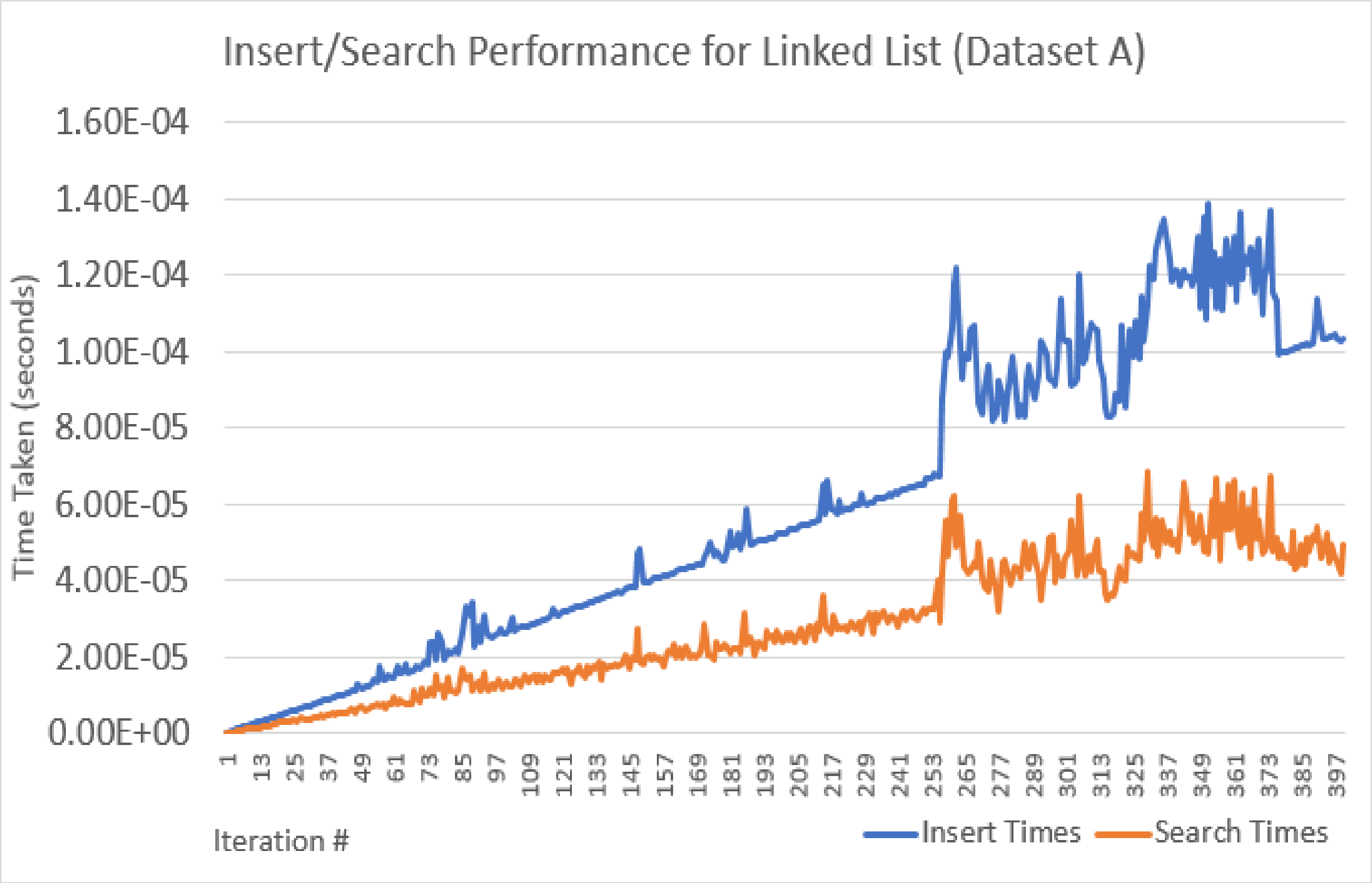
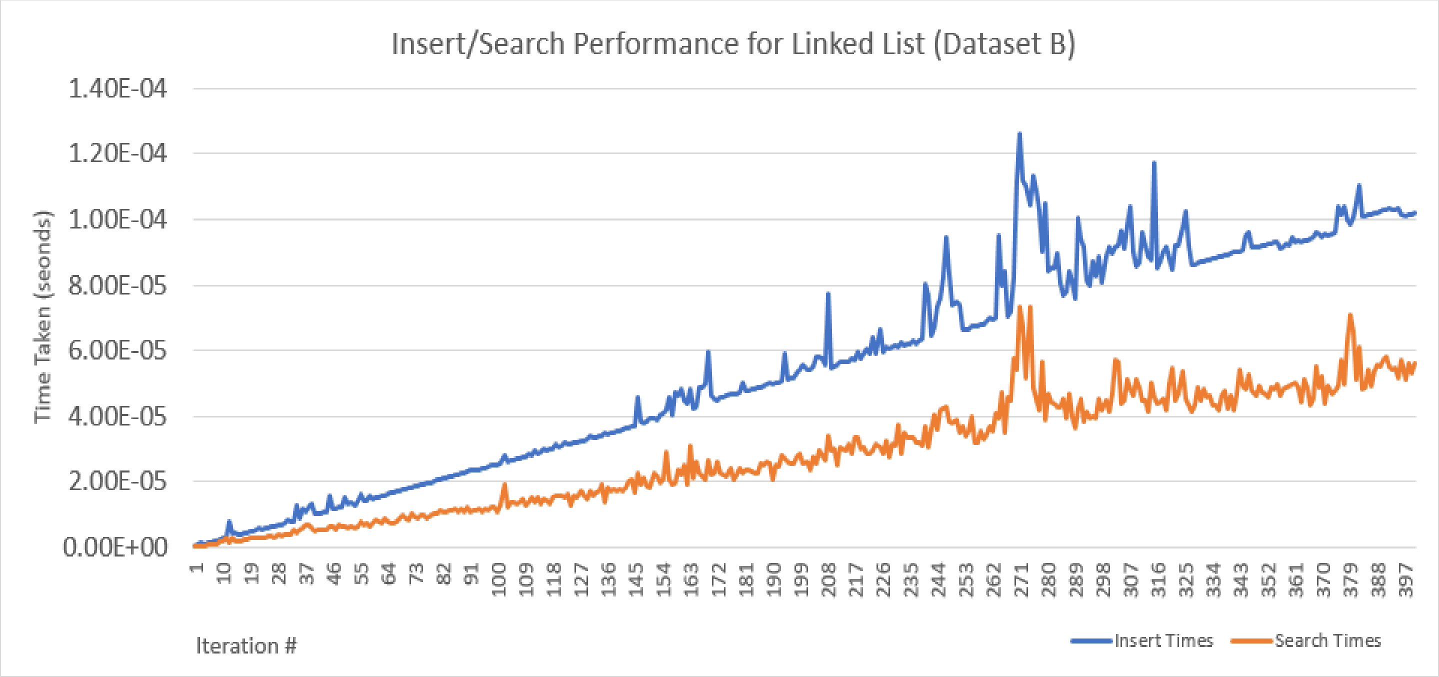
A Performance Analysis of Differing Data Structures

