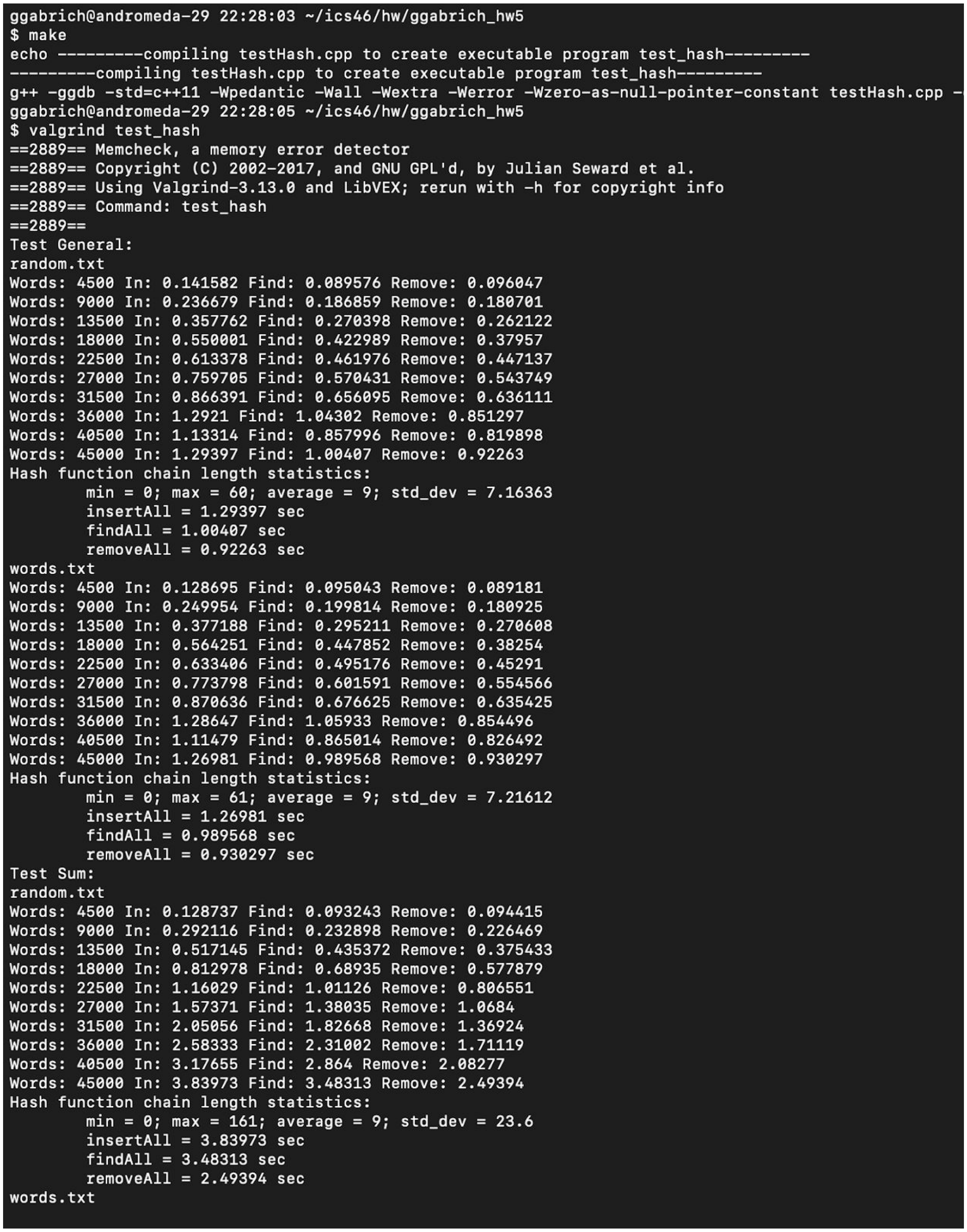
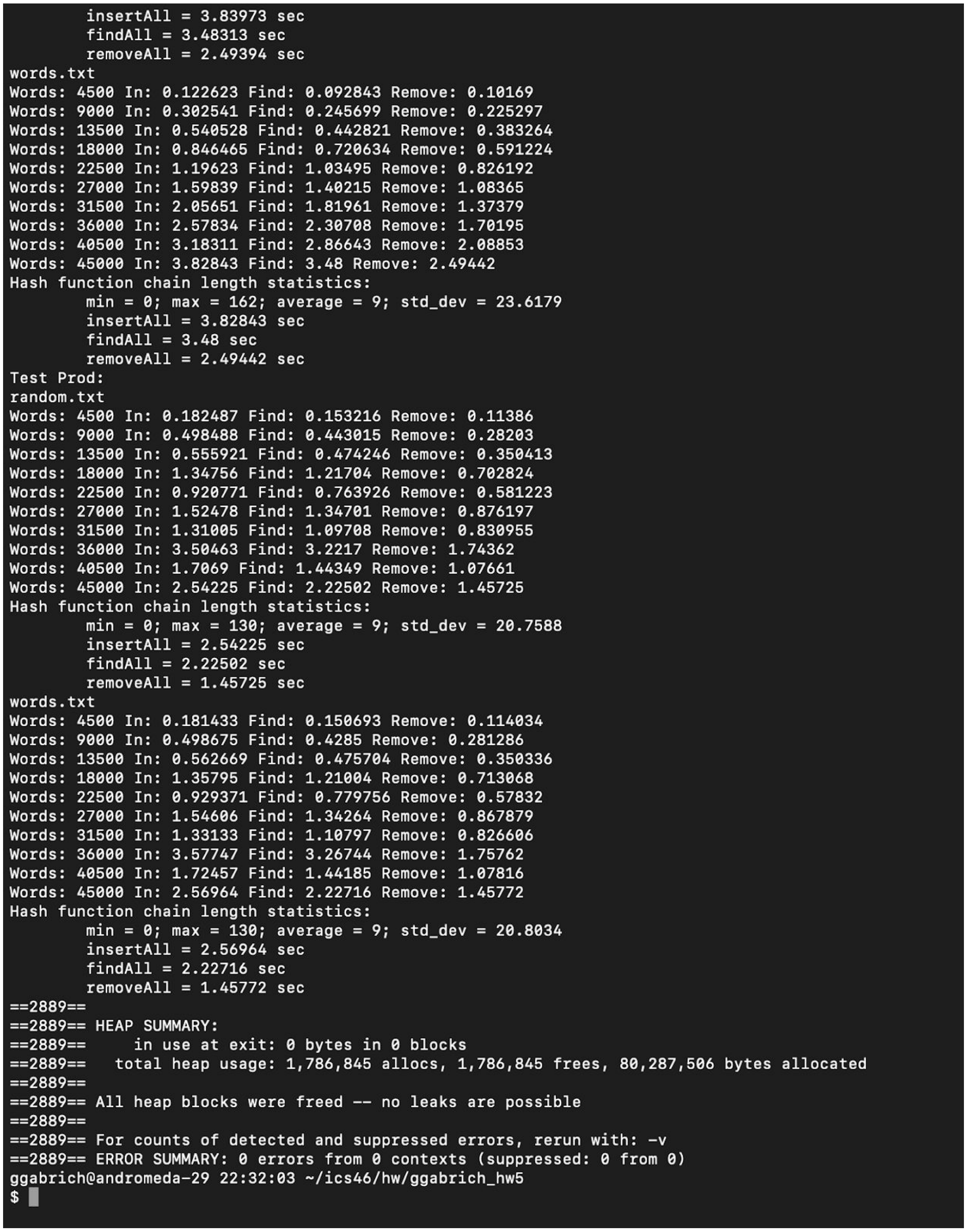
George Gabricht
Ggabrich - 56735102

ICS 46 - HW 5 Report



```
ggabrich@andromeda-29 22:28:03 ~/ics46/hw/ggabrich_hw5
$ make
echo ---------compiling testHash.cpp to create executable program test_hash---------
---------compiling testHash.cpp to create executable program test_hash---------
g++ -ggdb -std=c++11 -Wpedantic -Wall -Wextra -Werror -Wzero-as-null-pointer-constant testHash.cpp -
ggabrich@andromeda-29 22:28:05 ~/ics46/hw/ggabrich_hw5
$ valgrind test_hash
==2889== Memcheck, a memory error detector
==2889== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2889== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==2889== Command: test_hash
==2889==
Test General:
random.txt
Words: 4500 In: 0.141582 Find: 0.089576 Remove: 0.096047
Words: 9000 In: 0.236679 Find: 0.186859 Remove: 0.180701
Words: 13500 In: 0.357762 Find: 0.270398 Remove: 0.262122
Words: 18000 In: 0.550001 Find: 0.422989 Remove: 0.37957
Words: 22500 In: 0.613378 Find: 0.461976 Remove: 0.447137
Words: 27000 In: 0.759705 Find: 0.570431 Remove: 0.543749
Words: 31500 In: 0.866391 Find: 0.656095 Remove: 0.636111
Words: 36000 In: 1.2921 Find: 1.04302 Remove: 0.851297
Words: 40500 In: 1.13314 Find: 0.857996 Remove: 0.819898
Words: 45000 In: 1.29397 Find: 1.00407 Remove: 0.92263
Hash function chain length statistics:
        min = 0; max = 60; average = 9; std_dev = 7.16363
        insertAll = 1.29397 sec
        findAll = 1.00407 sec
        removeAll = 0.92263 sec
words.txt
Words: 4500 In: 0.128695 Find: 0.095043 Remove: 0.089181
Words: 9000 In: 0.249954 Find: 0.199814 Remove: 0.180925
Words: 13500 In: 0.377188 Find: 0.295211 Remove: 0.270608
Words: 18000 In: 0.564251 Find: 0.447852 Remove: 0.38254
Words: 22500 In: 0.633406 Find: 0.495176 Remove: 0.45291
Words: 27000 In: 0.773798 Find: 0.601591 Remove: 0.554566
Words: 31500 In: 0.870636 Find: 0.676625 Remove: 0.635425
Words: 36000 In: 1.28647 Find: 1.05933 Remove: 0.854496
Words: 40500 In: 1.11479 Find: 0.865014 Remove: 0.826492
