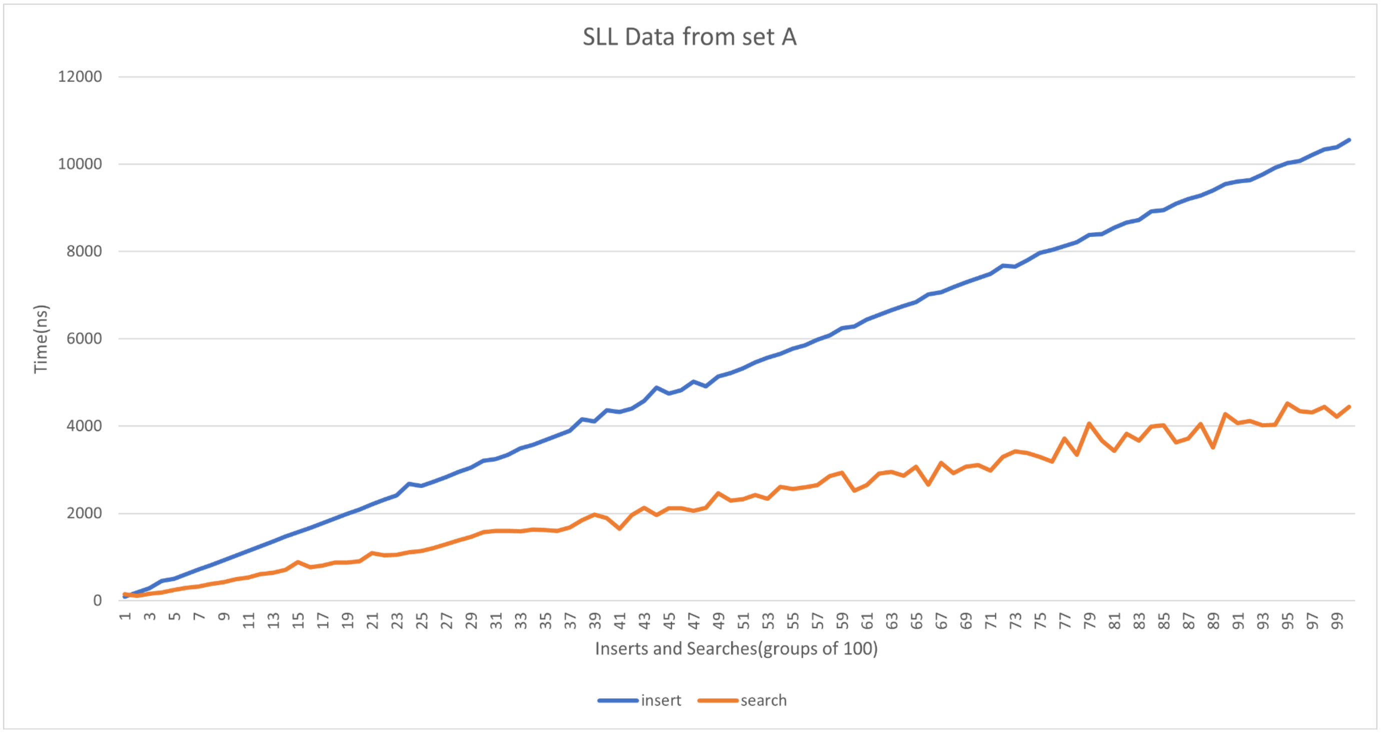
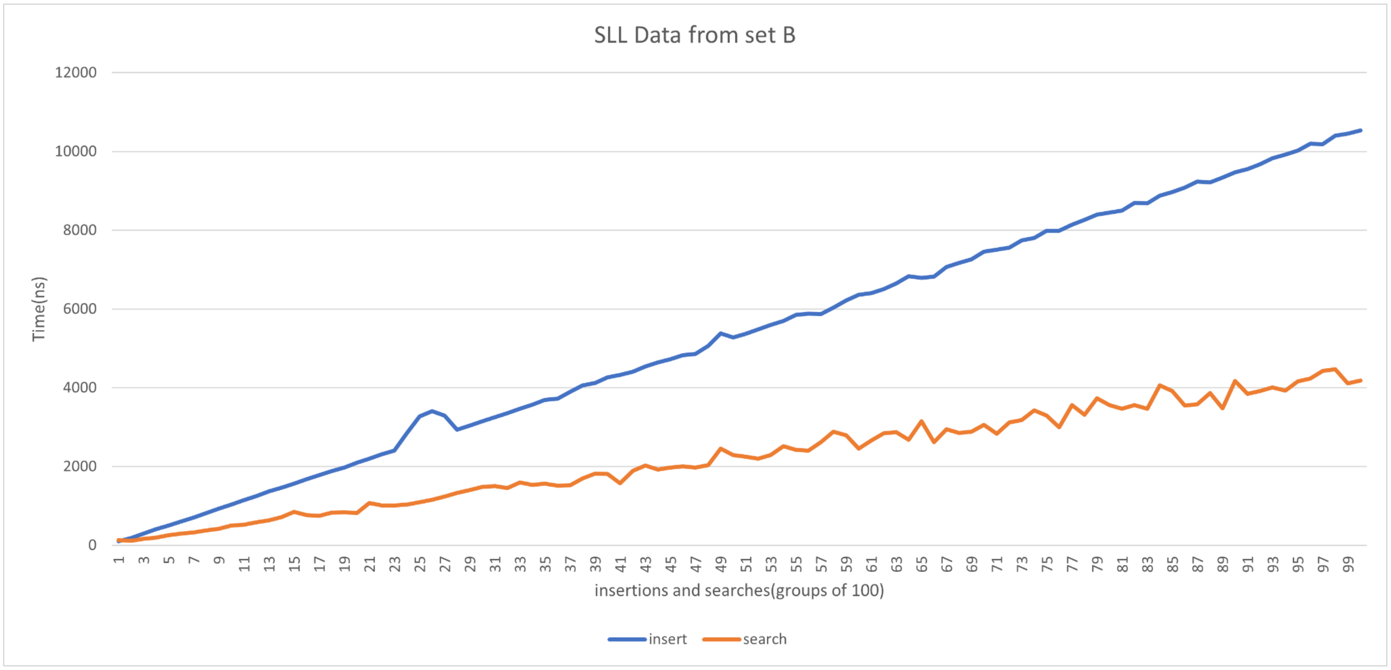
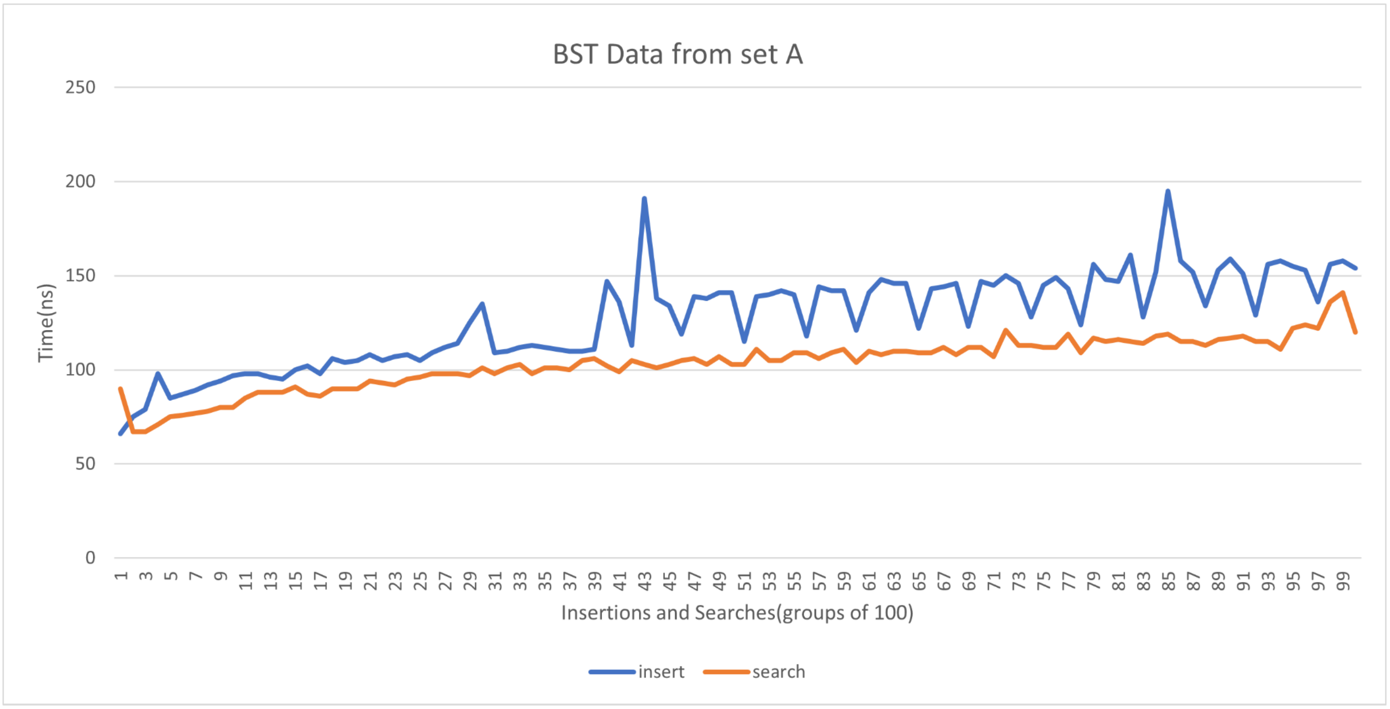
