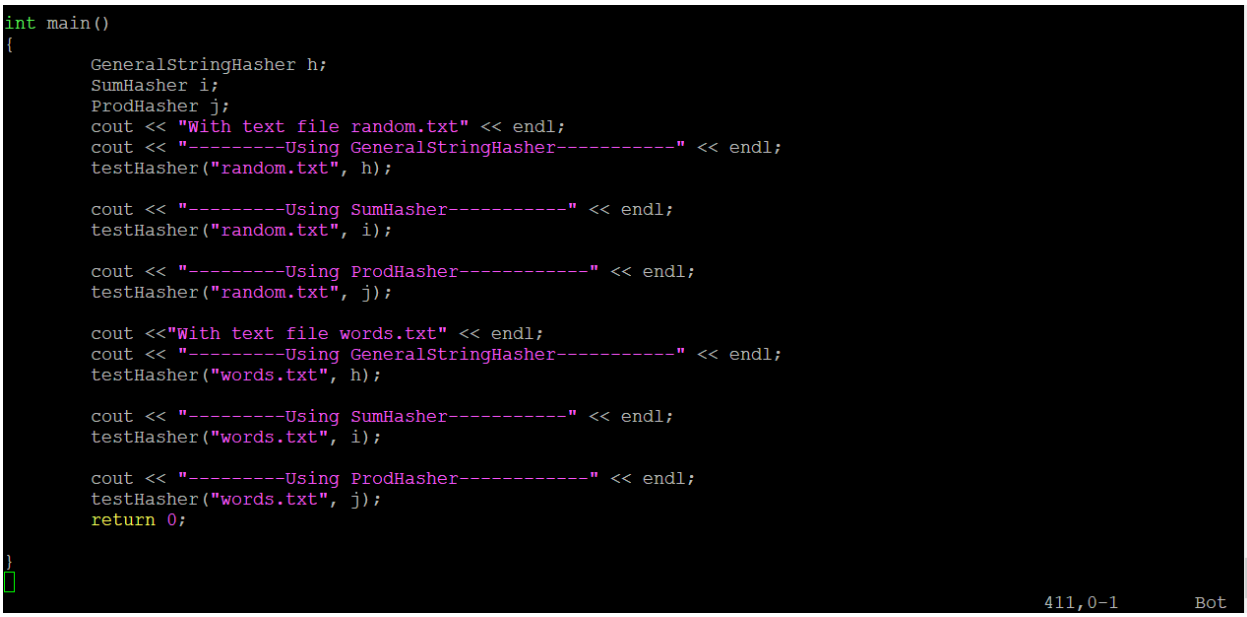
Raul Cervantes

77825705

HomeWork 5

Start of Report



```
int main()
{
        GeneralStringHasher h;
        SumHasher i;
        ProdHasher j;
        cout << "With text file random.txt" << endl;
        cout << "---------Using GeneralStringHasher-----------" << endl;
        testHasher("random.txt", h);

        cout << "---------Using SumHasher-----------" << endl;
        testHasher("random.txt", i);

        cout << "---------Using ProdHasher------------" << endl;
        testHasher("random.txt", j);

        cout <<"With text file words.txt" << endl;
        cout << "---------Using GeneralStringHasher-----------" << endl;
        testHasher("words.txt", h);

        cout << "---------Using SumHasher-----------" << endl;
        testHasher("words.txt", i);

        cout << "---------Using ProdHasher------------" << endl;
        testHasher("words.txt", j);
        return 0;

}
                                                                411,0-1       Bot
```

Above is a pic of the main function.

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

struct GeneralStringHasher: Hasher
{
        int hash(string key, int N)//O(N)
        {
                //use the "General Hash Function for Strings"
                //algrorithim from lecture
                const unsigned shift = 6;
                const unsigned zero = 0;
                int size = key.size();
                unsigned mask = ~zero >> (32-shift); // low 6 bits on
                unsigned result = 0;
                int len = min(size, 6);
                for (int i = 0; i < len; i++)
                {
                        result = (result << shift) | (key[i] & mask);
                }
                return result % N;
        }
};

struct SumHasher: Hasher
{
        int hash(string key, int N)//O(N)
        {
                int result = 0;
                for(unsigned int i = 0; i<key.size(); ++i)
                        result += key[i];
                return abs(result)%N;
        }
};

struct ProdHasher: Hasher
{
        int hash(string key, int N)//O(N)
        {
                int result = 1;
                for(unsigned int i = 0; i<key.size(); ++i)
                        result *= key[i];
                return abs(result)%N;
        }
};
```

Above are the hash functions with their typical time complexities in Big-O.

```
class ChainedHashTable
{
private:
        //data members
        //hint: have a data member that is a Hasher REFERENCE
        //(make sure you understand why!!!)

        struct ListNode
        {
                string key;
                int value;
                ListNode* next;
                ListNode(string new_key, int new_value, ListNode* new_next)
                        : key(new_key), value(new_value), next(new_next) {}

                ListNode(const ListNode& ln) = delete;

                ListNode& operator = (const ListNode& ln) = delete;

                static ListNode* insert(string key, int val, ListNode* L)//O(1)
                {
                        return new ListNode(key, val, L);
                }

                static ListNode* find(string key, ListNode* L)//O(N)
                {
                        for(ListNode* p = L; p!=nullptr; p = p->next)
                        {
                                if(p->key == key)
                                {
                                        return p;
                                }
                        }
                        return nullptr;
                }

                static ListNode* remove(string key, ListNode* L)//O(N)
                {
                        ListNode* p = L;
                        if(p == nullptr)
                        {
                                return nullptr;
                        }
                        if(p->next == nullptr)
                        {
                                if(p->key == key)
                                {
                                        delete p;
                                        return nullptr;
                                }
                                return L;
                        }
                        ListNode* prev = p;
                        while(p != nullptr)
                        {
                                p = p->next;
                                if(p == nullptr)
                                {
                                        return L;
                                }
                                if(p->key == key)
                                {
                                        prev->next = p->next;
                                        delete p;
```

