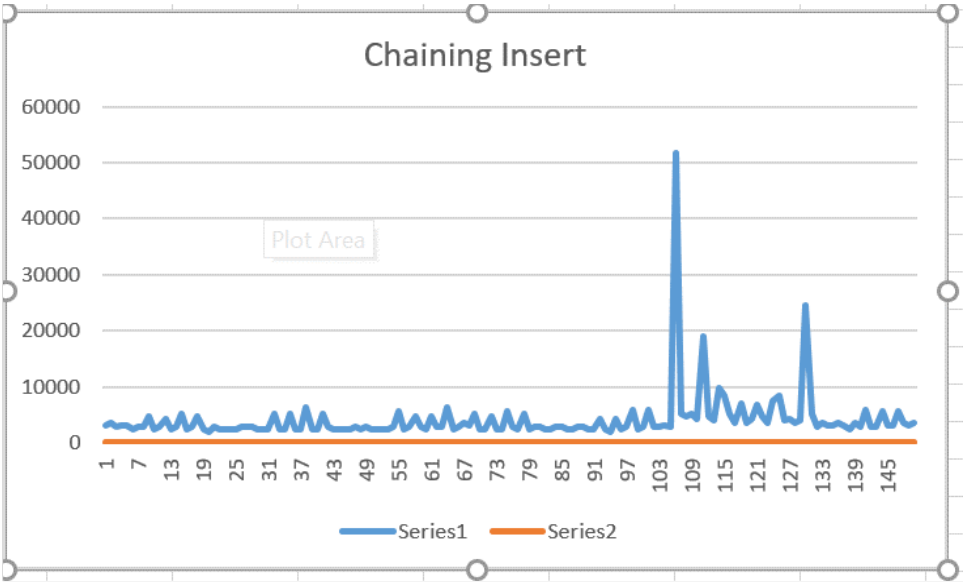

Quadratic Probe Put
6000
5000
4000
3000
2000
1000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Series1
Series2

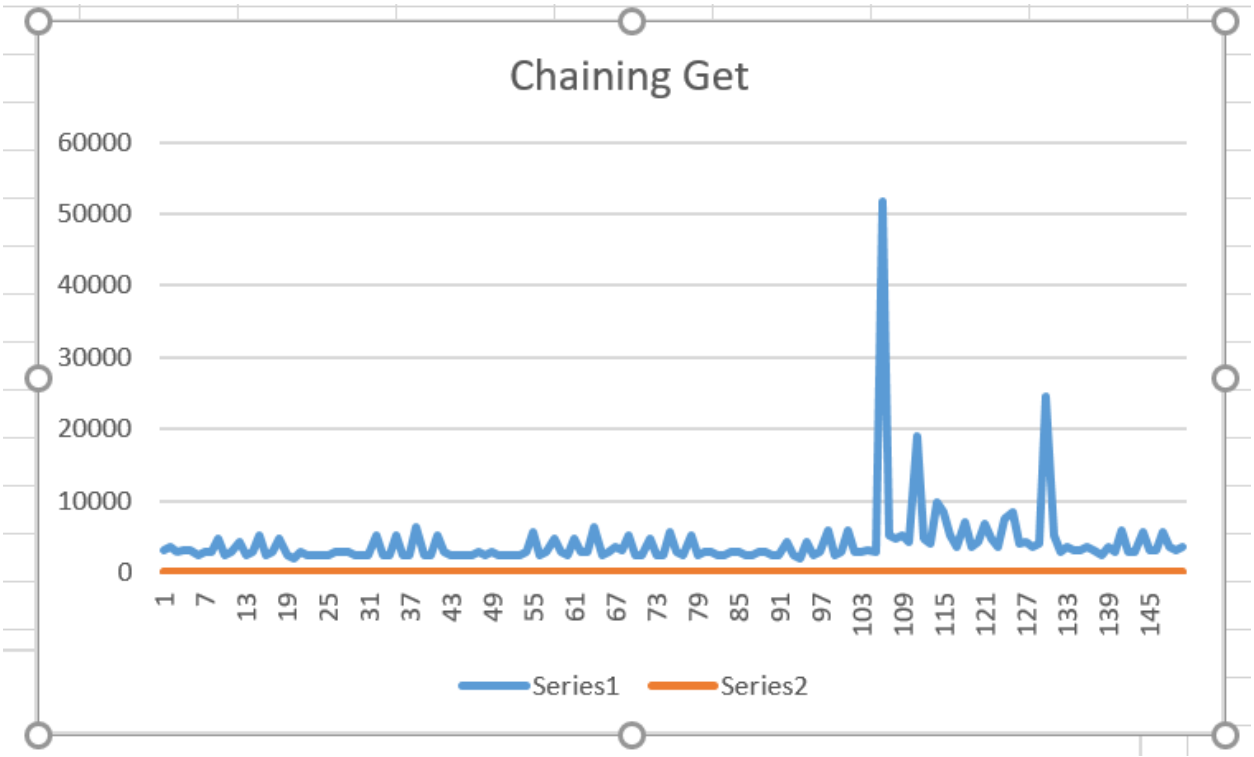

Quadratic Probe Get
6000
5000
4000
3000
2000
1000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Series1
Series2


Qudratic Remove
100000
50000
0
1 7 13 19 25 31 37 43 49 55 61 67 73 79 85 91 97 103 109 115 121 127 133 139 145
Column1
Column2


Quadratic Get After Remove
100000
50000
0
1 7 13 19 25 31 37 43 49 55 61 67 73 79 85 91 97 103 109 115 121 127 133 139 145
Column1
Column2


Chaining Insert
60000
50000
40000
30000
20000
10000
0
1 7 13 19 25 31 37 43 49 55 61 67 73 79 85 91 97 103 109 115 121 127 133 139 145
Series1
Series2


Chaining Insert
60000
50000
40000
30000
20000
10000
0
Plot Area
1 7 13 19 25 31 37 43 49 55 61 67 73 79 85 91 97 103 109 115 121 127 133 139 145
Series1
Series2


Chaining Get
60000
50000
40000
30000
20000
10000
0
1
7
13
19
25
31
37
43
49
55
61
67
73
79
85
91
97
103
109
115
121
127
133
139
145
Series1
Series2

Analysis

The graphs all appear to increase in size with each insert. This is because of more and more collisions happening. The linear probing takes the longest, quadratic second longest and the one with buckets is a decent speed.

The reason why some of these methods take an excessive time is due to the hash table running out of room and having to probe. It happens because the quadratic probing might take a long time to look for a location once it ran out of room.

Rerunning the experiment for quadratic probing reveals less time. This is because 101 is a prime number therefore the chances of collisions is reduced.

Benchmarking the cost of dynamic resizing., the cost of a put and a get is at least 2-3x more expensive when we are dynamically resizing the table. I got a lot more exceptions when I was trying to probe with quadratic probing as well.