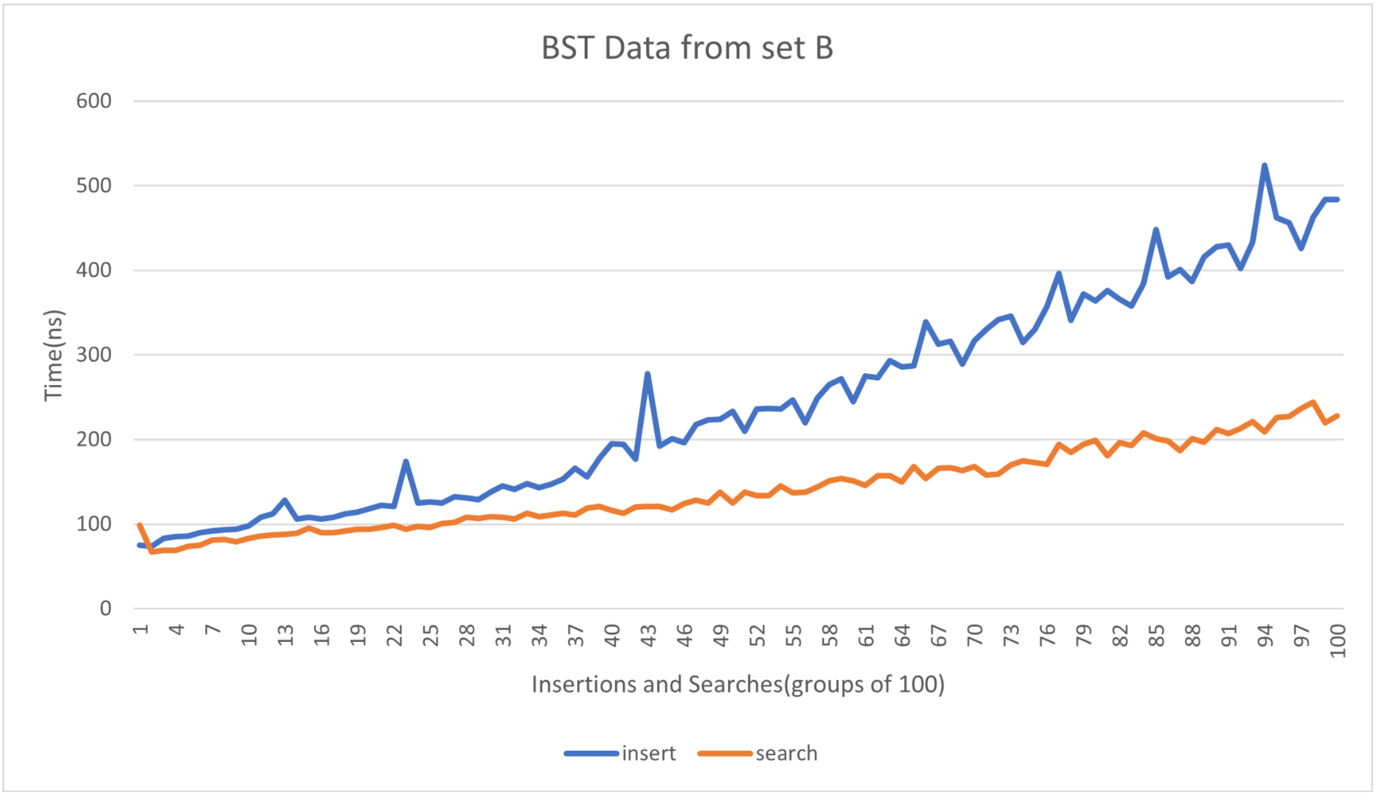
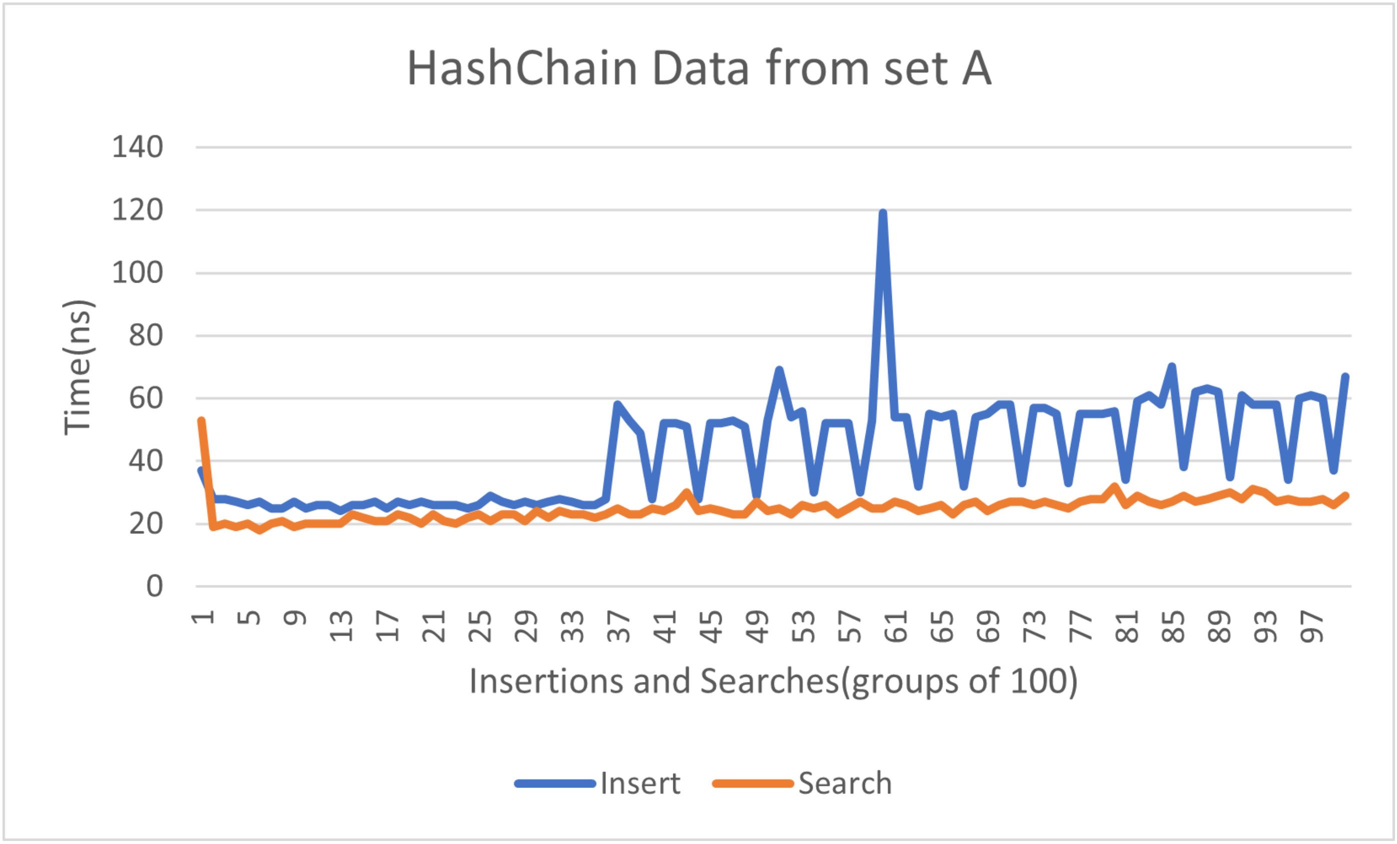
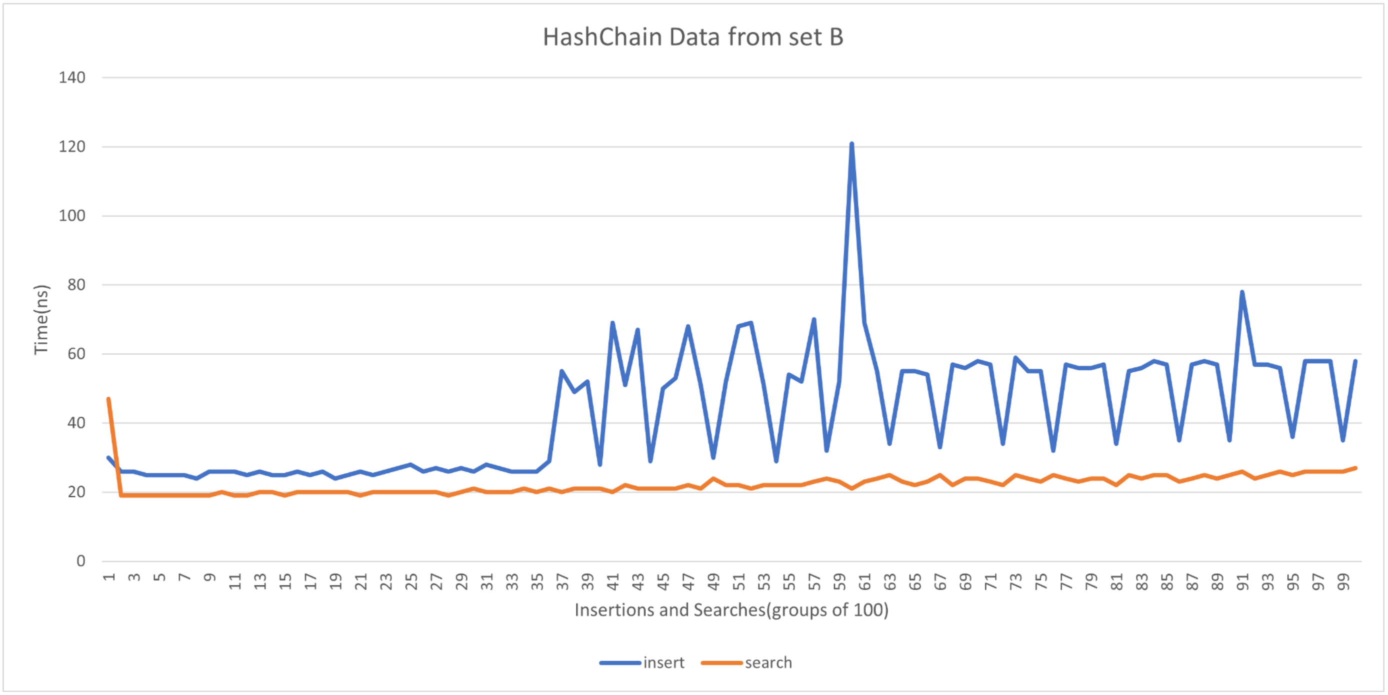