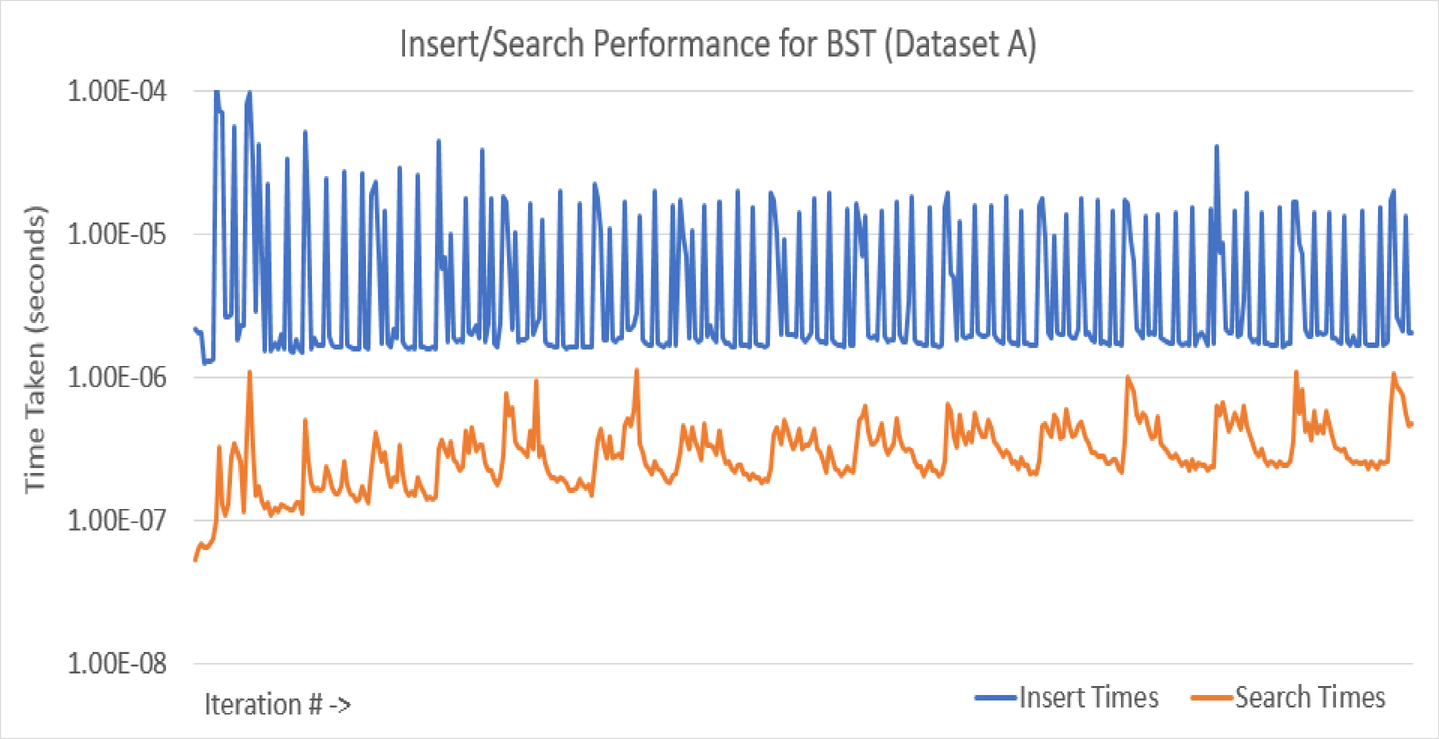
Vincent Curran

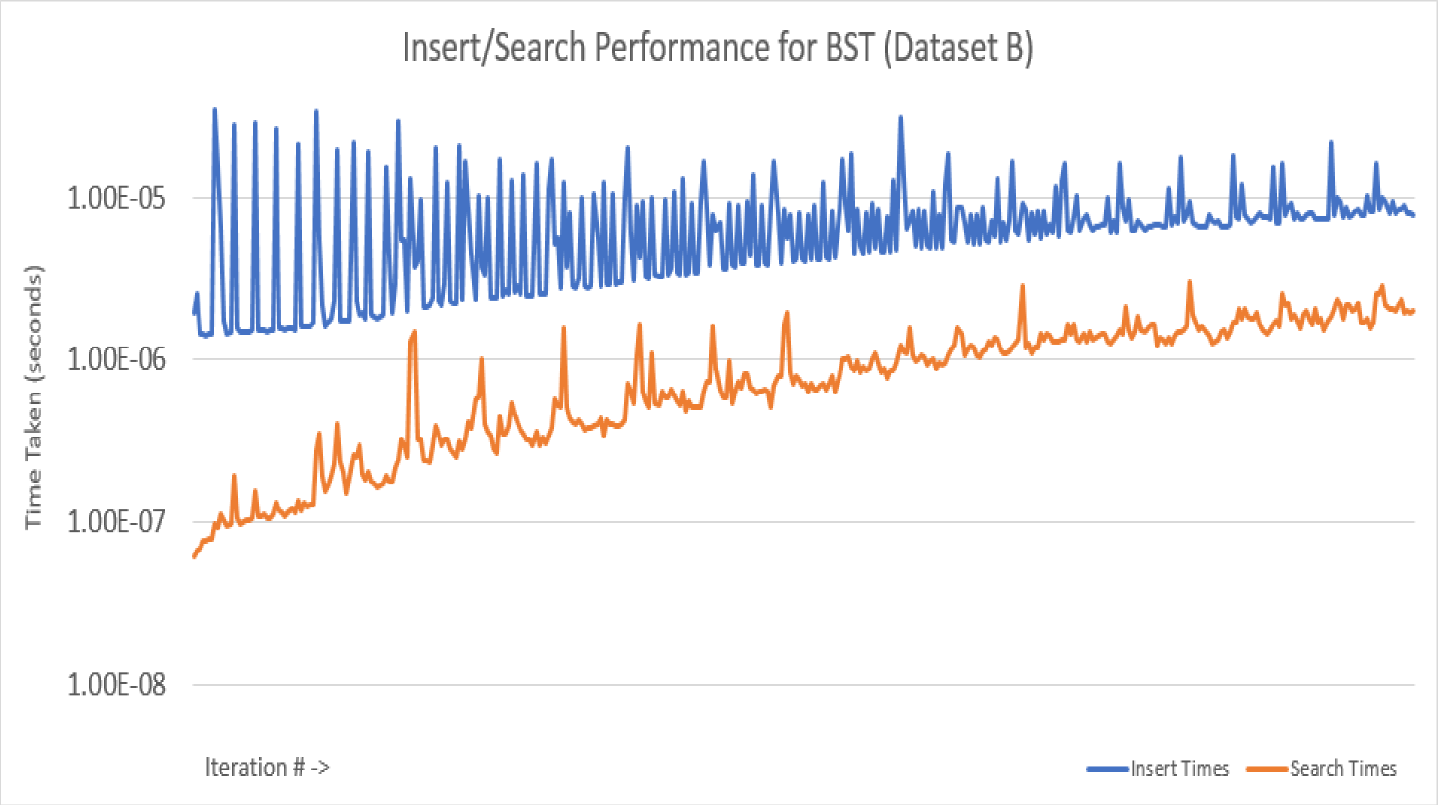
CSCI 2270-104

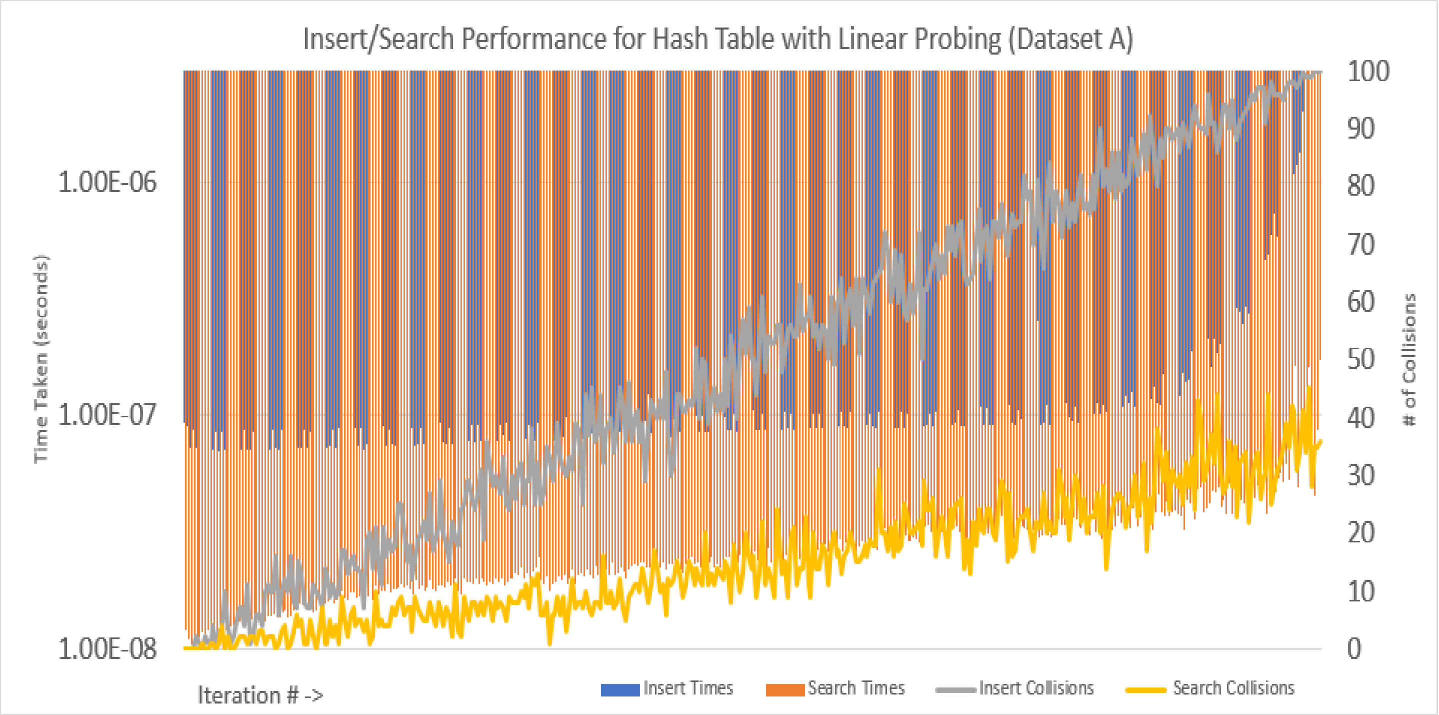
April 27th, 2020

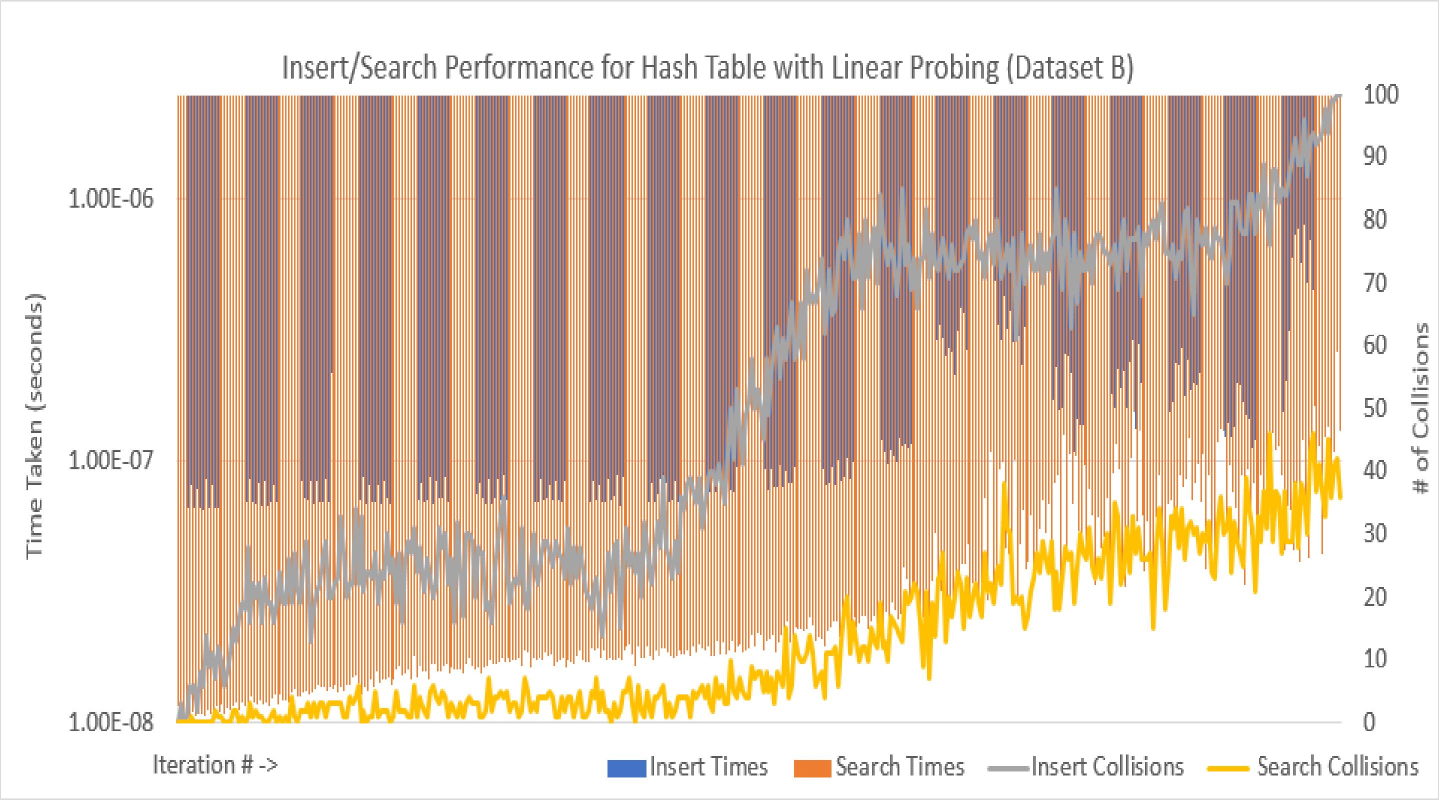

Insert/Search Performance for Linked List (Dataset A)
Time Taken (seconds)
1.60E-04
1.40E-04
