ASSIGNMENT 3: REPORT

The evaluation of linear probing,
quadratic probing, and sequence chaining Hash Tables.

Student Name: Steffi Baumgart
Student Number: BMGSTE001

5.5.1: Results

The data used for this assignment was a file *lexicon.txt* which contained 3739 English words. The number of probes for each method are counted for 3 different load factors:

A load factor of 0.5 would translate to a table size of 7481. A 0.75 load factor corresponds to a table size of 4987. Finally, a 3739 table size would be a full table (load factor of 1.)

TableSize = nextPrime(lexiconSize/loadFactor) is the formula used to obtain prime-number table sizes that correspond to the 3 load factors.

Load Test Results

- java –cp bin LoadTest <class name> <table size> <file name>

Table Size	Load Factor	Linear Probing Probes	Quadratic Probing Probes	Sequence Chaining Probes
7481	0.5	3349	3165	1682
4987	0.75	11598	7207	4082
3739	1	226514	Rehash	4639

Search Test Results

- java –cp bin SearchTest <class name> <table size> <file name> <sample size> <number of trials>
- NB: Sample size remains constant for every trial (100 words)
- NB: Number of trials remains constant for for all load factors and methods (1000 trials)

Linear Probing		Quadratic Probing		Sequence Chaining	
α	Ave Probes	α	Ave Probes	α	Ave Probes
0.5	83	0.5	54	0.5	37
0.75	341	0.75	169	0.75	51
1	97920	1	Rehash	1	66

5.5.2: Evaluation

Based on the results procured for the three hash table implementations, it can be deduced that for searching and inserting operations there is a directly proportional relationship between load factor and the number of probes: When load factor increases, the probe count increases.

Linear probing is quite slow, as it needs to search every array index after the index that the hash function provides if it is unable to insert/find. Primary clustering occurs at higher load factors, and therefore linear probing performance decreases almost exponentially.

Quadratic probing on the other hand completely fails for extreme load factors (the table needs to be rehashed), but it is better than linear probing for small values of α, since it will never search every index of the table.

If space is not an issue, which produces the fastest insert/search functions, and under what conditions?

For all load factors, SCHashTable is by far the fastest for insert and search operations .

Which technique generally performs best?

The probe counts of both LPHashTable and QPHashTable grow dramatically faster than that of the SCHashTable. While QPHashTable and LPHashTable are must slower at higher LoadFactors (sometimes requiring rehashing), SCHashTable works for load factors even greater than 1. However, SCHashTable requires O(n) complexity for traversal of the linked lists throughout the table, which, depending on the table size, takes up way more memory than the other 2 probing methods (e.g. SCHashTable with a small table size with a large amount of data that all hash to similar values.)

Allowing that the investigation has not considered the amounts of memory used, using estimates, which offers the best combination of speed and efficient storage?

Although Sequential Chaining achieves a higher overall speed out of the 3 implementations, Quadratic probing offers the best combination of speed and efficient storage, as it is faster than linear probing at low load factors, but does not take nearly as much memory as Sequential Chaining requires.