SLL Data from set A
Time(ns)
12000
10000
8000
6000
4000
2000
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
Inserts and Searches(groups of 100)
insert
search

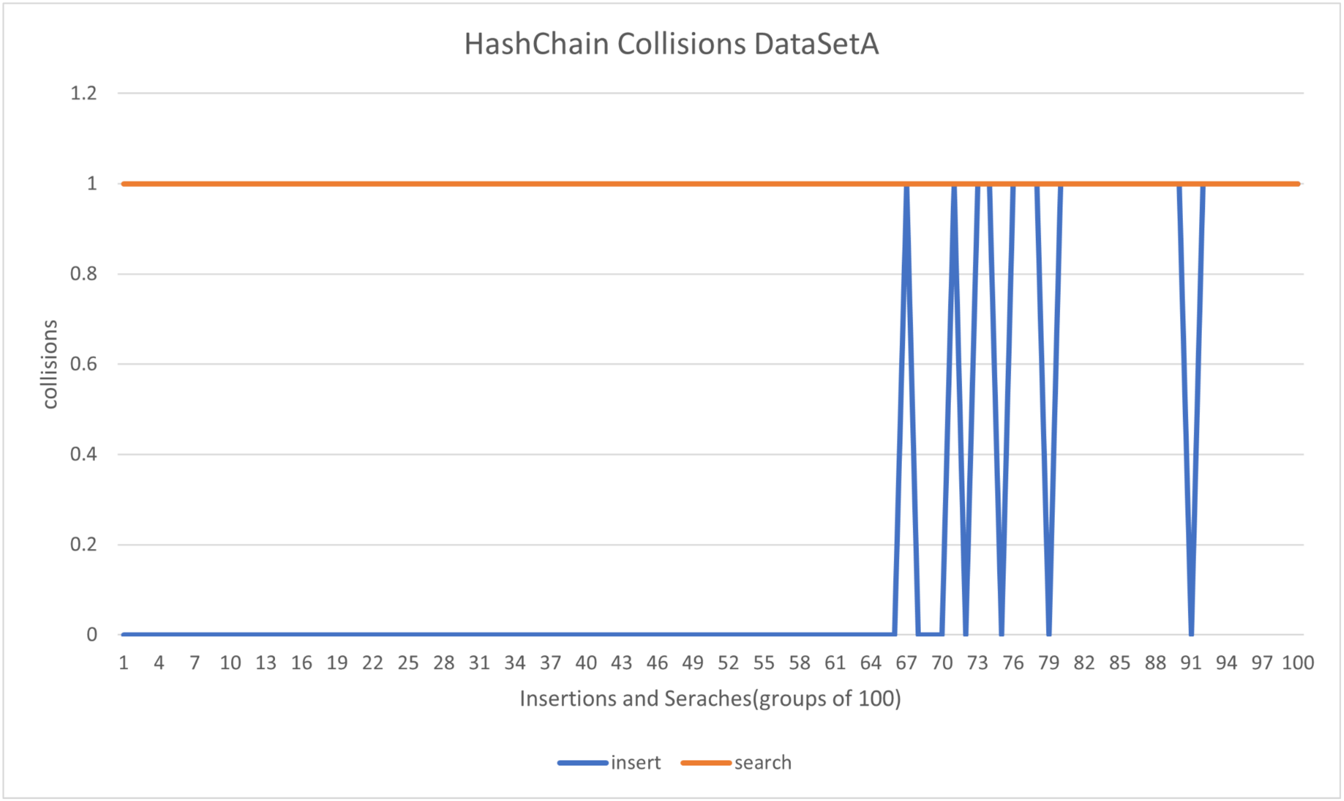

SLL Data from set B
Time(ns)
12000
10000
8000
6000
4000
2000
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
insertions and searches(groups of 100)
insert
search

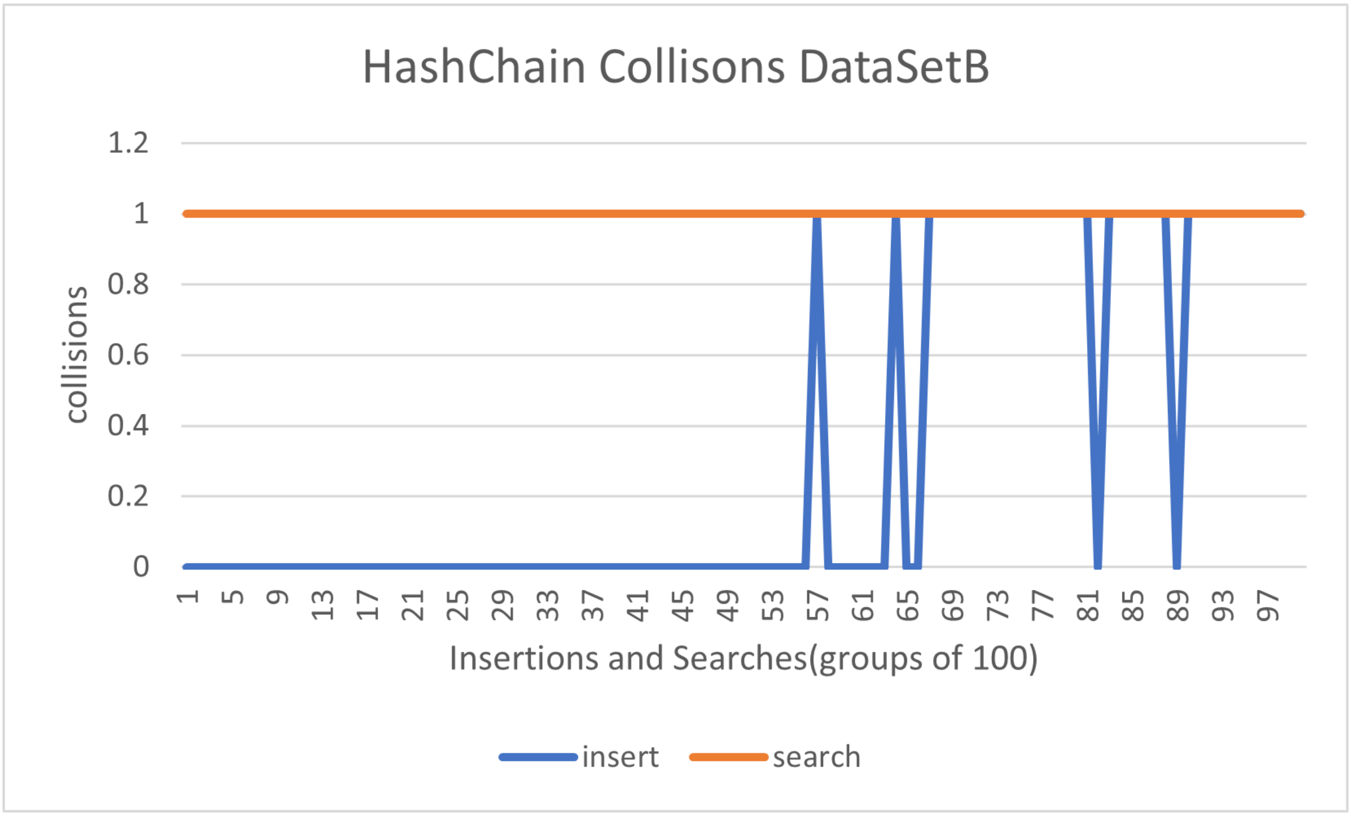

BST Data from set A
Time(ns)
250
200
150
100
50
0
Insertions and Searches(groups of 100)
insert
search