Words: 45000 In: 1.26981 Find: 0.989568 Remove: 0.930297
Hash function chain length statistics:
        min = 0; max = 61; average = 9; std_dev = 7.21612
        insertAll = 1.26981 sec
        findAll = 0.989568 sec
        removeAll = 0.930297 sec
Test Sum:
random.txt
Words: 4500 In: 0.128737 Find: 0.093243 Remove: 0.094415
Words: 9000 In: 0.292116 Find: 0.232898 Remove: 0.226469
Words: 13500 In: 0.517145 Find: 0.435372 Remove: 0.375433
Words: 18000 In: 0.812978 Find: 0.68935 Remove: 0.577879
Words: 22500 In: 1.16029 Find: 1.01126 Remove: 0.806551
Words: 27000 In: 1.57371 Find: 1.38035 Remove: 1.0684
Words: 31500 In: 2.05056 Find: 1.82668 Remove: 1.36924
Words: 36000 In: 2.58333 Find: 2.31002 Remove: 1.71119
Words: 40500 In: 3.17655 Find: 2.864 Remove: 2.08277
Words: 45000 In: 3.83973 Find: 3.48313 Remove: 2.49394
Hash function chain length statistics:
        min = 0; max = 161; average = 9; std_dev = 23.6
        insertAll = 3.83973 sec
        findAll = 3.48313 sec
        removeAll = 2.49394 sec
words.txt
```

George Gabricht
Ggabrich - 56735102



```
        insertAll = 3.83973 sec
        findAll = 3.48313 sec
        removeAll = 2.49394 sec
words.txt
Words: 4500 In: 0.122623 Find: 0.092843 Remove: 0.10169
Words: 9000 In: 0.302541 Find: 0.245699 Remove: 0.225297
Words: 13500 In: 0.540528 Find: 0.442821 Remove: 0.383264
Words: 18000 In: 0.846465 Find: 0.720634 Remove: 0.591224
Words: 22500 In: 1.19623 Find: 1.03495 Remove: 0.826192
Words: 27000 In: 1.59839 Find: 1.40215 Remove: 1.08365
Words: 31500 In: 2.05651 Find: 1.81961 Remove: 1.37379
Words: 36000 In: 2.57834 Find: 2.30708 Remove: 1.70195
Words: 40500 In: 3.18311 Find: 2.86643 Remove: 2.08853
Words: 45000 In: 3.82843 Find: 3.48 Remove: 2.49442
Hash function chain length statistics:
        min = 0; max = 162; average = 9; std_dev = 23.6179
        insertAll = 3.82843 sec
        findAll = 3.48 sec
        removeAll = 2.49442 sec
Test Prod:
random.txt
Words: 4500 In: 0.182487 Find: 0.153216 Remove: 0.11386
Words: 9000 In: 0.498488 Find: 0.443015 Remove: 0.28203