1.20E-04
1.00E-04
8.00E-05
6.00E-05
4.00E-05
2.00E-05
0.00E+00
1
13
25
37
49
61
73
85
97
109
121
133
145
157
169
181
193
205
217
229
241
253
265
277
289
301
313
325
337
349
361
373
385
397
Iteration #
Insert Times
Search Times

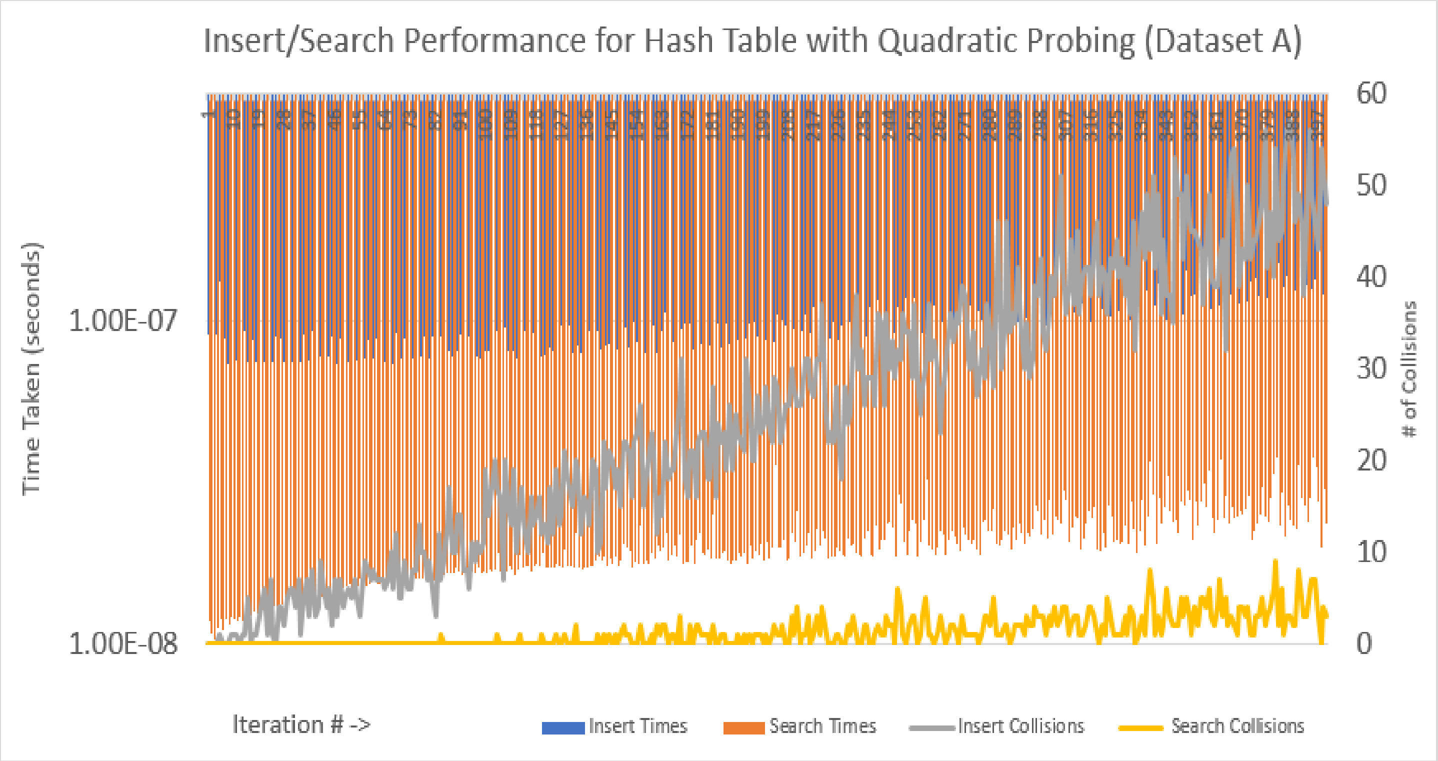

Insert/Search Performance for Linked List (Dataset B)
Time Taken (seonds)
1.40E-04
1.20E-04
1.00E-04
8.00E-05
6.00E-05
4.00E-05
2.00E-05
0.00E+00
1
10
19
28
37
46
55
64
73
82
91
100
109
118
127
136
145
154
163
172
181
190
199
208
217
226
235
244
253
262
271
280
289
298
307
316
325
334
343
352
361
370
379
388
397
Iteration #
Insert Times
Search Times