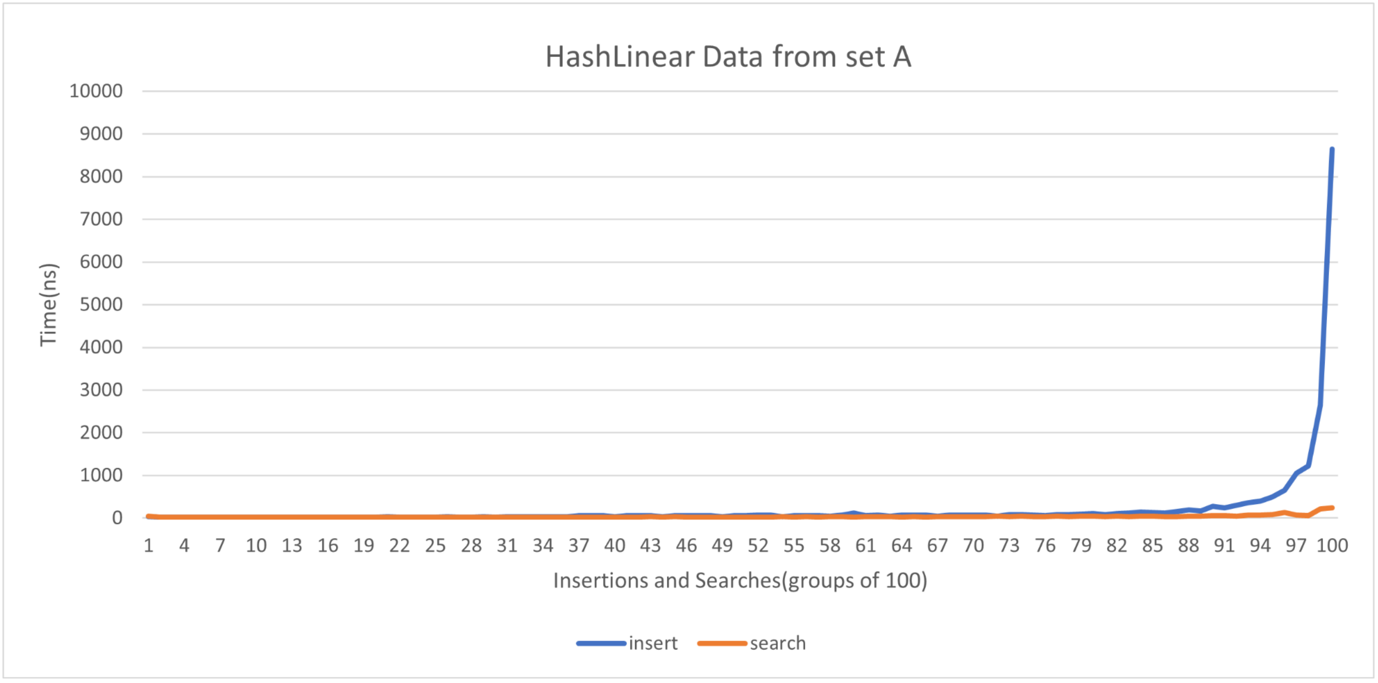

BST Data from set B
Time(ns)
600
500
400
300
200
100
0
Insertions and Searches(groups of 100)
insert
search

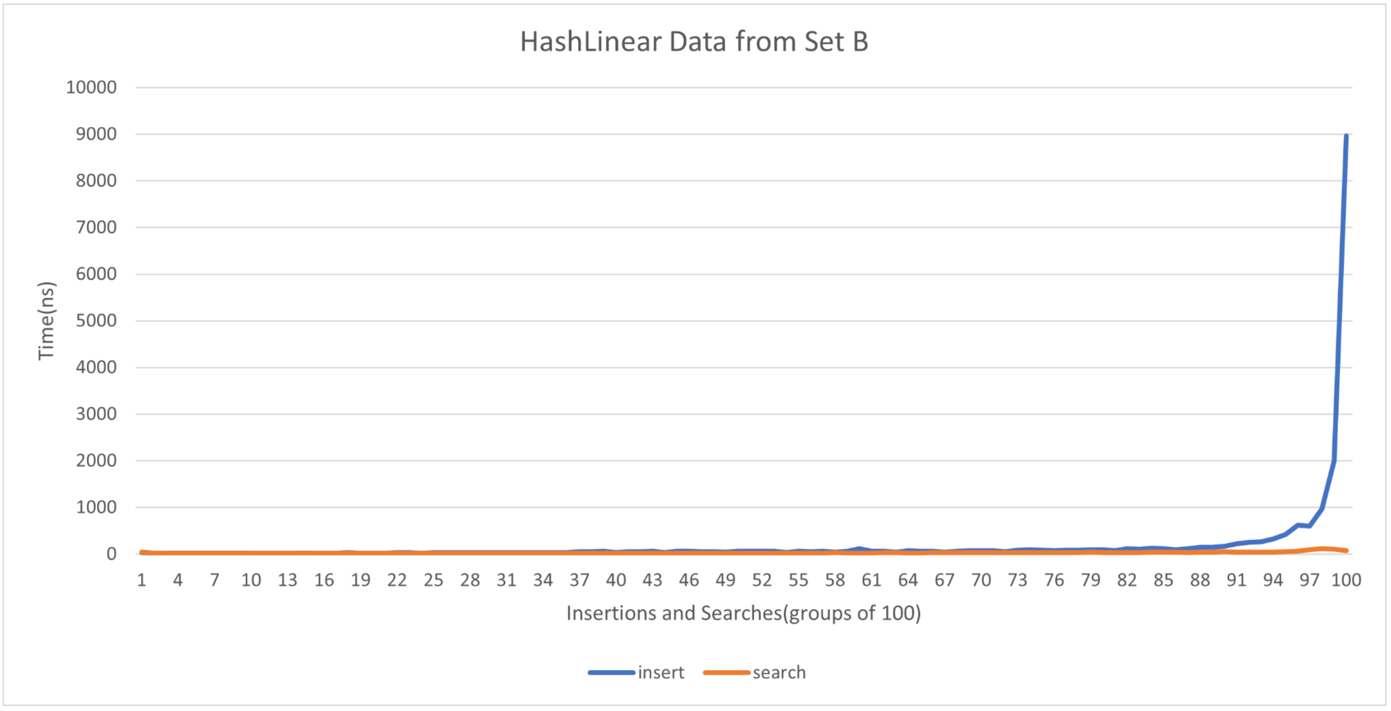

HashChain Data from set A
Time(ns)
140
120
100
80
60
40
20
0
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97
Insertions and Searches(groups of 100)
Insert
Search

