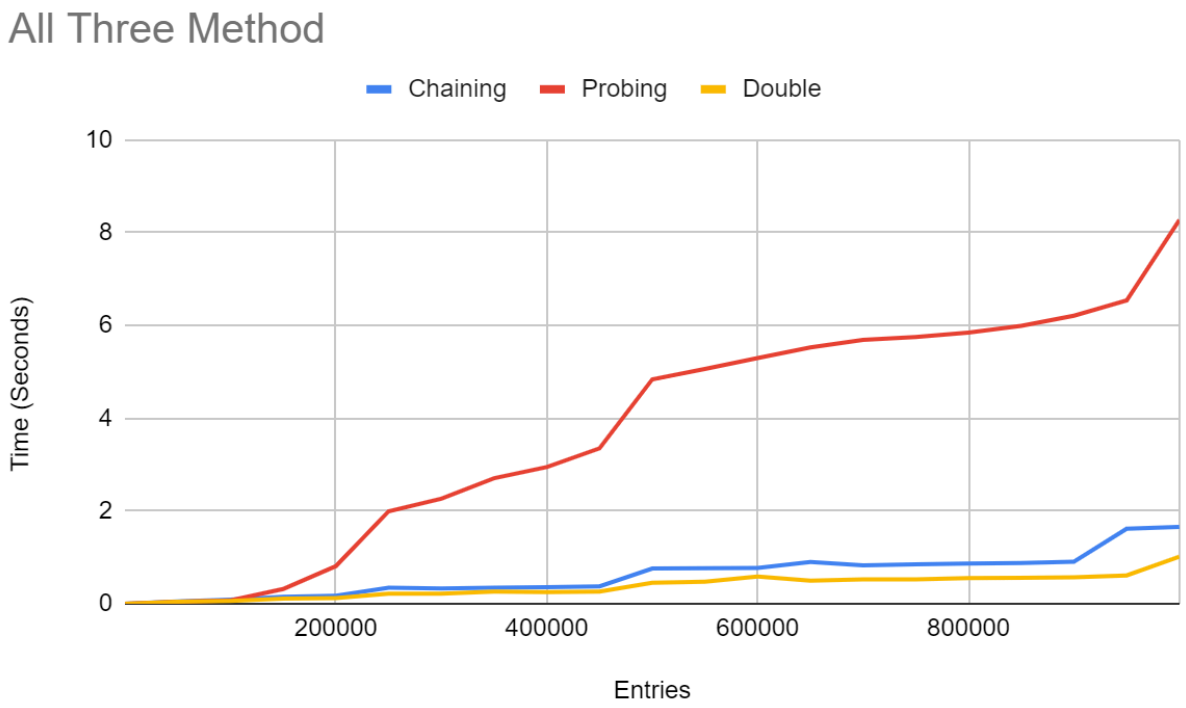
PA4 Report

Theoretical Statement:

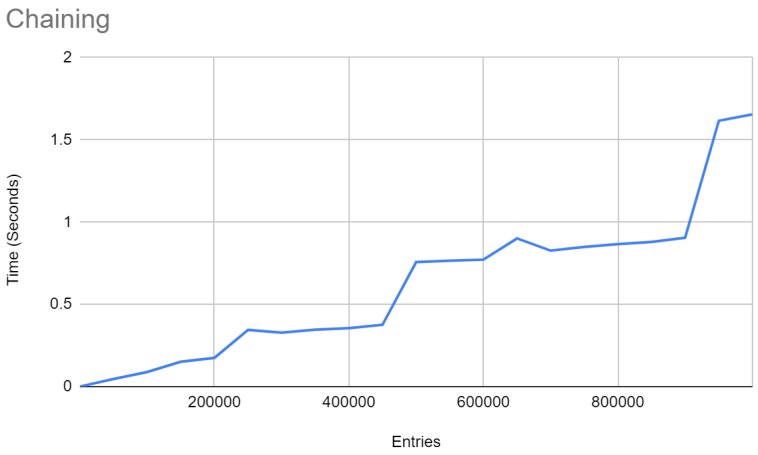
In this programming assignment, a hash table was implemented using 3 different strategies to deal with collisions. The first strategy was to have a linked list in each bucket so that the entry could be just added to the back of the list in cases where collision occurs. The second strategy was to use linear probing where the entry is added to the next available bucket after the assigned bucket in case a collision occurs. The final strategy was to use double hashing and this strategy is similar to the linear probing strategy, but the jump size to find the next available slot is determined by a second hash function. The first strategy's (separate chaining) insert function has a runtime complexity of O(1) and its delete function has a runtime complexity of O(N). The second and third strategies have an insert function with a runtime complexity of O(N) and a delete function with a runtime complexity of O(N). The amortized runtime for all the strategies is O(p) where p depends on the max load factor.