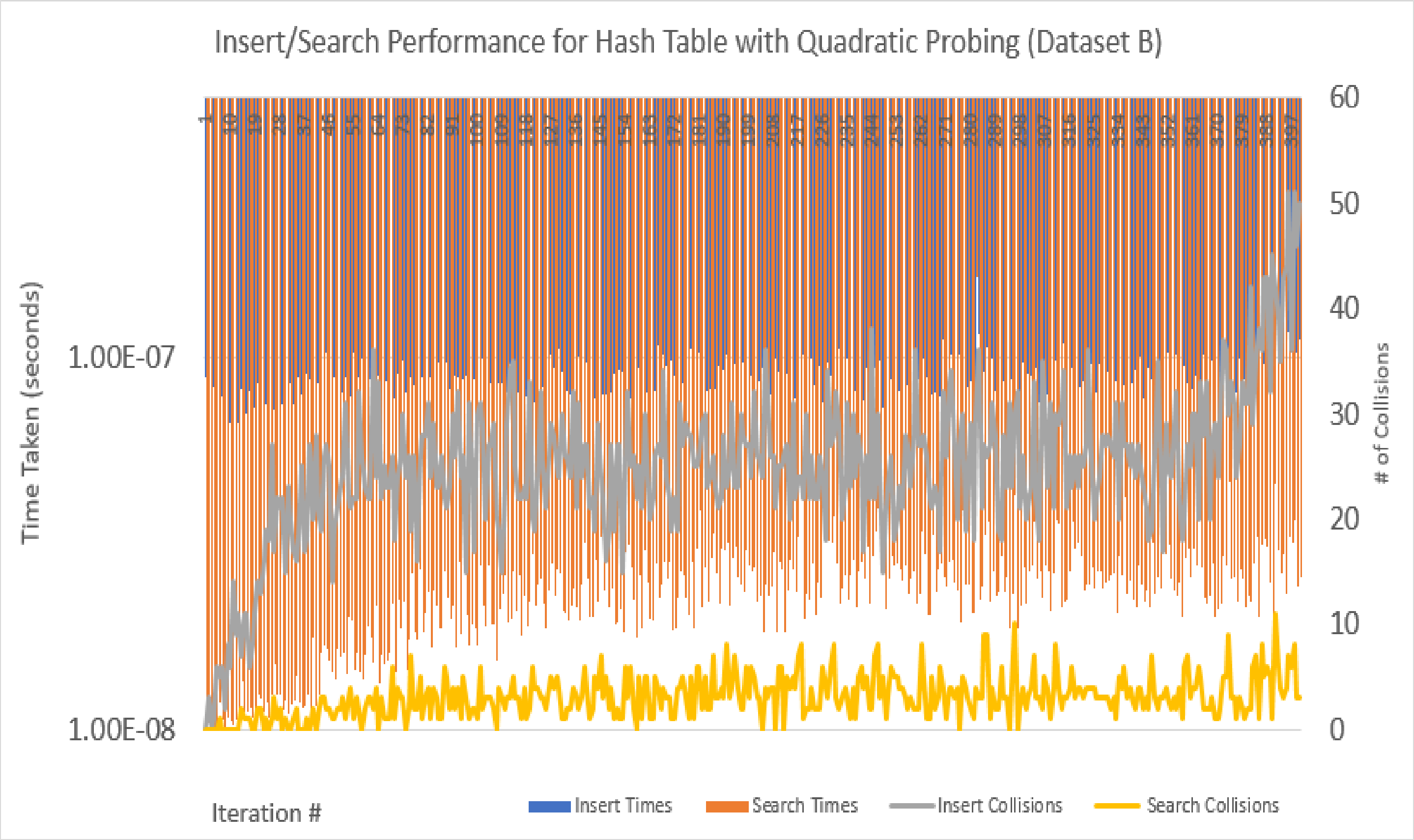

Insert/Search Performance for BST (Dataset A)
Time Taken (seconds)
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Iteration # ->
Insert Times
Search Times

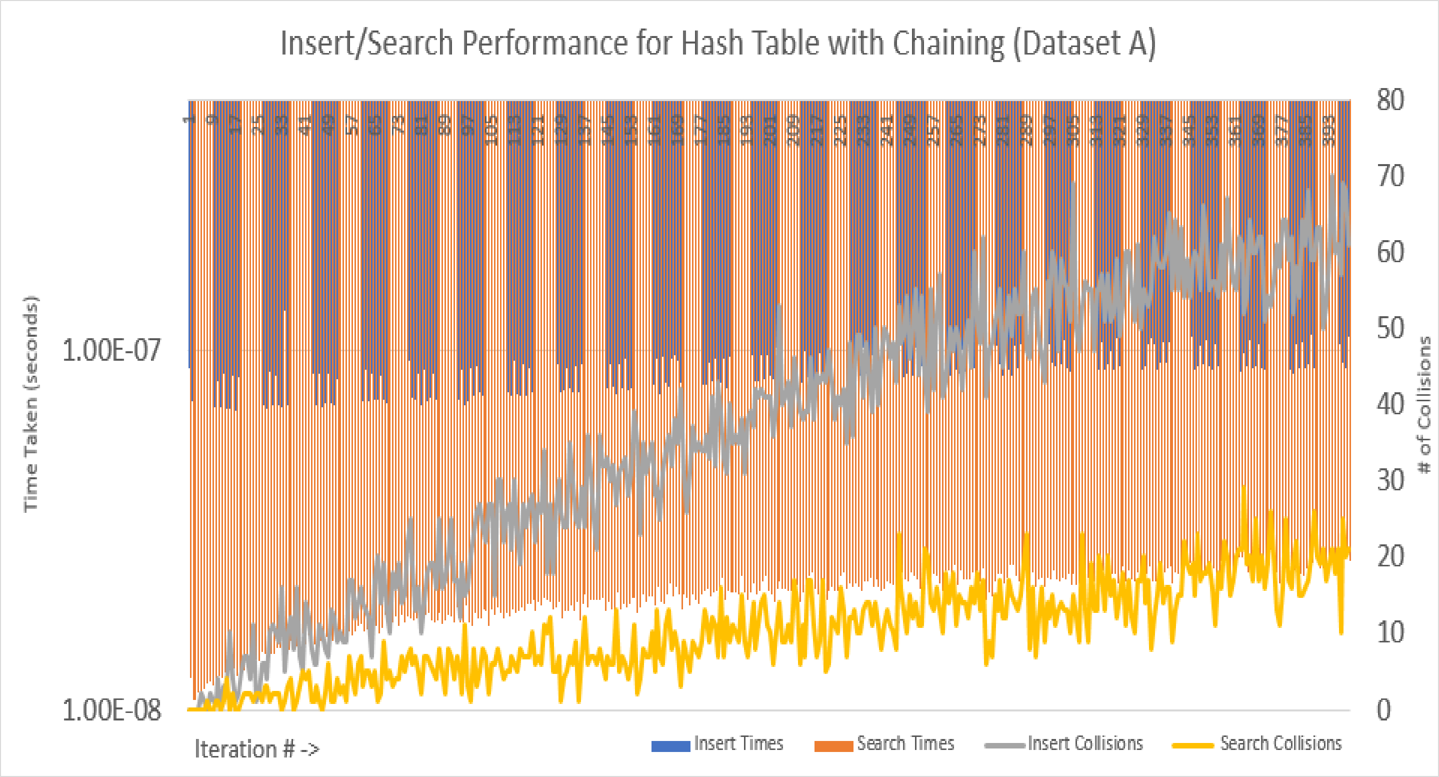

Insert/Search Performance for BST (Dataset B)
Time Taken (seconds)
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Iteration # ->
Insert Times
Search Times