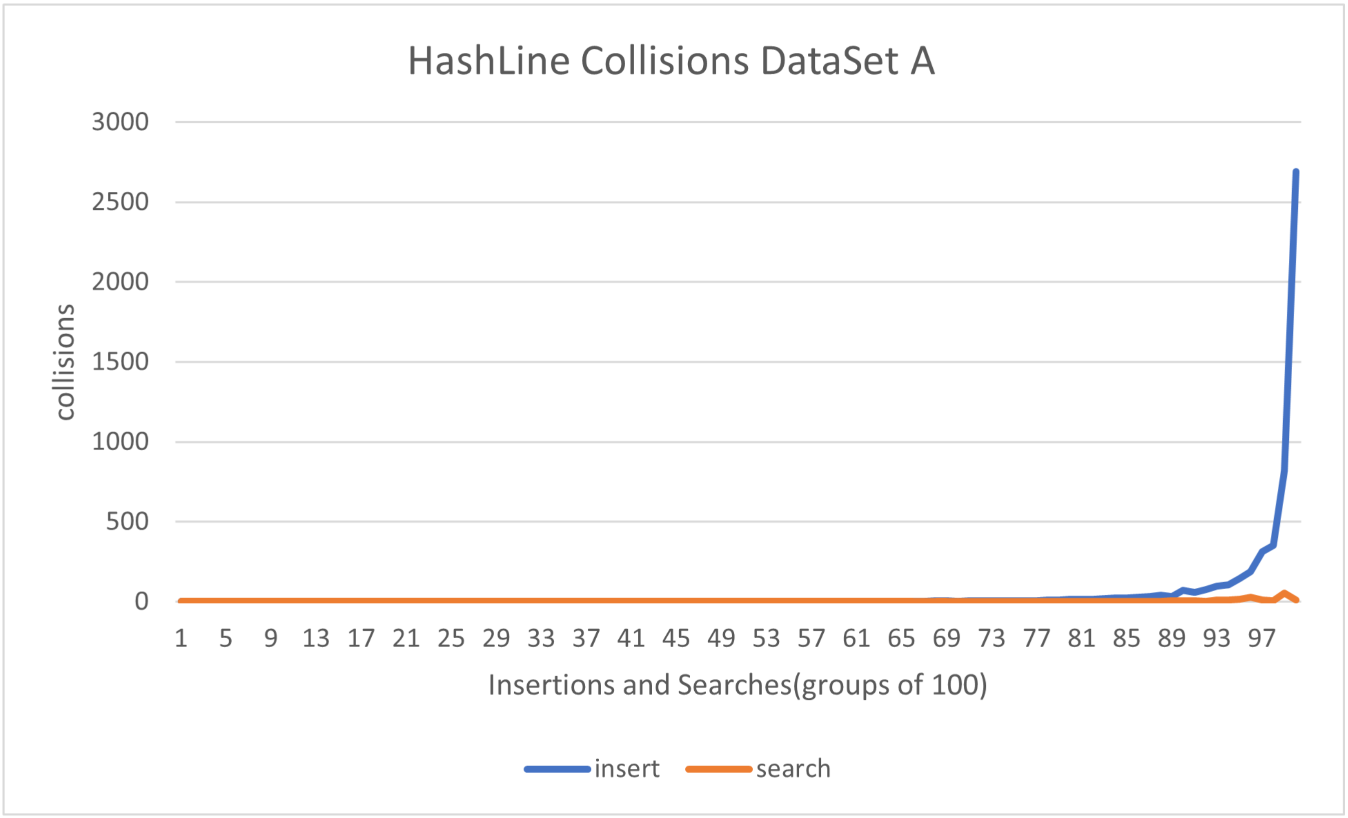

HashChain Data from set B
Time(ns)
140
120
100
80
60
40
20
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
Insertions and Searches(groups of 100)
insert
search

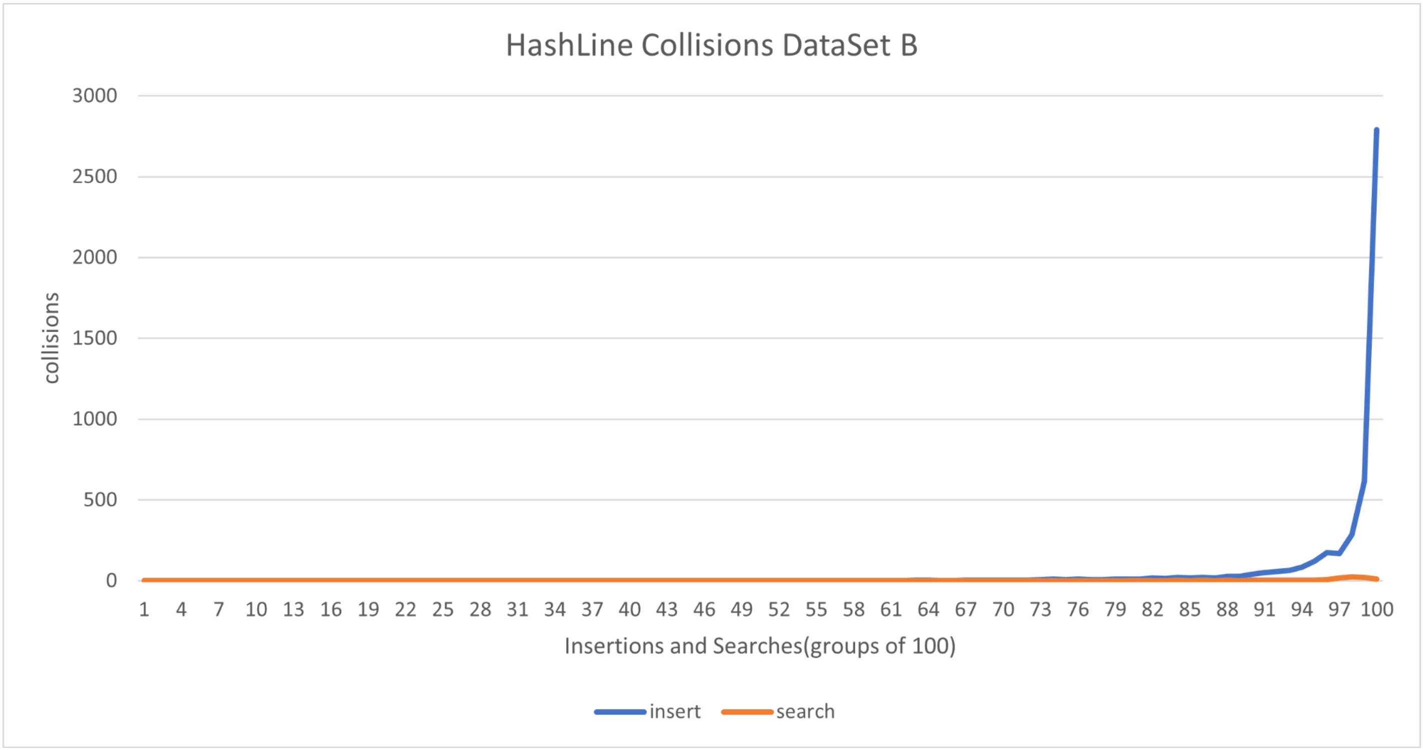

HashChain Collisions DataSetA
1.2
1
0.8
0.6
0.4
0.2
0
collisions
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97 100
Insertions and Seraches(groups of 100)
insert
search

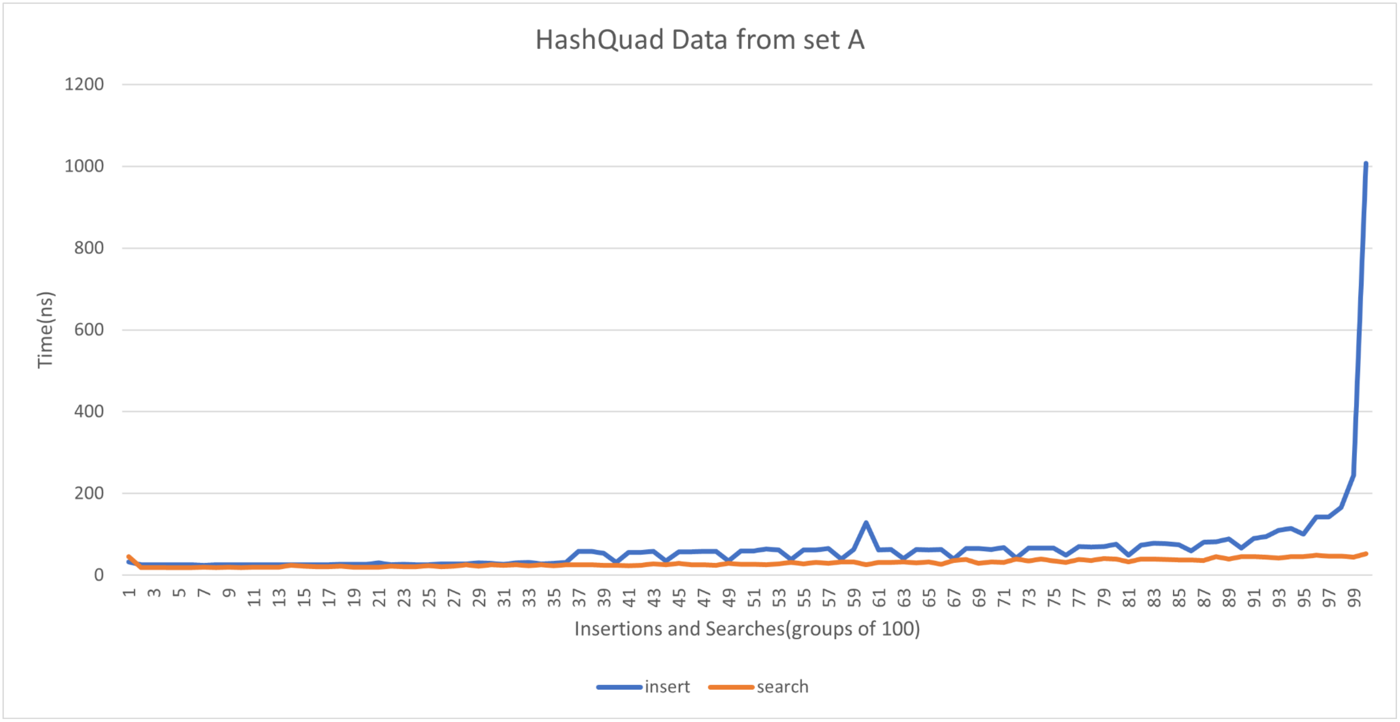

HashChain Collisons DataSetB
1.2
1
0.8
0.6
0.4
0.2
0
collisions
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97
Insertions and Searches(groups of 100)
insert
search