Words: 13500 In: 0.555921 Find: 0.474246 Remove: 0.350413
Words: 18000 In: 1.34756 Find: 1.21704 Remove: 0.702824
Words: 22500 In: 0.920771 Find: 0.763926 Remove: 0.581223
Words: 27000 In: 1.52478 Find: 1.34701 Remove: 0.876197
Words: 31500 In: 1.31005 Find: 1.09708 Remove: 0.830955
Words: 36000 In: 3.50463 Find: 3.2217 Remove: 1.74362
Words: 40500 In: 1.7069 Find: 1.44349 Remove: 1.07661
Words: 45000 In: 2.54225 Find: 2.22502 Remove: 1.45725
Hash function chain length statistics:
        min = 0; max = 130; average = 9; std_dev = 20.7588
        insertAll = 2.54225 sec
        findAll = 2.22502 sec
        removeAll = 1.45725 sec
words.txt
Words: 4500 In: 0.181433 Find: 0.150693 Remove: 0.114034
Words: 9000 In: 0.498675 Find: 0.4285 Remove: 0.281286
Words: 13500 In: 0.562669 Find: 0.475704 Remove: 0.350336
Words: 18000 In: 1.35795 Find: 1.21004 Remove: 0.713068
Words: 22500 In: 0.929371 Find: 0.779756 Remove: 0.57832
Words: 27000 In: 1.54606 Find: 1.34264 Remove: 0.867879
Words: 31500 In: 1.33133 Find: 1.10797 Remove: 0.826606
Words: 36000 In: 3.57747 Find: 3.26744 Remove: 1.75762
Words: 40500 In: 1.72457 Find: 1.44185 Remove: 1.07816
Words: 45000 In: 2.56964 Find: 2.22716 Remove: 1.45772
Hash function chain length statistics:
        min = 0; max = 130; average = 9; std_dev = 20.8034
        insertAll = 2.56964 sec
        findAll = 2.22716 sec
        removeAll = 1.45772 sec
==2889==
==2889== HEAP SUMMARY:
==2889==     in use at exit: 0 bytes in 0 blocks
==2889==   total heap usage: 1,786,845 allocs, 1,786,845 frees, 80,287,506 bytes allocated
==2889==
==2889== All heap blocks were freed -- no leaks are possible
==2889==
==2889== For counts of detected and suppressed errors, rerun with: -v
==2889== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
ggabrich@andromeda-29 22:32:03 ~/ics46/hw/ggabrich_hw5
$ 
```

George Gabricht
Ggabrich - 56735102

```
// Worst Case: O(L)
        // Typical Case: O(L)
        static TableNode * recursiveFind(string key, TableNode *cur) {
