A. Introduction

This lab exercise involves the creation of hash tables with three different kinds of implementation: linear probing implementation, chaining implementation and tree map implementation.

B. Design Approach

A class StudentRecord is created with instance field names of first name, last name and ID, all String objects. The IDs of the objects are used as a form of identification.

An interface class called Roster housing methods used to interact with objects that store StudentRecord objects

Three classes of HashMaps that implement Roster are created. LinearHashRoster uses linear probing implementation, TreeRoster extends TreeMap<String, StudentRecord> and uses this class as its implementation. ChainHashRoster uses chaining method as its HashMap implementation.

An ExperimentController class measures the time performance of access methods and insert methods. These values are then graphed and compared.

C. Method

A .csv file containing the list of names of names and ID, an input data file in .txt format containing parameters for simulation and the name of the output file of the data is sent as inputs to the main function of ExperimentController.

Parameters run are in format:
Initial size Load factor Maximum chain for rehashing

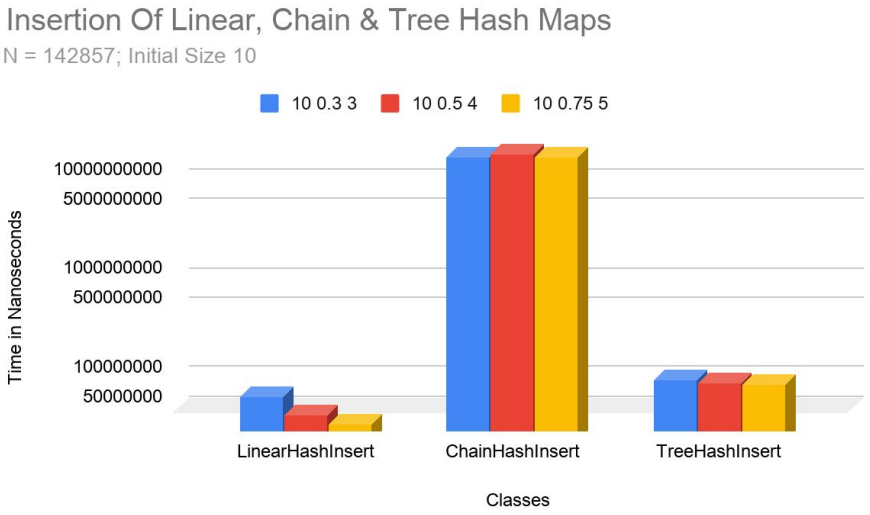
Parameters:
10 0.3 3
10 0.5 4
10 0.75 5
20 0.3 3
20 0.5 4
20 0.75 5
30 0.3 3
30 0.5 4
30 0.75 5
40 0.3 3

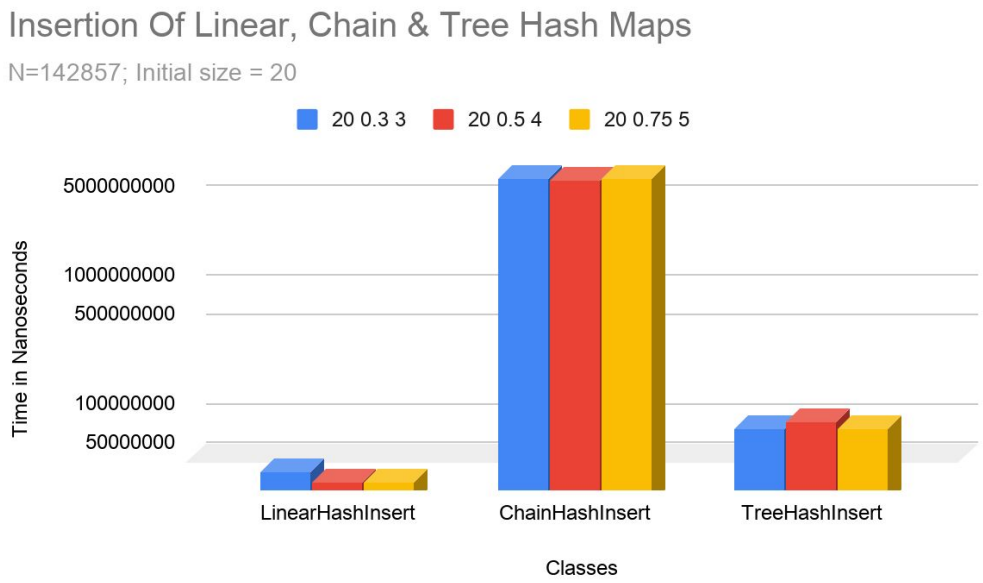
40 0.5 4
40 0.75 5

The Random class object chose to insert a value of 142857 StudentRoster objects into the three classes. Values of initial hash table of 10, 20 and 40 were compared to see the effects in runtime if initial hash size is doubled and quadrupled.