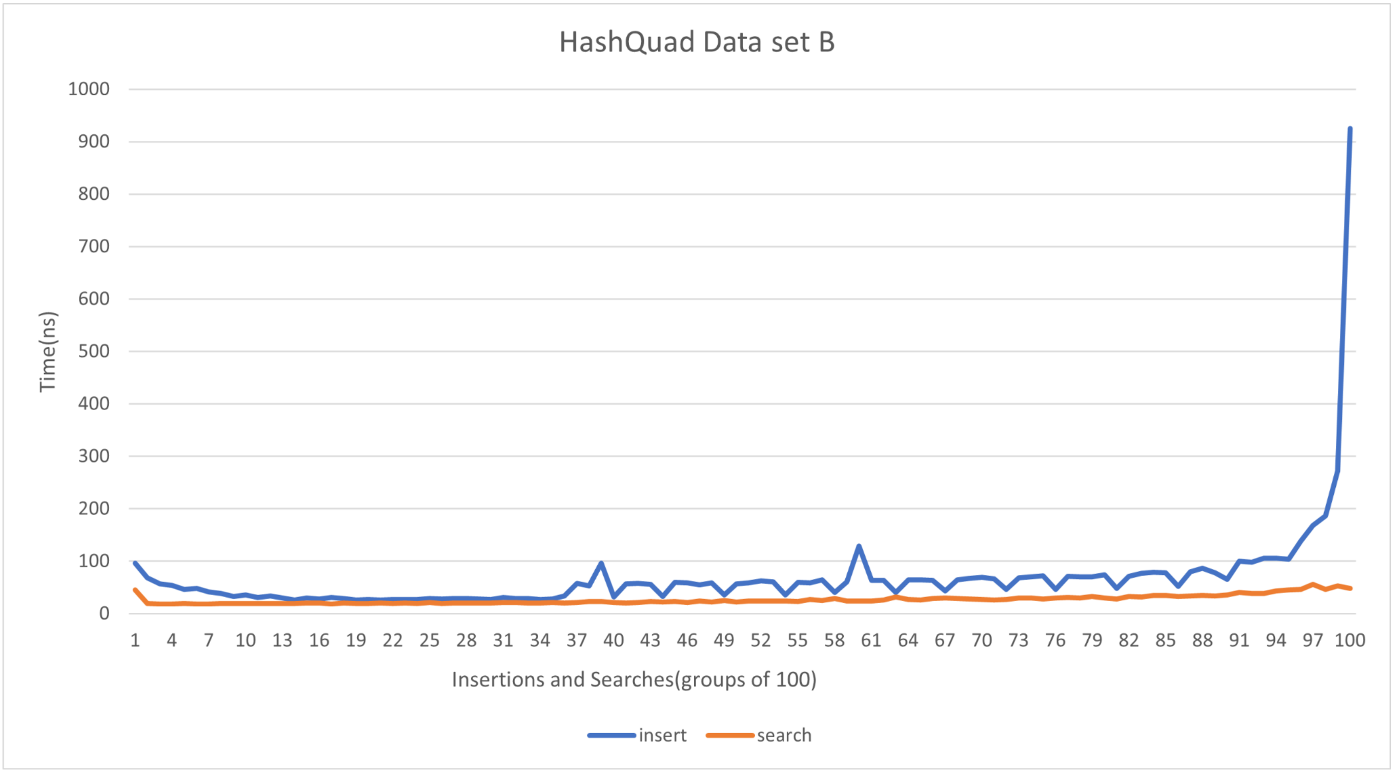

HashLinear Data from set A
Time(ns)
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97 100
Insertions and Searches(groups of 100)
insert
search

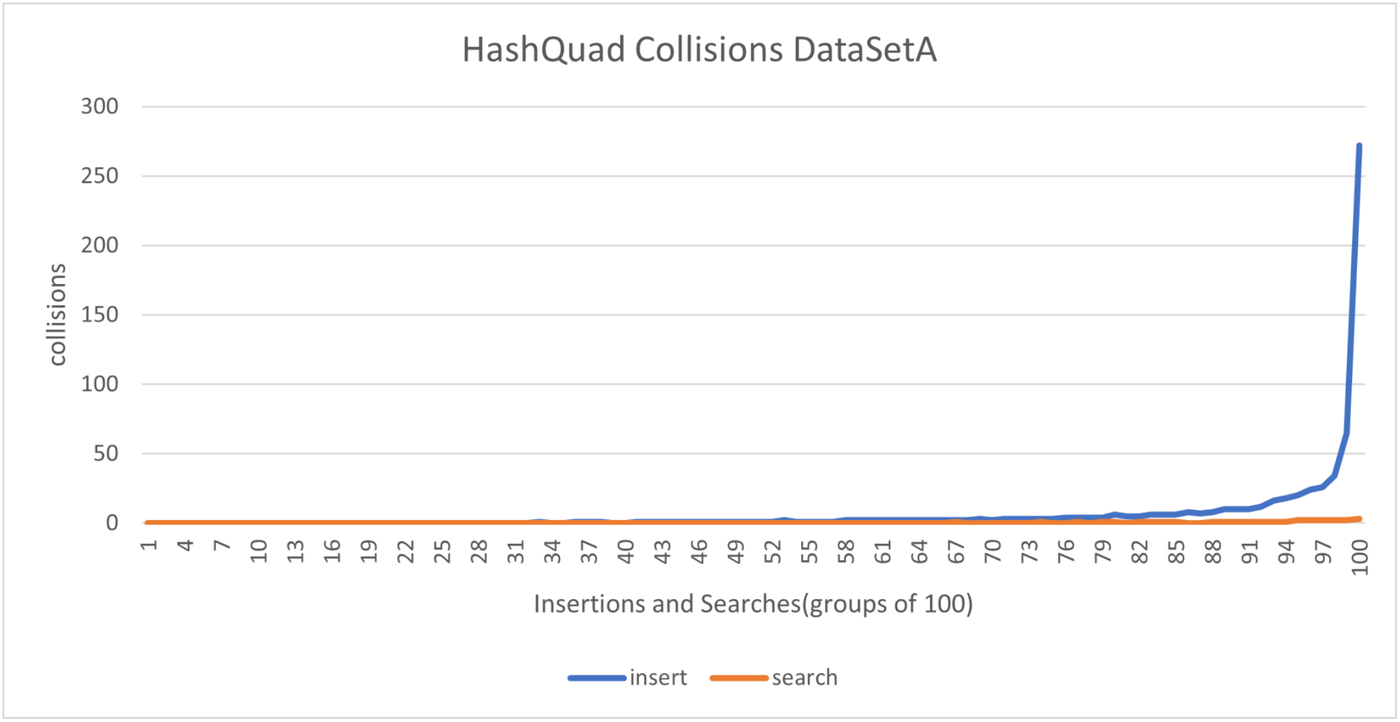

HashLinear Data from Set B
Time(ns)
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97 100
Insertions and Searches(groups of 100)
insert
search