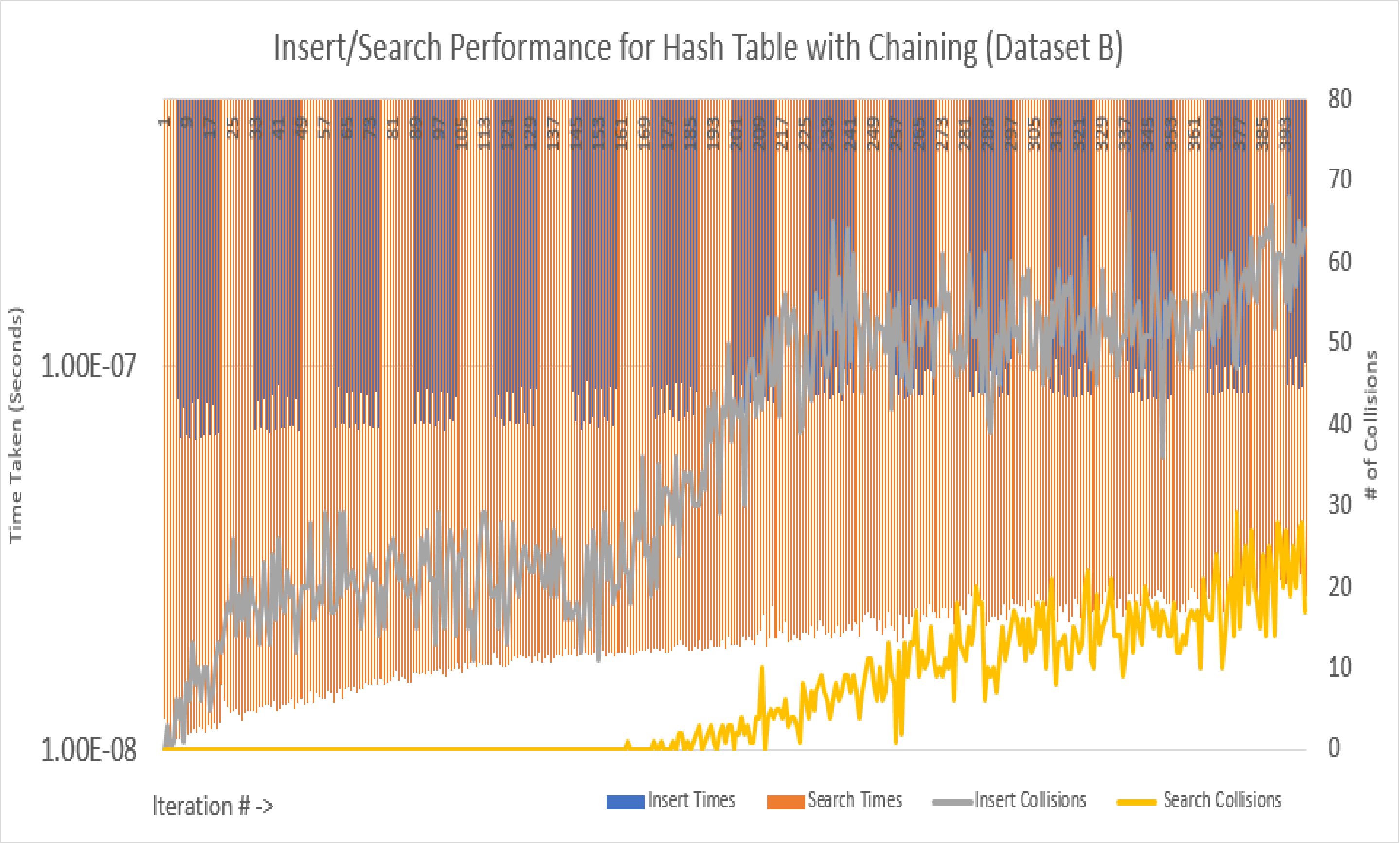

Insert/Search Performance for Hash Table with Linear Probing (Dataset A)
Time Taken (seconds)
1.00E-06
1.00E-07
1.00E-08
# of Collisions
100
90
80
70
60
50
40
30
20
10
0
Iteration # ->
Insert Times
Search Times
Insert Collisions
Search Collisions

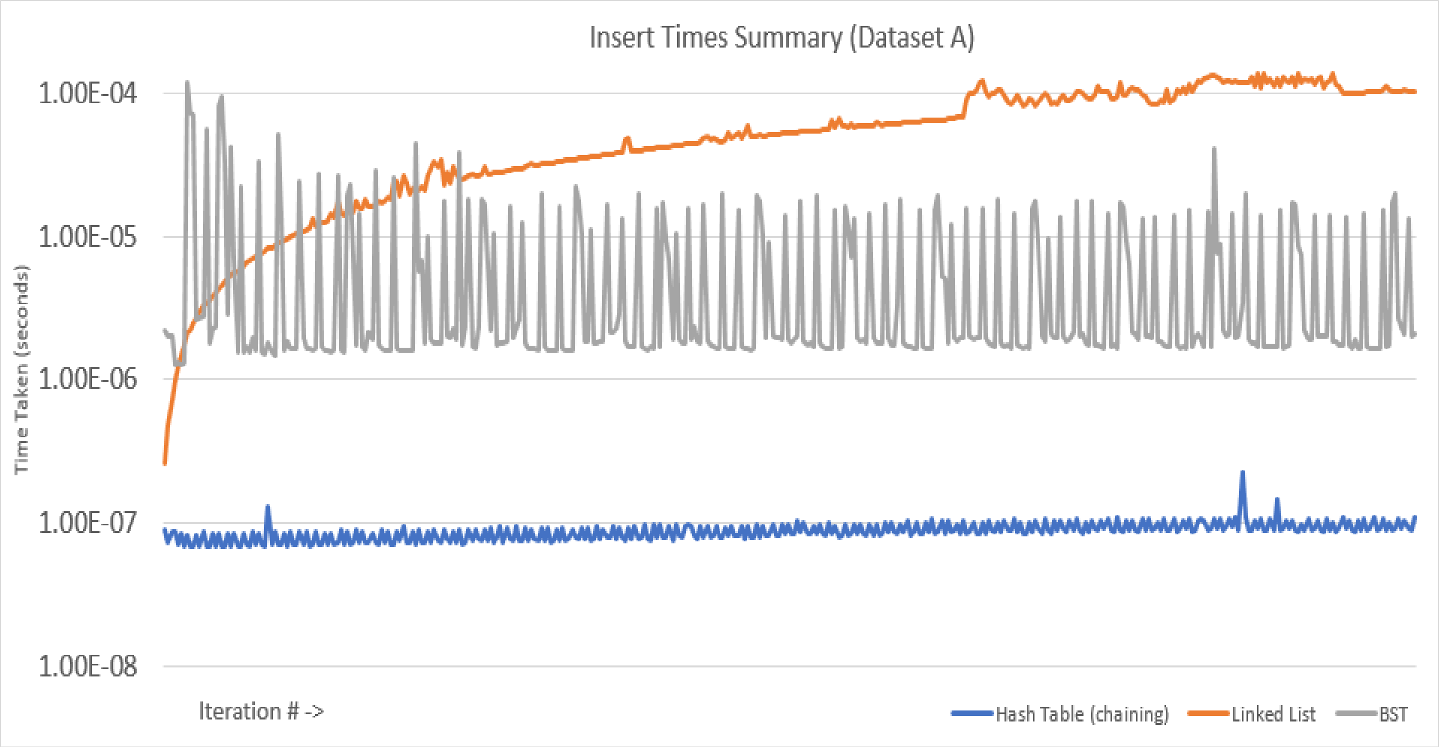

Insert/Search Performance for Hash Table with Linear Probing (Dataset B)
Time Taken (seconds)
1.00E-06
1.00E-07
1.00E-08
# of Collisions
100
90
80
70
60
50
40
30
20
10
0
Iteration # ->
Insert Times
Search Times
Insert Collisions
Search Collisions

