This is the design file for the different hash table algorithms. In this test run, four different algorithms were compared, linked, linear, quadratic, and double hash tables. In terms of implementation, linked list uses its own class to store data into separate nodes, while the other three hash tables are based on the same concept, only requiring that the hash function be changed. These three algorithms were modeled after a parent HashTable class. All HashTables then have their respective methods for insertion removal and finding invoked through a shared interface, Hash.

Below are the results of the complete working code for algorithms, linked, linear, quadratic and double hash tables. The number of duplicate removes generated by the system is also logged for clarification, though this time offset from duplicate remove is relative as the list of elements removed from each table is the same. Time is printed in seconds:

|  |  |
| --- | --- |
| Duplicate Removes: 1 Table Size: 10 Sample Size: 7 Elements Removed: 6, 1, 4, 2, 2 Linked Hash: 0.000043192  Linear Hash: 0.000045757  Quadratic Hash: 0.000028224  Double Hash: 0.000030362 | Duplicate Removes: 235 Table Size: 8209 Sample Size: 8192 Number Removed: 1000 Linked Hash: 0.004104449  Linear Hash: 0.115588269  Quadratic Hash: 0.098224969  Double Hash: 0.234960687 |

According to the results, using the original sample size, we can clearly observe that the linked list preforms significantly better than the other three hash functions. Consequently, this performance gain in utilizing linked list comes with the cost of space complexity, as each new element is inserted through instantiation of another object which contains the data within it.

On the up scaled sample size, the linear hash function performance becomes significantly worse than that of its counter parts, since it traverses, or probes, through each element individually. Having to do this on a much larger scale makes it exponentially slower than the other methods. Surprisingly the Quadratic Hash yielded the best results of the three open addressing hash algorithms. This boost in performance is more noticed in the larger sample set as well. I speculate that this is the case due to the higher chance that the double hash function will yield cases in which it probes through most of the table before coming to a point in which the algorithm can safely assume that no more open spots exist for the value that is being inserted, given the specific collision and insertion circumstances. Since the quadratic algorithm makes smaller jumps, this issue is minimized, though not completely reduced, with quadratic probing. On the contrary, these times can vary significantly based on the sample size and elements being inserted and removed. One cannot know for certain which algorithm will perform best for all general cases, and so the various algorithms are run through tests pertaining to the relevant sample data for the specific project. From there, one can only speculate which storage algorithm will be best for the project of discussion.