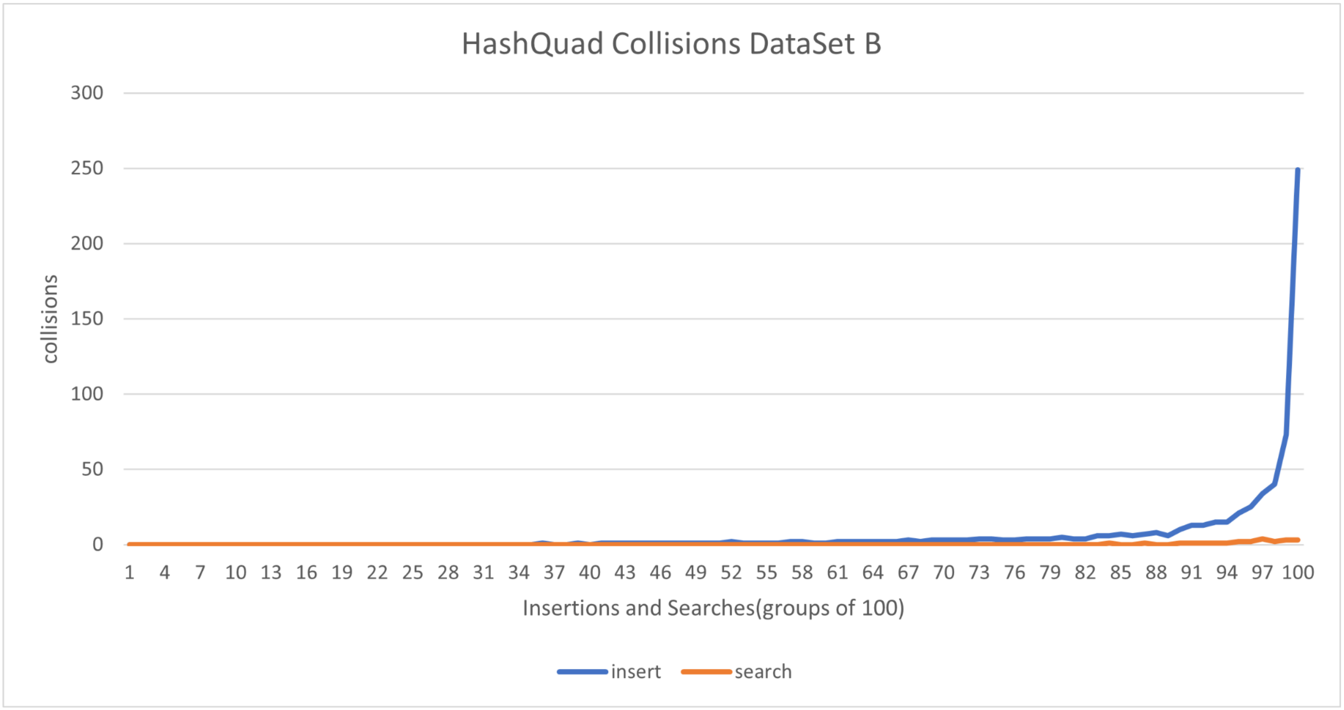

HashLine Collisions DataSet A
collisions
3000
2500
2000
1500
1000
500
0
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97
Insertions and Searches(groups of 100)
insert
search

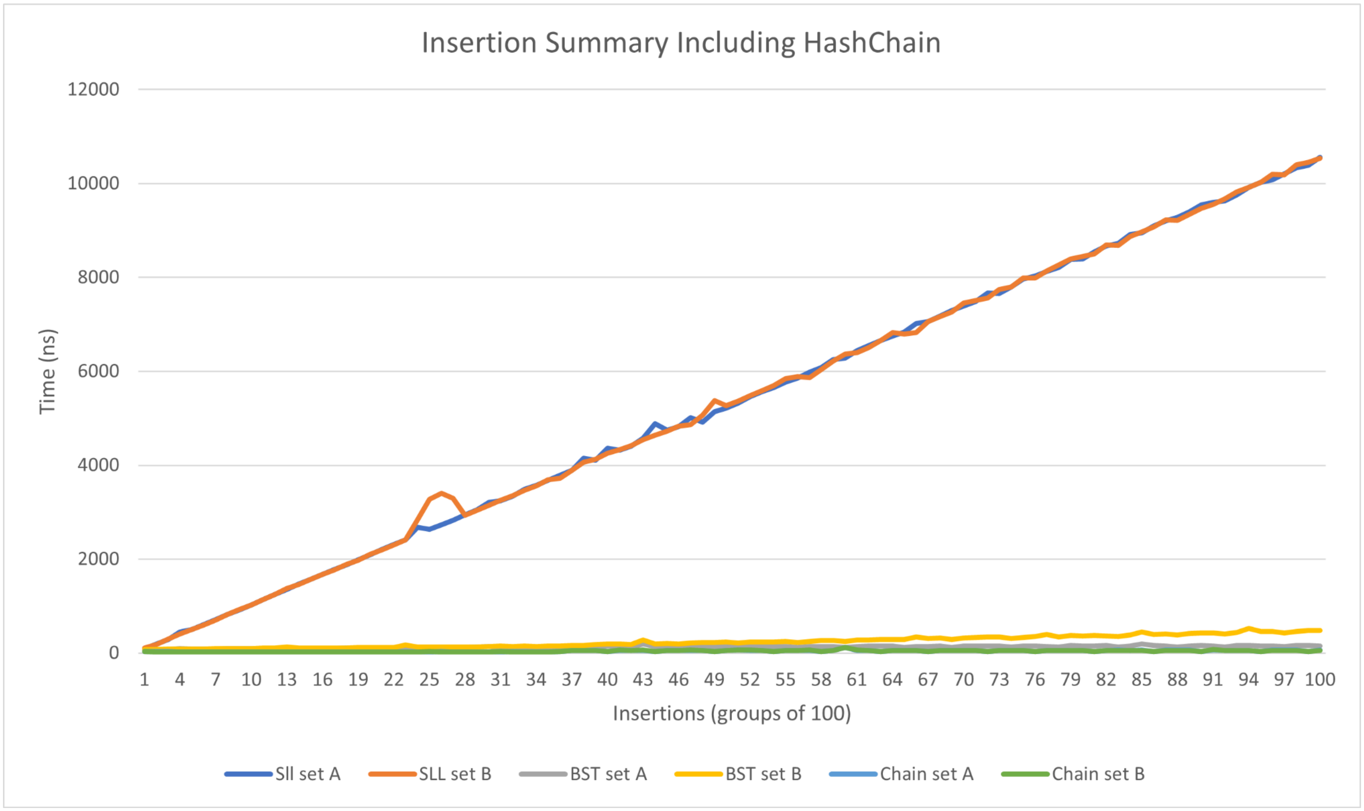

HashLine Collisions DataSet B
collisions
3000
2500
2000
1500
1000
500
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97 100
Insertions and Searches(groups of 100)
insert
search

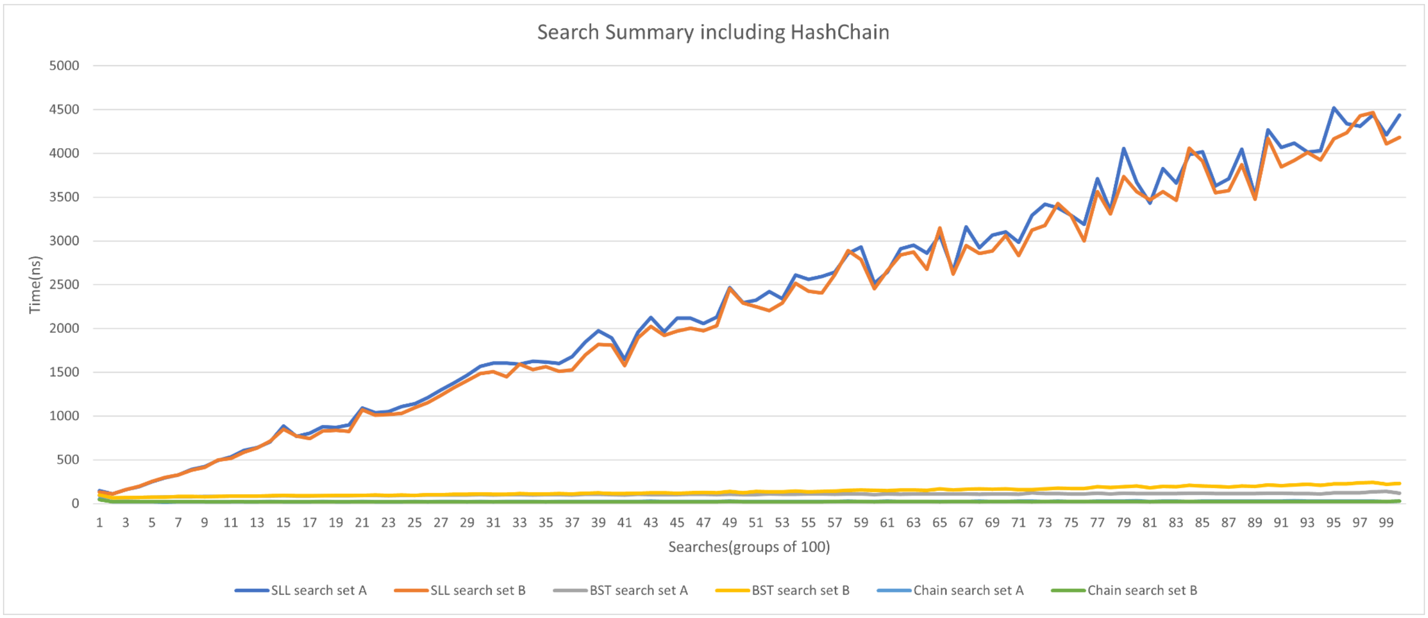

HashQuad Data from set A
Time(ns)
1200
1000
800
600
400
200
0
Insertions and Searches(groups of 100)
insert
search