            if ((cur == nullptr) || (cur->key == key)) { // 1
                return cur; // 1
            } else {
                return recursiveFind(key, cur->next); // L
            }
        }

// Worst Case: O(L)
        // Typical Case: O(L)
        static TableNode * insert(string key, unsigned int val, TableNode *cur) {
            TableNode *result = recursiveFind(key, cur); // L
            if (result == nullptr) { // 1
                cur = new TableNode(key, val, cur); // 1
            } else {
                result->value = val; // 1
            }
            return cur; // 1
        }

 // Worst Case: O(L)
        // Typical Case: O(L)
        static TableNode * remove(string key, TableNode *cur) {
            TableNode *temp; // 1
            if (cur == nullptr) { // 1
                return nullptr; // 1
            } else if (cur->key == key) { // 1
                temp = cur->next; // 1
                delete cur; // 1
                return temp; // 1
            }
            for (TableNode *ndx = cur; ndx->next != nullptr; ndx = ndx->next) { // L
                if (ndx->next->key == key) { // 1
                    temp = ndx->next; // 1
                    ndx->next = temp->next; // 1
                    delete temp; // 1
                    return cur; // 1
                }
            }
            return cur; // 1
        }
```

George Gabricht
Ggabrich - 56735102

```
// Worst Case: O(1)
          // Typical Case: O(1)
          static int * getValue(TableNode *cur) { // O(1)
               if (cur) { // 1
                    return &cur->value; // 1
               } else {
                    return &not_found; // 1
               }
          }

// Worst Case: O(N) When N == L
     // Typical Case: O(L)
     void insert(string key, unsigned int value = 1) {
          int ndx = hash(key); // 1
          table[ndx] = TableNode::insert(key, value, table[ndx]); // L
          chained_list_lengths[ndx] = TableNode::listLength(table[ndx]);
     }

// Worst Case: O(N) When N == L
     // Typical Case: O(L)
