Sarthak Shukla

Report

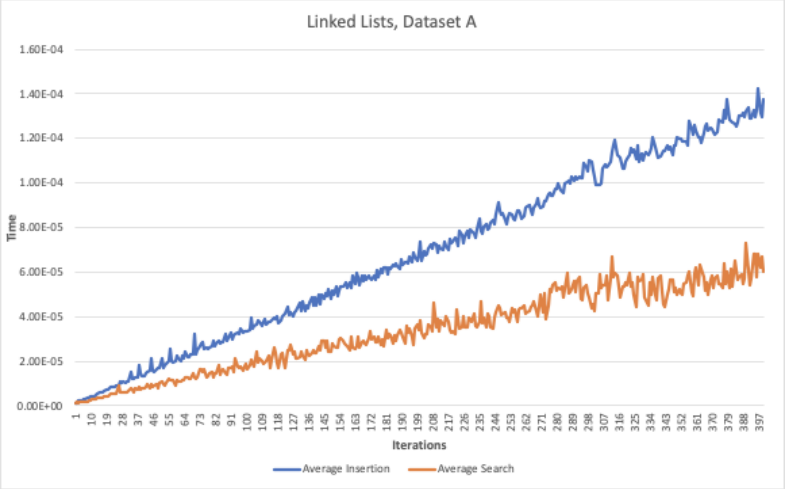
Based on my findings, I believe that the hash table implemented with quadratic probing is the most efficient of the data structures I have tested. As we know, a hash table with the collision resolution of quadratic probing is supposed to be O(1) in ideal situations. Linked lists have a O(n) time complexity for insertion and for search. In the first two graphs below, we can see that the linked lists do portray that in fact. With datasets of this size however, we should not be using linked lists because as we are inserting more objects into the list, it takes longer since we are inserting at the end, Moreover, search will take longer as well because we have to traverse the whole list from start to finish. In our graphs for data set A and data set B, we can see that the time linearly increases.

For binary search trees, the best-case time complexity is O(logn), and in this case we can see that being portrayed in our graphs. Now, binary search trees are more efficient than linked lists, because the tree itself is balanced allowing for more efficient insertion and search. In my graphs we can see that for data set A, the graph is mostly O(logn) for both insertion and search, although for data set B it is not. This might be due to the fact that the data in set B is not as balanced as in set A.

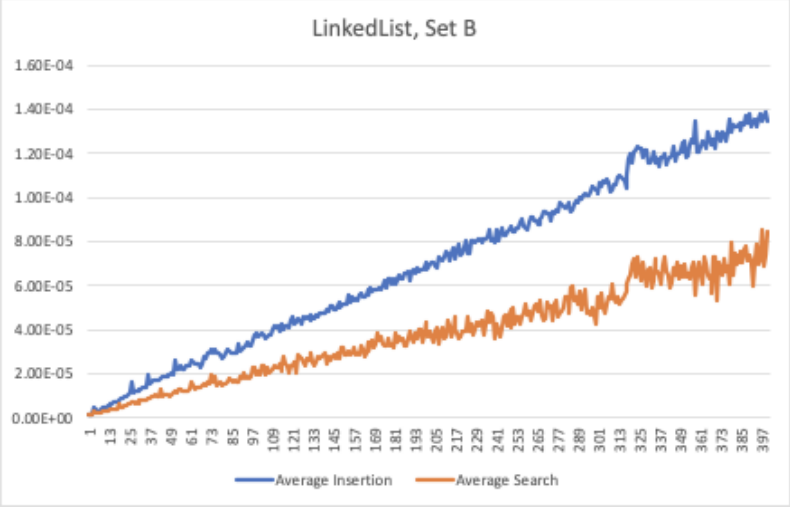
Now, when we look at all of our hash table graphs (chaining, linear probe, and quadratic probe), the best-case scenario for insertion and search in O(1). This is not very evident in our graphs because of the spikes we can see in the graph, but the spikes are caused each collision method. I believe that quadratic probing is the most efficient of all three collision resolutions because when looking at its insertion collisions and search collisions, they are relatively low for both datasets, while compared to linear probing and chaining. Additionally, quadratic probing seems to be the most constant of the spiking in the graphs, so I would say that it is the closest to O(1). Therefore, I would conclude that hash table with quadratic probing implemented as the collision resolution is the most efficient of the data structures that I have tested.