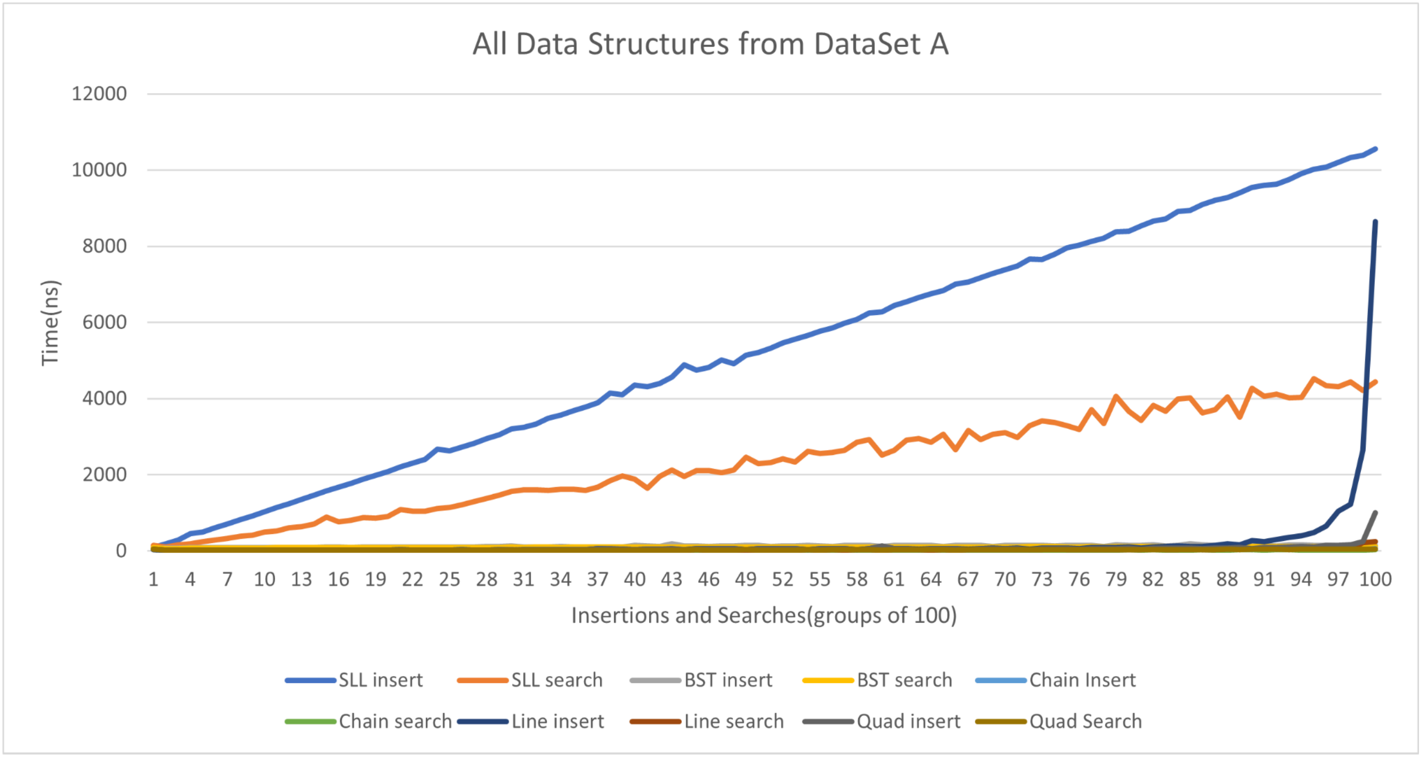

HashQuad Data set B
Time(ns)
1000
900
800
700
600
500
400
300
200
100
0
Insertions and Searches(groups of 100)
insert
search

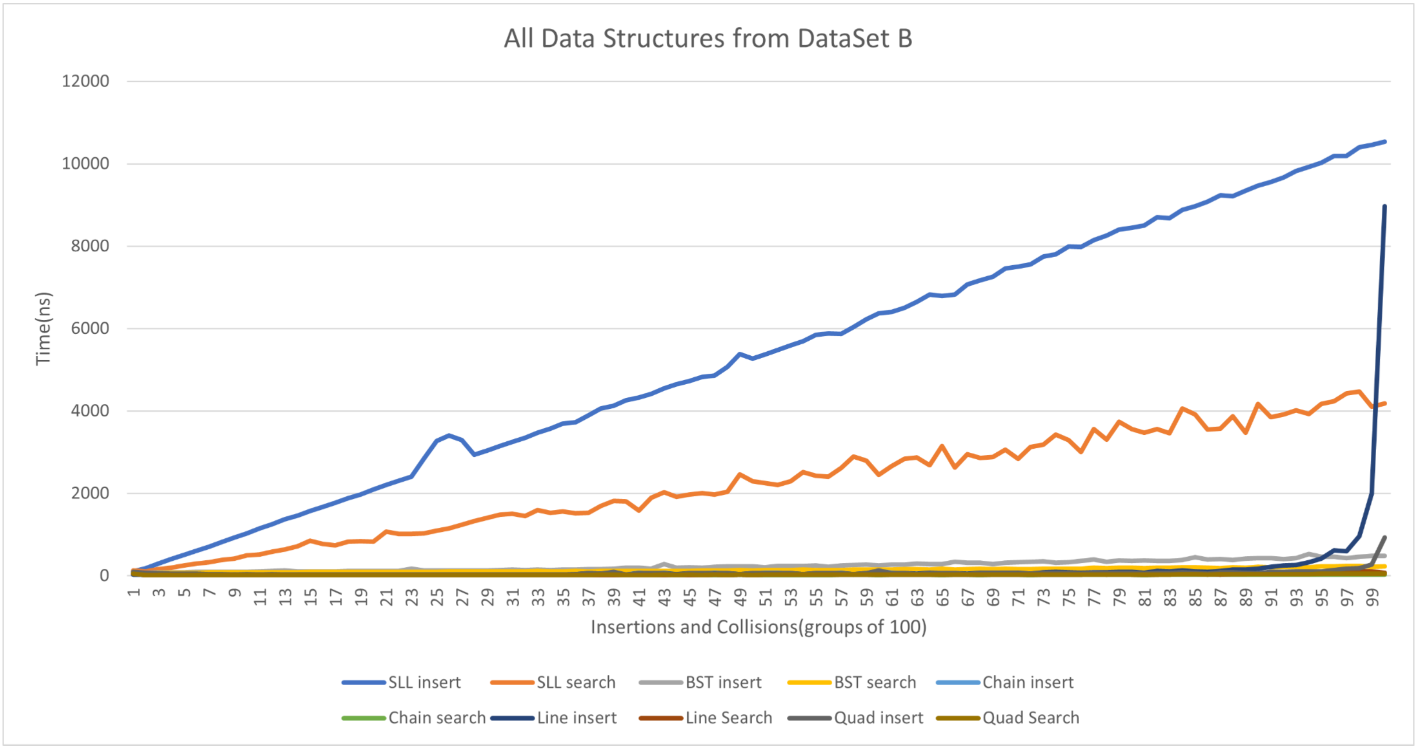

HashQuad Collisions DataSetA
collisions
300
250
200
150
100
50
0
Insertions and Searches(groups of 100)
insert
search


HashQuad Collisions DataSet B
collisions
300
250
200
150
100
50
0
Insertions and Searches(groups of 100)
insert
search


Insertion Summary Including HashChain
Time (ns)
12000
10000
8000
6000
4000
2000
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97 100
Insertions (groups of 100)
Sll set A
SLL set B
BST set A
BST set B
Chain set A
Chain set B


Search Summary including HashChain
Time(ns)
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
Searches(groups of 100)
SLL search set A
SLL search set B
BST search set A
BST search set B
Chain search set A
Chain search set B


All Data Structures from DataSet A
Time(ns)
12000
10000
8000
6000
4000
2000
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97 100
Insertions and Searches(groups of 100)
SLL insert
SLL search
BST insert
BST search
Chain Insert
Chain search
Line insert
Line search
Quad insert
Quad Search


All Data Structures from DataSet B
Time(ns)
12000
10000
8000
6000
4000
2000
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
Insertions and Collisions(groups of 100)
SLL insert
SLL search
BST insert
BST search
Chain insert
Chain search
Line insert
Line Search
Quad insert
Quad Search

Final Report

After examining the insertion and search times of all 5 data structures, all three Hash tables seem to be the best for searching as they do so in constant time. However, for the quad and linear hash tables the insertion times grew exponentially as more elements were placed and the table started to become clustered, but they were much better until the last items were being inserted. The chain implementation did not seem to have this issue as it just inserted into a linked list if the index was full. Also, the linear implementation had many more collisions than the quad, and the chain had the least amount of collisions which most likely contributed to the shorter time. The BST is still a very solid data structure however and was not far behind the hash tables and outperformed them, excepting the chain, in insertion times as more elements were inserted. The linked list was by far the worst structure as the insertion times and search times were thousands of times greater than all 4 of the other implementations. Regarding the data sets themselves DataSet A seems to be less ordered than DataSet B and the ordering could potentially lead to more clustering in the probing tables if the data is ordered. Although it would be beneficial for the BST to have ordered data. Overall, the hash chain table was the best for inserting and searching with this data. However with very large amounts of data it could eventually become like a linked list and lose a lot of performance.