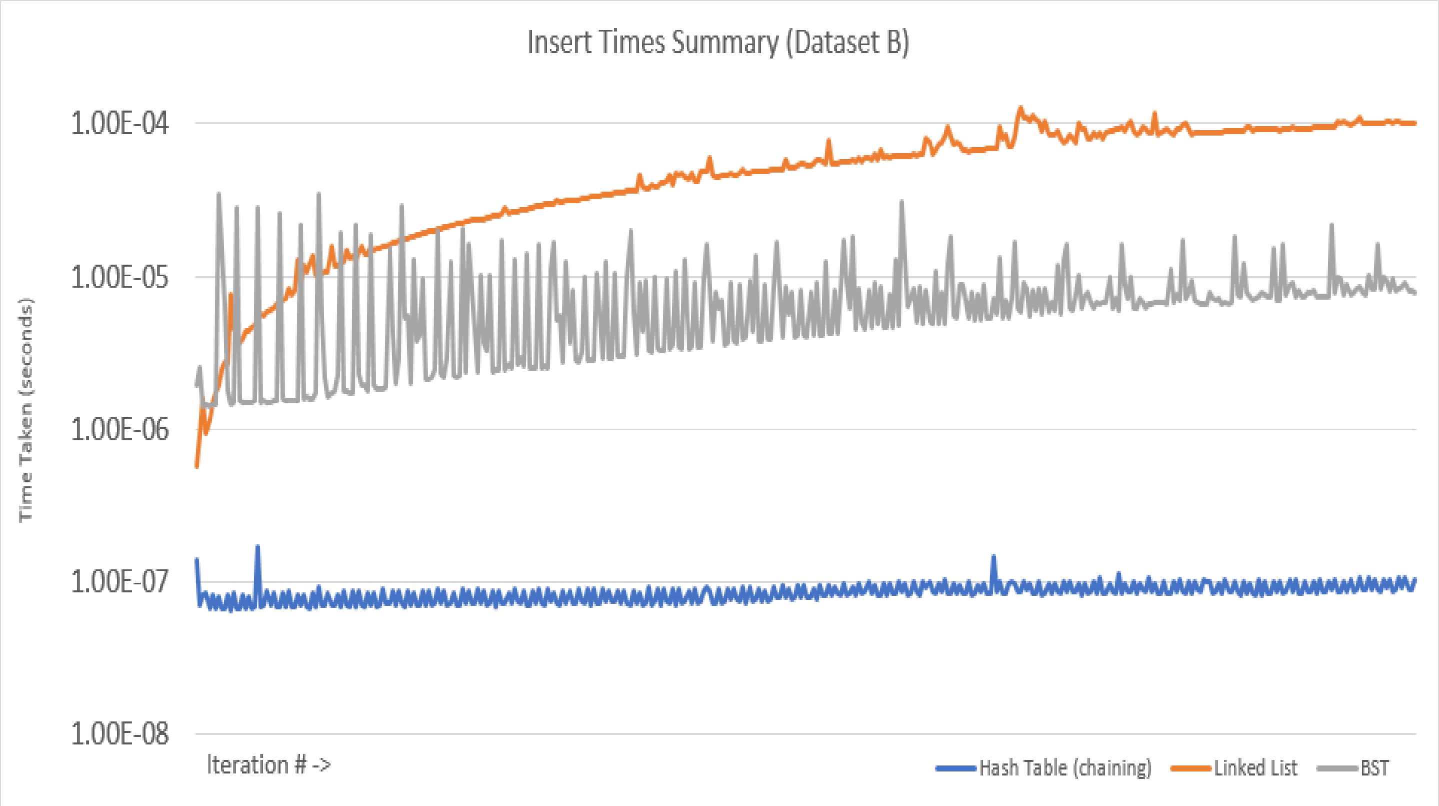

Insert/Search Performance for Hash Table with Quadratic Probing (Dataset A)
Time Taken (seconds)
1.00E-07
1.00E-08
# of Collisions
60
50
40
30
20
10
0
1 10 19 28 37 46 55 64 73 82 91 100 109 118 127 136 145 154 163 172 181 190 199 208 217 226 235 244 253 262 271 280 289 298 307 316 325 334 343 352 361 370 379 388 397
Iteration # ->
Insert Times
Search Times
Insert Collisions
Search Collisions