D. Data & Analysis

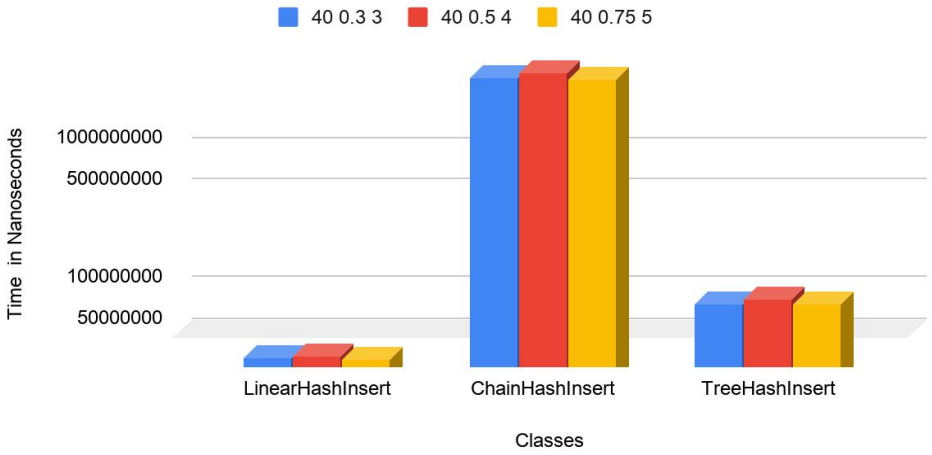
The graphs below show the time performance of insertion, successful and unsuccessful search methods of these classes.





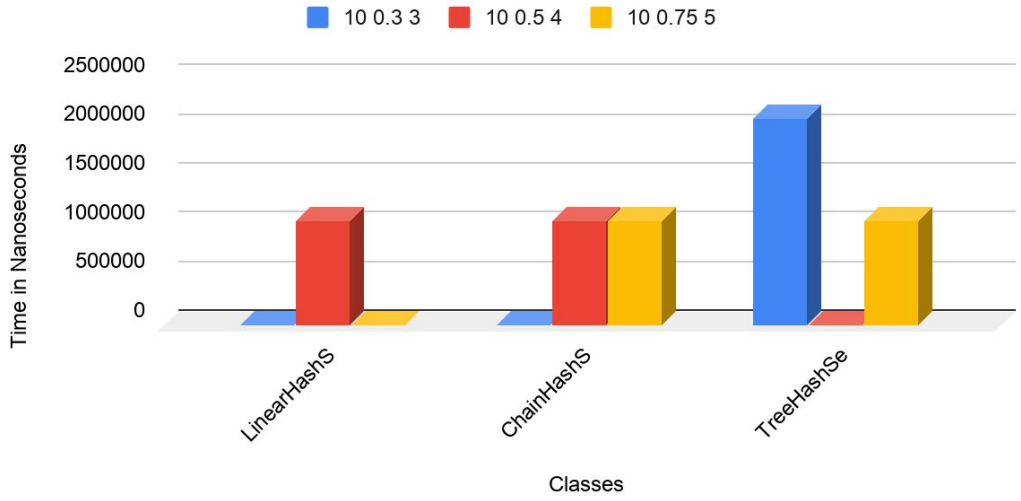
Insertion Of Linear, Chain & Tree Hash Maps

N=142857; Initial size = 40



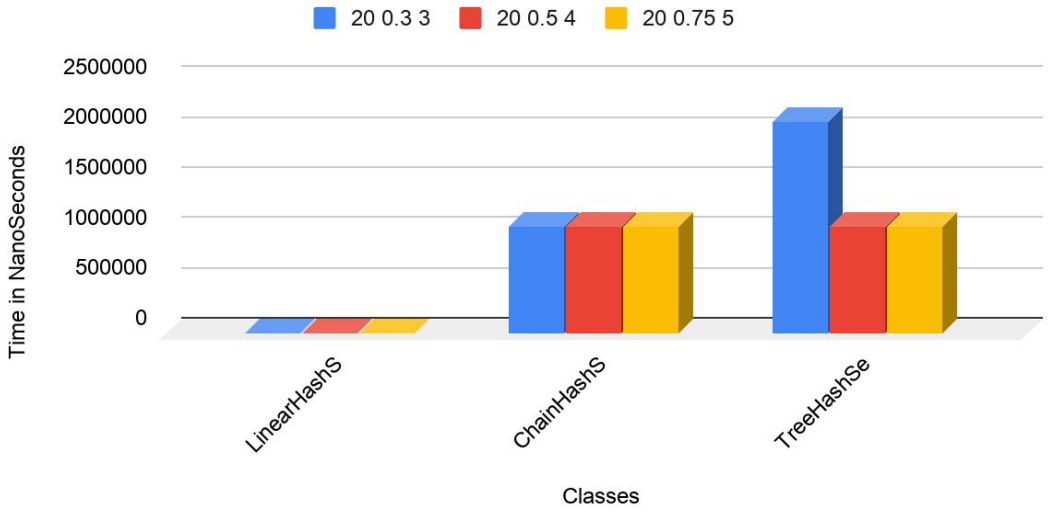
Search Successful For Linear, Chain and TreeMap Hash Table