     int & operator[](string key) {
          TableNode *result = TableNode::recursiveFind(key, table[hash(key)]); // L
          if (result == nullptr) { // 1
               if (not_found >= -1) { // 1
                    not_found = -5; // 1
               }
               return not_found; // 1
          } else {
               return (int &) *TableNode::getValue(result); // 1
          }
     }

// Worst Case: O(N) When N == L
     // Typical Case: O(L)
     int find(string key) {
          TableNode *result = TableNode::recursiveFind(key, table[hash(key)]); // L
          return *TableNode::getValue(result); // 1
     }

// Worst Case: O(N) When N == L
     // Typical Case: O(L)
     void remove(string key) {
          int hsh = hash(key); // 1
```

```
		table[hsh] = TableNode::remove(key, table[hsh]); // L
		chained_list_lengths[hsh] = TableNode::listLength(table[hsh]);
	}

// Worst Case: O(N^2) When N == L
// Typical Case: O(N)
double insertAll(ChainedHashTable & tbl, const char* inputFileName, int numWords) {
	ifstream ifile(inputFileName); // 1
	if (!ifile.is_open()) { // 1
		cout << "File Error!" << endl; // 1
		exit(-1); // 1
	}
	string curLine; // 1
	int ndx = 0; // 1
	Timer t;
	double eTime;
	t.start();
	while (getline(ifile, curLine)) { // N
		if (++ndx > numWords) { // 1
			break; // 1
		}
		tbl.insert(curLine, 1); // N/L
		/*if (ndx % 1000 == 0) {
			cout << "i";
		}*/
	}
	t.elapsedUserTime(eTime);
	// cout << endl; // 1
	ifile.close(); // 1
	return eTime;
}

// Worst Case: O(N^2) When N == L
// Typical Case: O(N)
double findAll(ChainedHashTable & tbl, const char* inputFileName, int numWords) {
	ifstream ifile(inputFileName); // 1
	if (!ifile.is_open()) { // 1
		cout << "File Error!" << endl; // 1
		exit(-1); // 1
	}
	string curLine; // 1
	int ndx = 0; // 1
	Timer t;
```

```
        double eTime;
        t.start();
        while (getline(ifile, curLine)) { // N
                if (++ndx > numWords) { // 1
                        break; // 1
                } else if (tbl.find(curLine) < 0) { // N/L