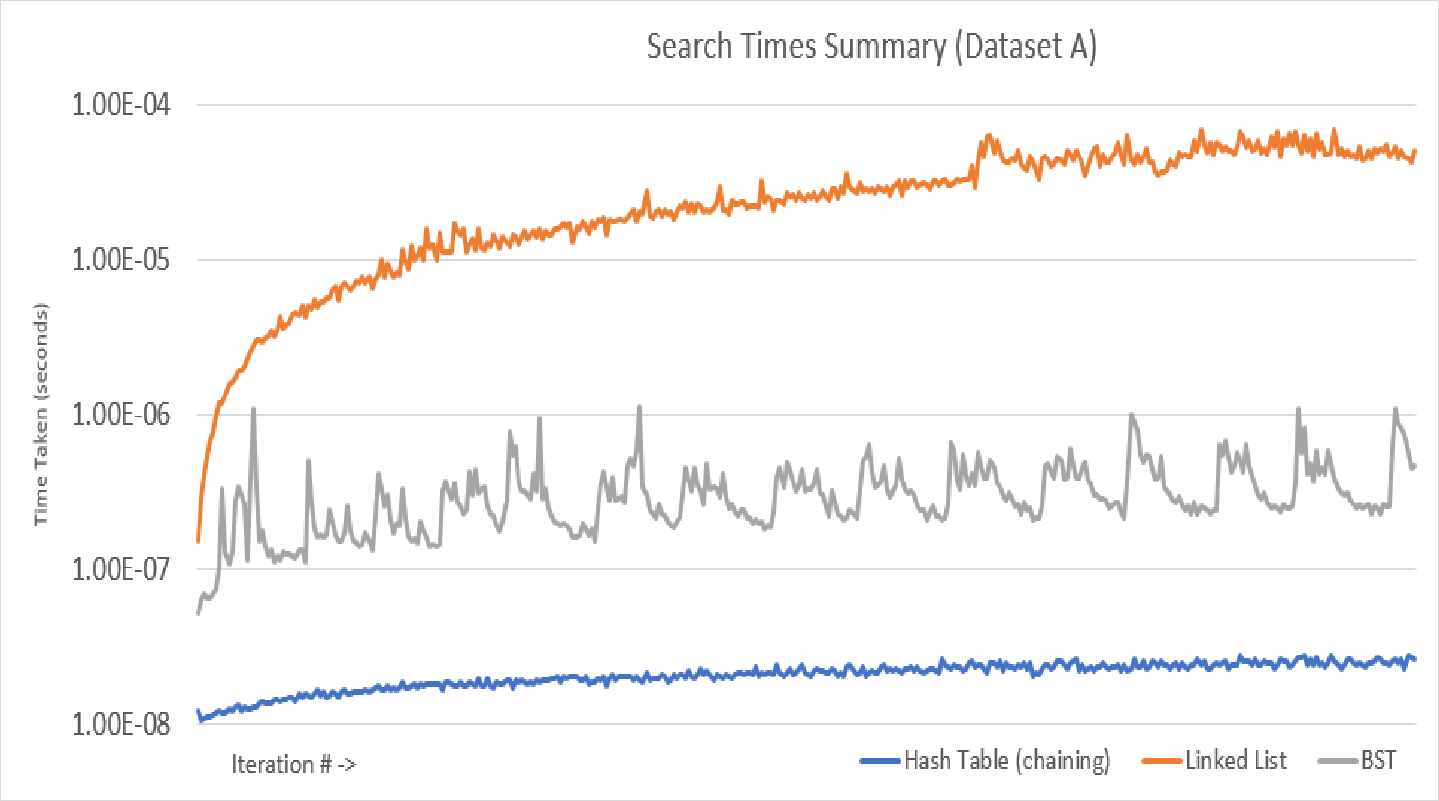

Insert/Search Performance for Hash Table with Quadratic Probing (Dataset B)
Time Taken (seconds)
1.00E-07
1.00E-08
# of Collisions
60
50
40
30
20
10
0
1 10 19 28 37 46 55 64 73 82 91 100 109 118 127 136 145 154 163 172 181 190 199 208 217 226 235 244 253 262 271 280 289 298 307 316 325 334 343 352 361 370 379 388 397
Iteration #
Insert Times
Search Times
Insert Collisions
Search Collisions

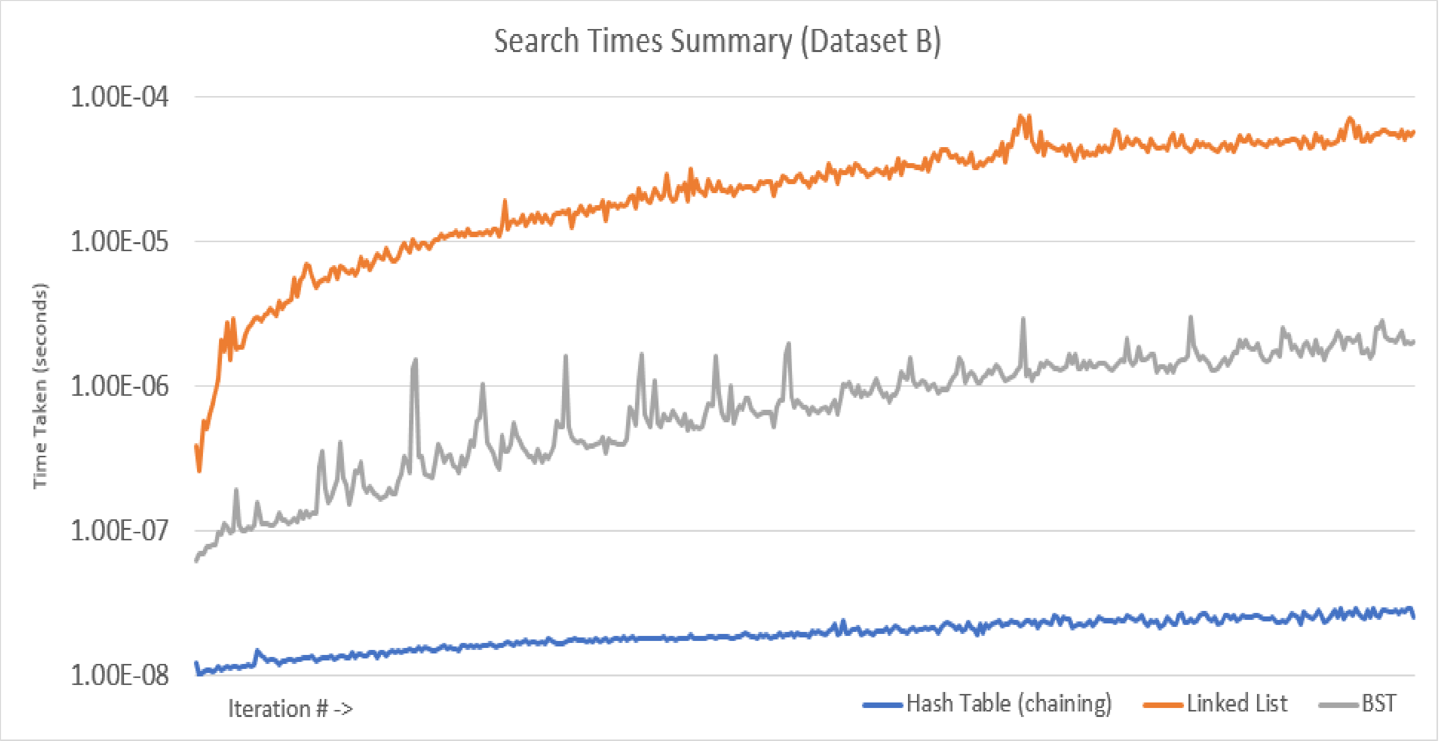

Insert/Search Performance for Hash Table with Chaining (Dataset A)


Insert/Search Performance for Hash Table with Chaining (Dataset B)


Insert Times Summary (Dataset A)
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Time Taken (seconds)
Iteration # ->
Hash Table (chaining)
Linked List
BST


Insert Times Summary (Dataset B)
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Time Taken (seconds)
Iteration # ->
Hash Table (chaining)
Linked List
BST


Search Times Summary (Dataset A)
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Time Taken (seconds)
Iteration # ->
Hash Table (chaining)
Linked List
BST


Search Times Summary (Dataset B)
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Time Taken (seconds)
Iteration # ->
Hash Table (chaining)
Linked List
BST

After conducting the necessary tests and analyzing the results, I have concluded the best data structure for the USPS to use for tracking shipping numbers is a hash table utilizing chaining for collision resolution. The linked list was the first and easiest to rule out, as it performed the slowest to start and got much worse as the data scaled up. This is because to find or insert anything, you must traverse past every other point already in the list. The binary search tree was next to go, and while it performed better than the linked list, it was subject to the same issues with data scaling as the list. This makes sense, because the binary search tree is effectively a method of cutting through the list, so there's less datapoints to go through before reaching the desired place. I then found that a hash table which uses chaining, linear probing, or quadratic probing for collision resolution can both insert and search through their data several orders of magnitude faster than when compared to a linked list or even binary search tree. This is because the hash table reduces the time it takes to reach any data point to a single line, then if a collision occurs it's likely only a few more operations. Among the hash tables, the quadratic probing and chaining methods of collision resolution proved superior to the linear probing method. This was due to a higher number of collisions during linear probing, which led to a large increase in both insert and search functions when the table neared capacity. I ultimately chose to recommend the hash table with chaining as the preferred method of data storage for the USPS because to get the quadratic probing method to have the same performance, it needed a table size twice as large (the quadratic method cannot guarantee a spot in the table if table size is equal to what was used for linear probing). Also, while it proved to have less collisions than with the chaining method, the latter seemed to solve the collisions faster as it did not seem to have much of an impact on time performance.