N =142857, Initial size = 10



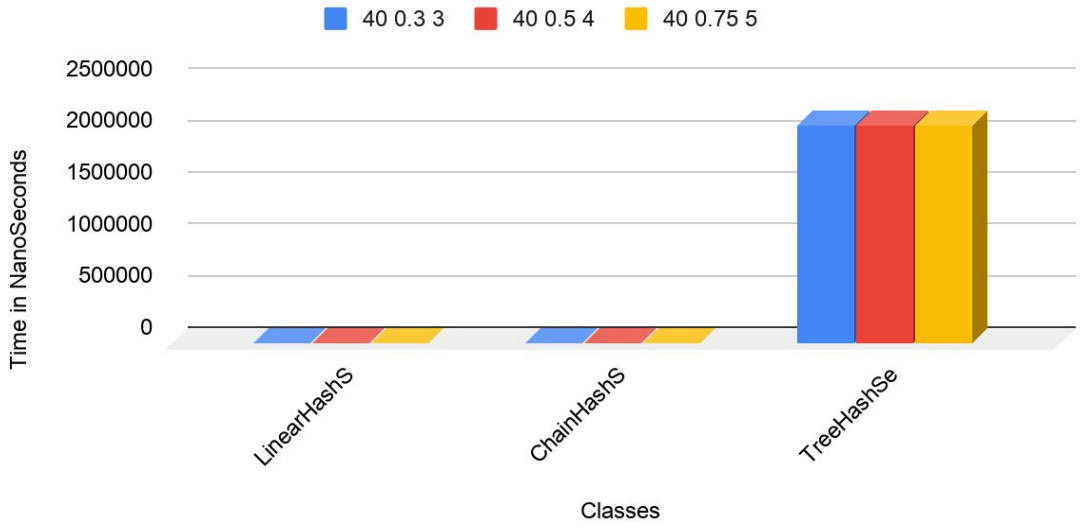
Search Successful For Linear, Chain and TreeMap Hash Table

N = 142857; Initial Size 20



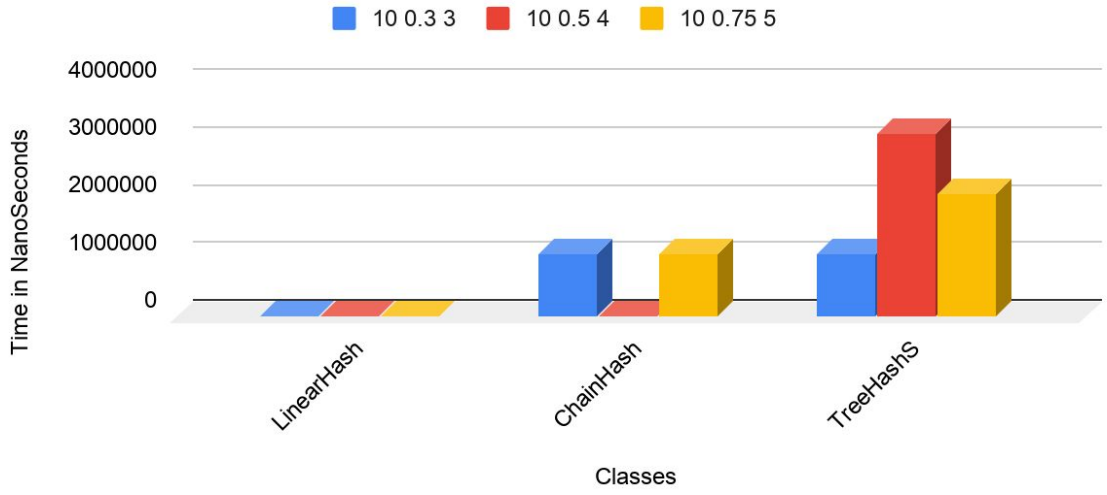
Search Successful For Linear, Chain and TreeMap Hash Table