                        cout << "Not Found: " << curLine << endl; // 1
                }
                /*if (ndx % 1000 == 0) {
                        cout << "f";
                }*/
        }
        t.elapsedUserTime(eTime);
        // cout << endl; // 1
        ifile.close(); // 1
        return eTime;
}

// Worst Case: O(N^2) When N == L
// Typical Case: O(N)
double removeAll(ChainedHashTable & tbl, const char* inputFileName, int numWords) {
        ifstream ifile(inputFileName); // 1
        if (!ifile.is_open()) { // 1
                cout << "File Error!" << endl; // 1
                exit(-1); // 1
        }
        string curLine; // 1
        int ndx = 0; // 1
        Timer t;
        double eTime;
        t.start();
        while (getline(ifile, curLine)) { // N
                if (++ndx > numWords) { // 1
                        break; // 1
                }
                tbl.remove(curLine); // N/L
                /*if (ndx % 1000 == 0) {
                        cout << "r";
                }*/
        }
        t.elapsedUserTime(eTime);
        // cout << endl; // 1
        ifile.close(); // 1
```

```
	return eTime;
}

//an abstract struct to parent your various hasher classes//
struct Hasher {
	virtual int hash(string s, int N) = 0;
};

//your first working hashing class//
struct GeneralStringHasher: Hasher {
	// Worst Case: O(1)
	// Typical Case: O(1)
	virtual int hash(string key, int N) override {
		const unsigned shift = 6; // 1
		const unsigned zero = 0; // 1
		unsigned mask = ~zero >> (32 - shift); // 1
		unsigned result = 0; // 1
		int len = min((int)key.size(), 6); // 1
		for (int ndx = 0; ndx < len; ndx++) { // 1
			result = (result << shift) | (key[ndx] & mask); // 1
		}
		return result % N; // 1
	}
};

//A rough idea of these Hashers - //
//some modifications may be needed//
struct SumHasher : Hasher {
	// Worst Case: O(1)
	// Typical Case: O(1)
	virtual int hash(string key, int N) override {
		int result = 0, len = key.size(); // O(1)
		for (int ndx = 0; ndx < len; ndx++) // O(1)