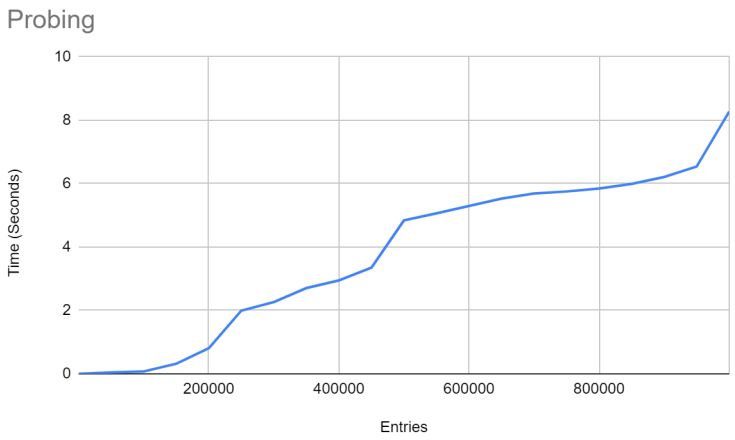
Experimental Analysis:

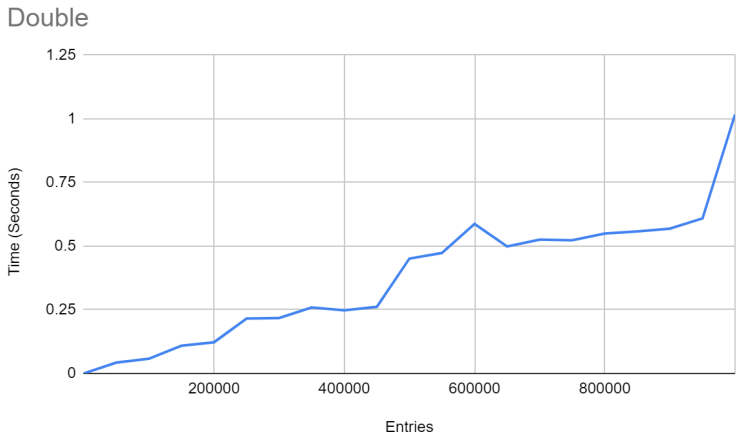
Graph with all Results:



Graphs With Individual Results:







Data Table:

Entries	Chaining	Probing	Double
10	2.14E-05	6.84E-06	7.05E-06
50010	0.0460046	0.0425027	0.0425827
100010	0.0884211	0.0713522	0.0577597
150010	0.150798	0.316918	0.108858
200010	0.173999	0.810683	0.122343
250010	0.344333	1.99268	0.215511
300010	0.328326	2.26195	0.217752
350010	0.345715	2.70512	0.259199
400010	0.355055	2.94668	0.247534
450010	0.375617	3.35086	0.261468
500010	0.756788	4.83775	0.451117
550010	0.76521	5.06309	0.472989
600010	0.771706	5.29464	0.587349
650010	0.900707	5.52768	0.498488
700010	0.826487	5.68937	0.525695
750010	0.848982	5.75223	0.52325
800010	0.8665	5.84427	0.549638
850010	0.878951	5.99111	0.557553
900010	0.904615	6.21041	0.568543
950010	1.61601	6.54042	0.60897
1000010	1.65472	8.27234	1.01664

Firstly, this experiment was conducted by inserting about 1 million entries (keys were unique and randomly generated) into 3 hash tables, one for each of the strategies, with an interval of 50000 entries. At each interval the time elapsed was recorded and written to a csv file in order to generate the graphs above.

According to the results, the implementation that has the best performing insert function is the double hashing method. The next best implementation was the chaining method and the worst implementation was the probing method which came in last with a considerably bigger margin.

Initially when there were low amounts of insertions being made, the insert function of the chaining method was performing worse than the probing method, but after 100,000 entries, the chaining method started having a lower runtime. The double hashing method always had a better runtime than the other 2 strategies regardless of the number of entries.

Discussion:

The results of the experiment were not as expected due to the fact that the chaining method was expected to have a runtime complexity of O(1) which is better when compared to the runtime complexity of the insert function of the double hashing method which is O(N), but the results from this experiment show that the double hashing method was more efficient than the chaining method in terms of the runtime. This might be due to the fact that the double hashing method stores the data contiguously which would mean that there would not be as many cache misses when trying to insert in a place because the cache fetches blocks of memory when accessing a particular part of the memory. On the other hand the chaining method cannot store its entries contiguously because the entries are stored as a linked list and each node in a linked list is stored as an individual data point. This would also mean that the chaining method would have a greater runtime when trying to find if a key already exists before inserting the entry into the hash table. The probing strategy had the worst runtime and this was also not expected due to the fact that both double hash and linear probing were expected to have the same results. A possible explanation for this might be that the data set (keys) favored double hashing rather than linear probing in this experiment and this might have led to primary clustering where the entries might have been placed in closeby buckets and hence having the greater jumping offset would have helped in improving the runtime. In conclusion, it can be said that the best performing hash table implementation was the double hashing strategy.