N = 142857; Initial Size 40



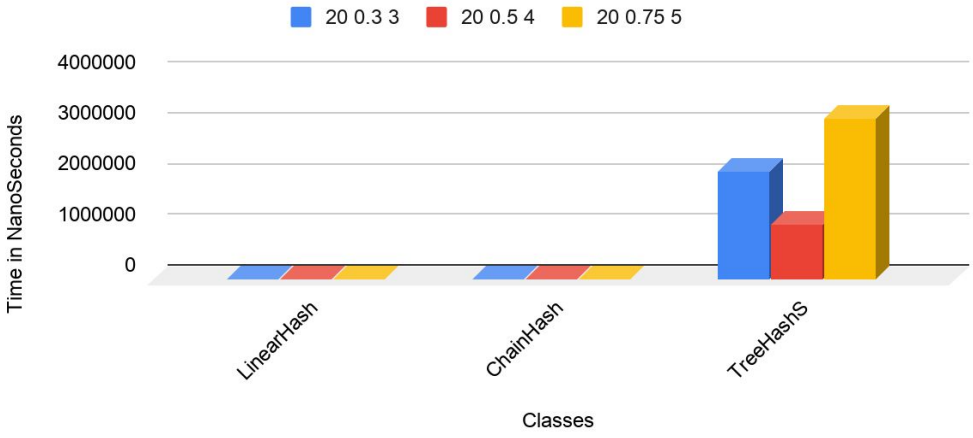
Search Unsuccessful For Linear, Chain and TreeMap Hash Table

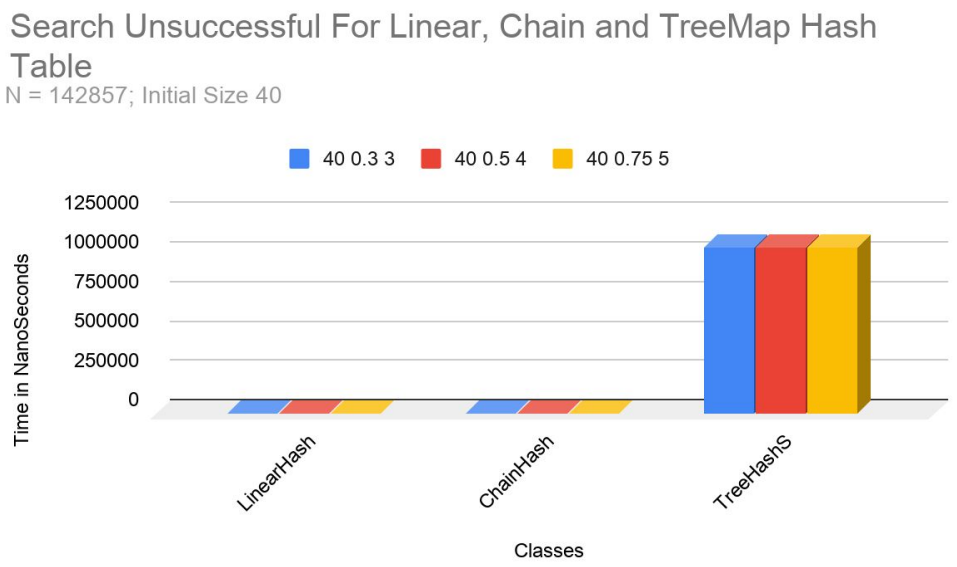
N = 142857; Initial Size 10



Search Unsuccessful For Linear, Chain and TreeMap Hash Table

N = 142857; Initial Size 20





For insertion of objects, the chaining implementation has the worst runtime. The runtime does not vary with maximum chaining length for rehashing. The linear probing implementation is the best performing, with performance increasing with increase in load factor. This reason might be due to fewer rehashes which take a lot of time. Increasing the initial size for chaining and linear probing implemented hash tables tend to increase time performance.

For successful searching of objects, it took a mode time of 0ns to obtain objects for linear probing implementation of hash table. For chaining, it took longer periods to access objects in hash table in some cases but some were of access time 0ns. The TreeMap implementation was the worst of them all. Same trend was noticed with increasing initial size of hash tables and increasing load factor and max-chaining length.

For unsuccessful search, it took a mode time of 0ns to obtain objects for linear probing implementation of hash table. For chaining, there was a mode of 0ns for performance time. The TreeMap implementation was the worst in unsuccessful search. Same trend was noticed with increasing initial size of hash tables and increasing load factor and max-chaining length

E. Conclusion

Linear probing implementation of hash table is the best when compared with chaining and TreeMap implementation. Access and insert time is the highest performing amongst the three kinds of implementation.