			result += key[ndx]; // O(1)
		return abs(result) % N; // O(1)
	}
};

struct ProdHasher : Hasher {
	// Worst Case: O(1)
	// Typical Case: O(1)
	int hash(string key, int N) {
		int result = 1, len = key.size(); // 1
```

George Gabricht
Ggabrich - 56735102

```
        for (int ndx = 0; ndx < len; ndx++) // 1
            result *= key[ndx]; // 1
        return abs(result) % N; // 1
    }
};

// Worst Case: O(1)
    // Typical Case: O(1)
    int hash(string key) {
        return hashr.hash(key, buckets); // 1
    }
```

Hash Functions

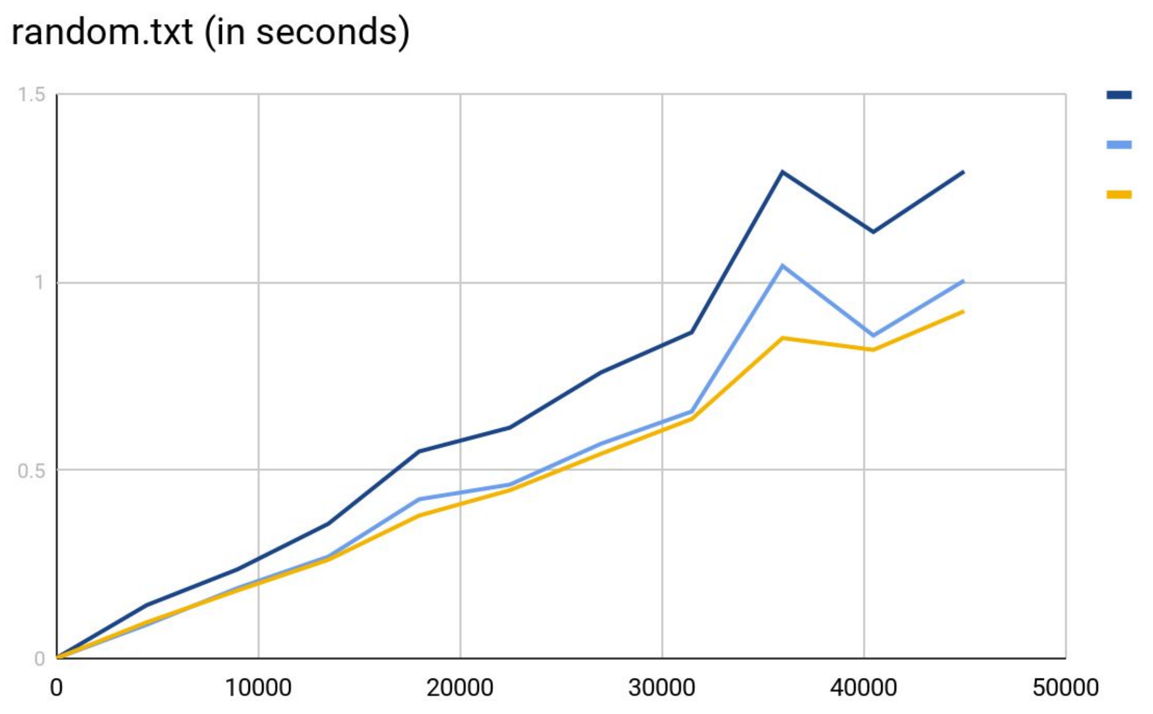
```
Hash function 1 (general) chain length statistics:
      min = 0; max = 60; average = 9; std_dev = 7.16363
      insertAll = 1.29397 sec
      findAll = 1.00407 sec
      removeAll = 0.92263 sec

Hash function 2 (sum) chain length statistics:
      min = 0; max = 161; average = 9; std_dev = 23.6
      insertAll = 3.83973 sec
      findAll = 3.48313 sec
      removeAll = 2.49394 sec

Hash function 3 (prod) chain length statistics:
      min = 0; max = 130; average = 9; std_dev = 20.7588
      insertAll = 2.54225 sec
      findAll = 2.22502 sec
      removeAll = 1.45725 sec
```

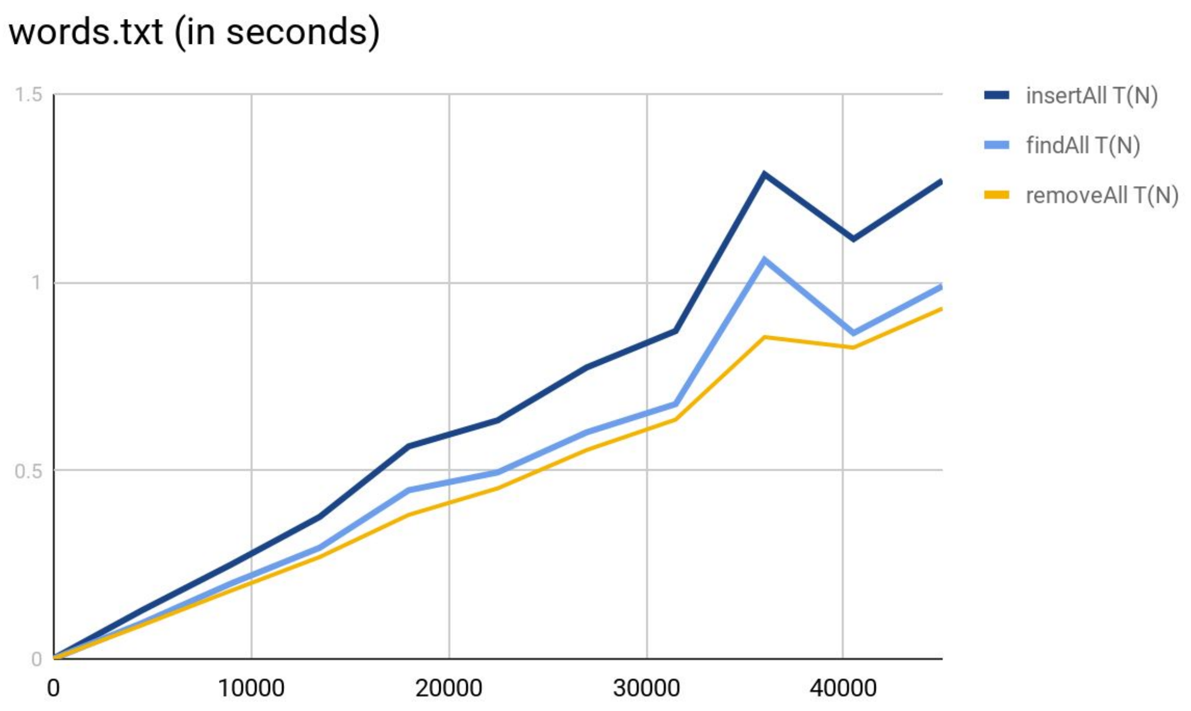
George Gabricht
Ggabrich - 56735102

Random.txt (in seconds)			
N (# inputs)	insertAll T(N)	findAll T(N)	removeAll T(N)
4500	0.1416	0.0896	0.0960
9000	0.2367	0.1869	0.1807
13500	0.3578	0.2704	0.2621
18000	0.5500	0.4230	0.3796
22500	0.6134	0.4620	0.4471
27000	0.7597	0.5704	0.5437
31500	0.8664	0.6561	0.6361
36000	1.2921	1.0430	0.8513
40500	1.1331	0.8580	0.8199
45000	1.2940	1.0041	0.9226



George Gabricht
Ggabrich - 56735102

Words.txt (in seconds)			
N (# inputs)	insertAll T(N)	findAll T(N)	removeAll T(N)
4500	0.1287	0.0950	0.0892
9000	0.2500	0.1998	0.1809
13500	0.3772	0.2952	0.2706
18000	0.5643	0.4479	0.3825
22500	0.6334	0.4952	0.4529
27000	0.7738	0.6016	0.5546
31500	0.8706	0.6766	0.6354
36000	1.2865	1.0593	0.8545
40500	1.1148	0.8650	0.8265
45000	1.2698	0.9896	0.9303


words.txt (in seconds)
insertAll T(N)
findAll T(N)
removeAll T(N)