Graphs

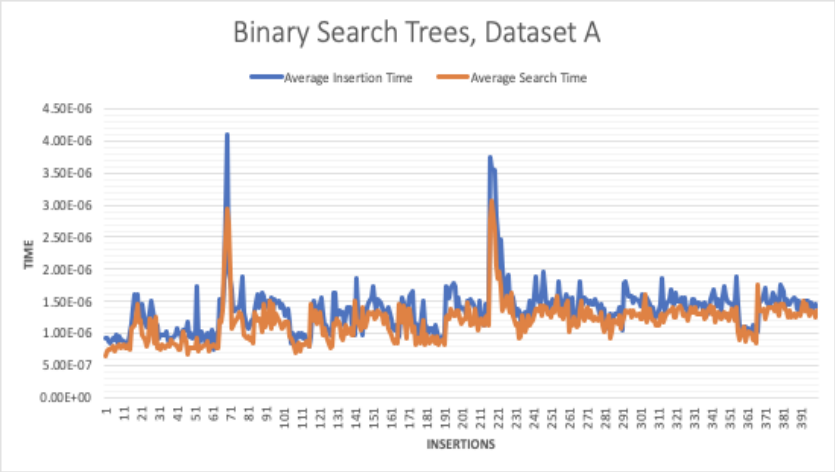
Linked Lists, Data Set A:



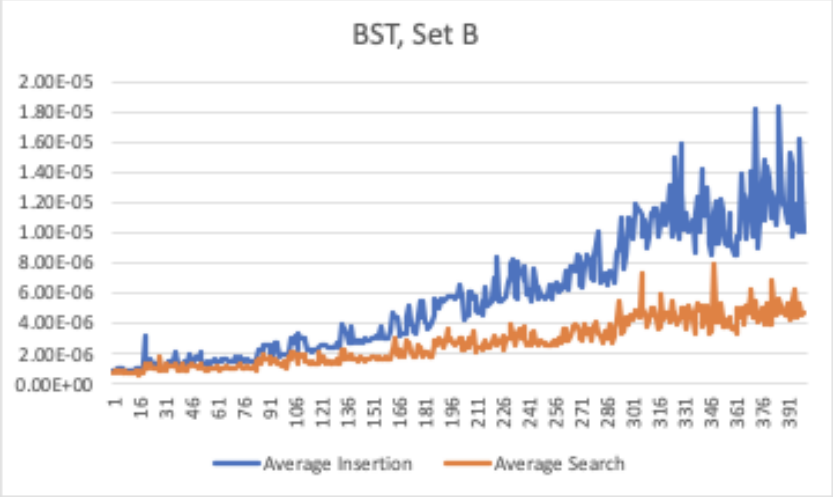
Linked Lists, Data Set B:



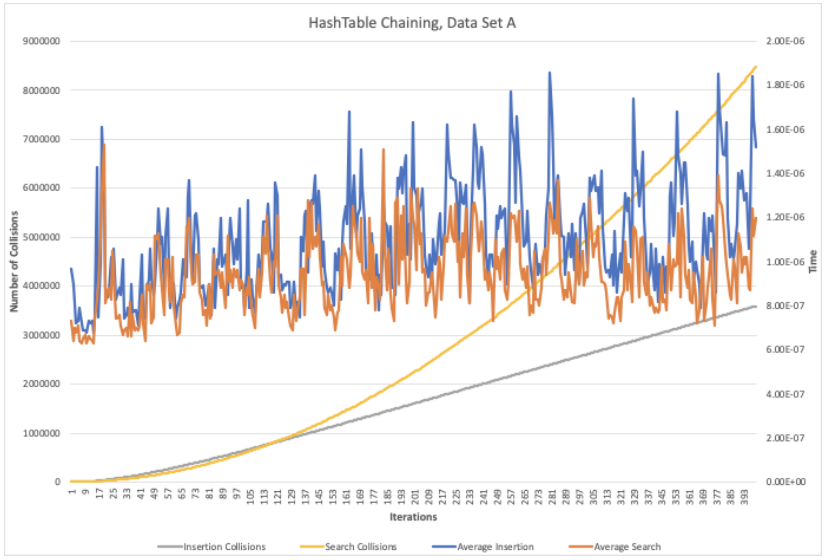
BST, Data Set A:



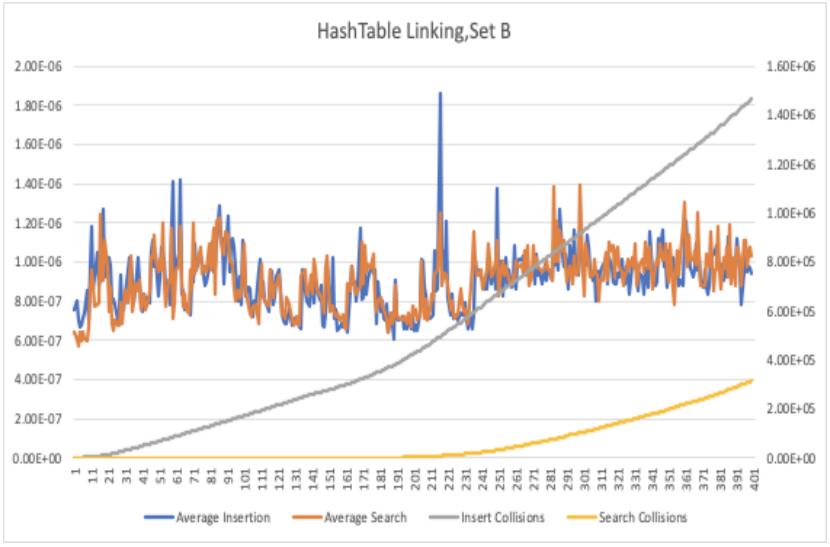
BST, Data Set B:



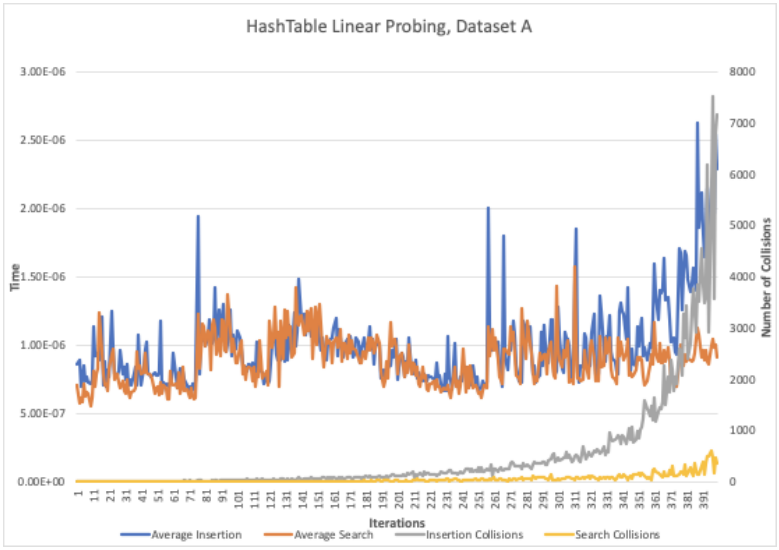
Hash Table with Linking, Dataset A:



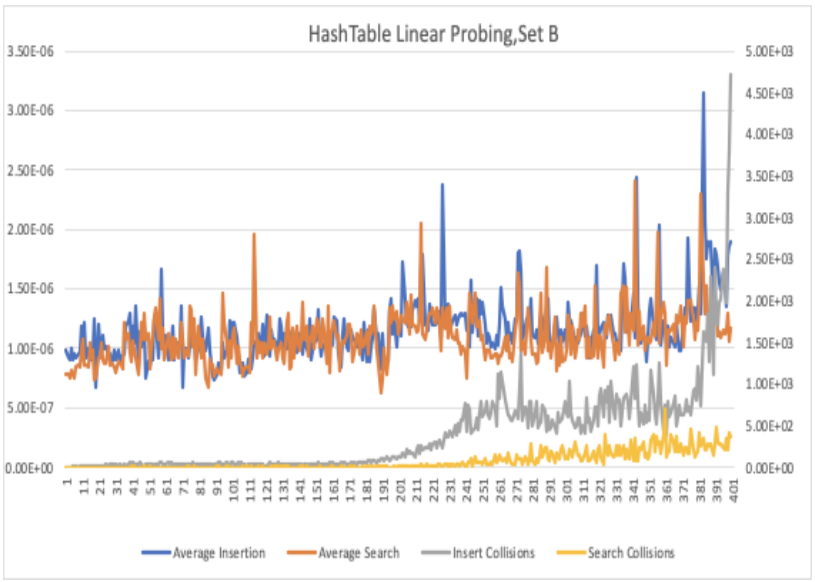
Hash Table with Linking Dataset B:



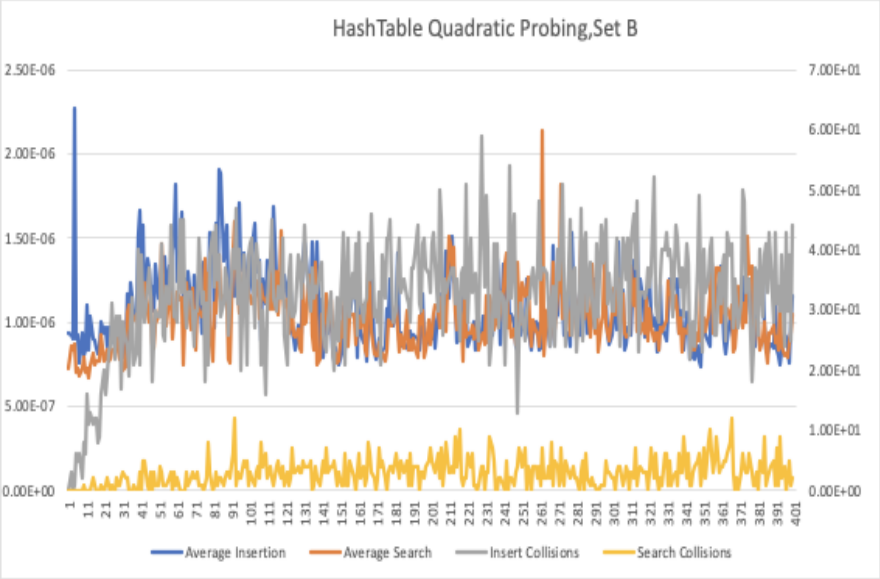
Hash Table with Linear Probing, Dataset A:



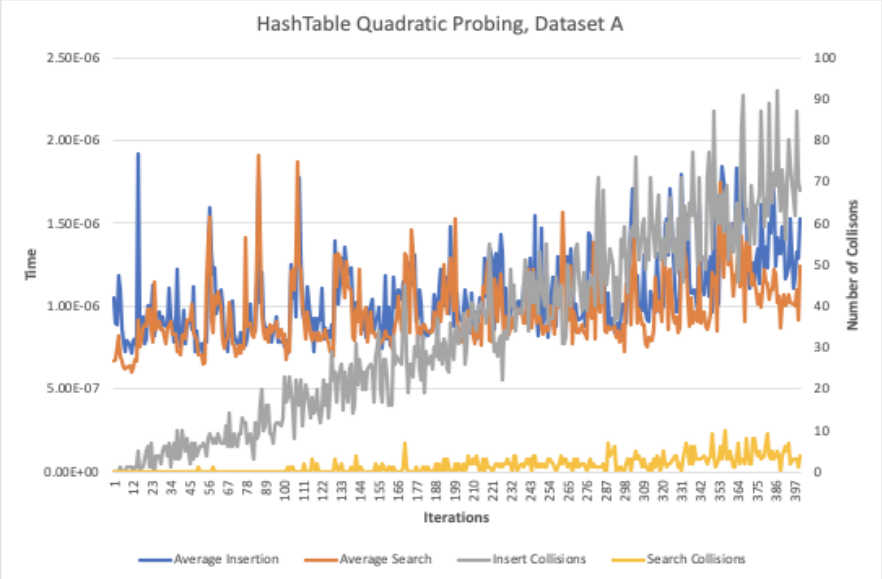
Hash Table with Linear Probing, Dataset B:



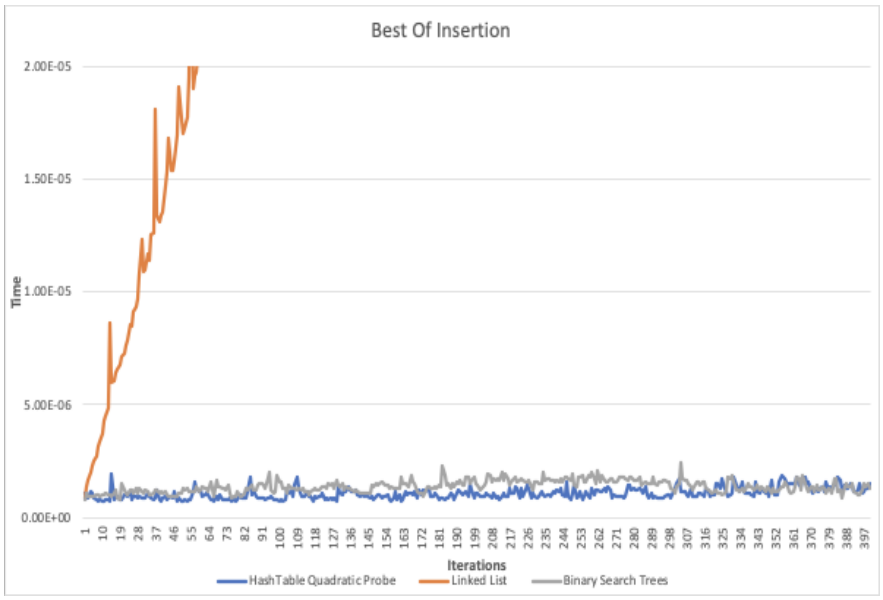
Hash Table with Quadratic Probing, Dataset A:



Hash Table with Quadratic Probing, Dataset B:



Linked Lists, Binary Trees, Hash Table with Quadratic Probing vs Time (Dataset A):
Best insertion – Hash Table with Quadratic Probing



Linked Lists, Binary Trees, Hash Table with Quadratic Probing vs Time (Dataset A):
Best of Search - Hash Table with Quadratic Probing