```
                                        return L;
                                }
                                prev = prev->next;
                        }
                        return L;
                }

                static ListNode* copy(ListNode* L)//O(N)
                {
                        ListNode* result = nullptr;
                        for(ListNode* p = L; p!=nullptr; p = p->next)
                        {
                                result = new ListNode(p->key, p->value, result);
                        }
                        return result;
                }
        };

        ListNode** buf;
        int capacity;
        Hasher& myHash;
        int size;
        int* stat;
public:
        ChainedHashTable(int cap, Hasher& myHasher)//O(N)
                : buf(new ListNode*[cap]), capacity(cap), myHash(myHasher), size(0), stat(new int [cap])
        {
                for(int i = 0; i < capacity; ++i)
                {
                        buf[i] = nullptr;
                        stat[i] = 0;
                }
        }


        ChainedHashTable(const ChainedHashTable& C)//O(N^2)
                :  capacity(C.capacity), myHash(C.myHash), size(C.size), stat(C.stat)
        {
                buf = new ListNode*[capacity];
                stat= new int[capacity];

                for(int i = 0; i < capacity; ++i)
                {
                        stat[i] = C.stat[i];
                        buf[i] = ListNode::copy(C.buf[i]);
                }
        }

        void insert(string key, int val)//O(N)
        {
                int index = myHash.hash(key, capacity);
                ListNode* temp = ListNode::insert(key, val, buf[index]);
                ++stat[index];
                ++size;
                buf[index] = temp;
        }

        bool find(string key)//O(N)
        {
                int index = myHash.hash(key, capacity);
                ListNode* temp = ListNode::find(key, buf[index]);
                if(temp == nullptr)
                {
                        return 0;
                                                                        187,3-24      34%
```

```
        }
        return 1;
}

void remove(string key)//O(N)
{
        int index = myHash.hash(key, capacity);
        ListNode* temp = ListNode::remove(key, buf[index]);
        buf[index] = temp;
}

int& operator[] (string key)//O(N)
{
        int index = myHash.hash(key, capacity);
        ListNode* temp = ListNode::find(key, buf[index]);
        if(temp == nullptr)
        {
                throw;
        }
        return temp->value;
}

int max()//O(N)
{
        int result = 0;
        for(int i = 0; i < capacity; ++i)
        {
                if(stat[i] > result)
                {
                        result = stat[i];
                }
        }
        return result;
}

int min()//O(N)
{
        int smol = stat[0];
        for(int i = 0; i < capacity; ++i)
        {
                if(stat[i] < smol)
                {
                        smol = stat[i];
                }
        }
        return smol;
}

double avg()//O(1)
{
        return size*1.0/capacity*1.0;
}

double SD()//O(N)
{
        double variance_sum = 0.0, variance = 0.0, std_dev = 0.0;
        for(int i = 0; i < capacity; ++i)
        {
                variance_sum += (stat[i] - avg())*(stat[i] - avg());
        }
        variance = variance_sum / capacity;
        std_dev = sqrt(variance);
        return std_dev;
}
                                                                    251,2-9       51%
```

```
        ChainedHashTable& operator = (const ChainedHashTable& C) = delete;

        ~ChainedHashTable()//O(N^2)
        {
                for(int i = 0; i < capacity; ++i)
                {
                        ListNode* knob = buf[i];
                        while(knob != nullptr)
                        {
                                ListNode* temp = knob;
                                knob = knob->next;
                                delete temp;
                        }
                }
                delete[] buf;
                delete[] stat;
        }

};
```

The above 4 images is the ChainedHashTable class with their typical time complexities.

```
void testConstructor(Hasher& hasher)//O(N^2)
{
        ChainedHashTable A(5000, hasher);
        //create an empty ChainedHashTable object
}

void testCopyConstructor(Hasher& hasher)//O(N^2)
{
        ChainedHashTable A(5000, hasher);
        ChainedHashTable B(A);
        //create a ChainedHashTable object
        //create a second object as a copy of the first object
}

void insertAll(ifstream& file, ChainedHashTable& C, int NWords)//O(N^2)
{
        string word;
        for(int i = 0; i < NWords; file >> word, ++i)
        {
                C.insert(word, 1);
        }
}

void findAll(ifstream& file, ChainedHashTable& C, int NWords) //O(N^2)
{
        string word;
        for(int i = 0; i < NWords; file >> word, ++i)
        {
                C.find(word);
        }
}

void removeAll(ifstream& file, ChainedHashTable& C, int NWords)//O(N^2)
{

        string word;
        for(int i = 0; i < NWords; file >> word, ++i)
        {
                C.remove(word);
        }
}
```

Above are the test functions with their typical time complexities.

```
#define NPartitions 10
#define NSamples 45392
void measure(string file_name, int NWords, Hasher& hasher)
{
        ifstream in(file_name);
        ChainedHashTable myTable(5000, hasher);
        Timer t1, t2, t3;
        double t1Time, t2Time, t3Time;

        t1.start();
        insertAll(in, myTable, NWords);
        t1.elapsedUserTime(t1Time);

        in.clear();
        in.seekg(0, ios::beg);

        t2.start();
        findAll(in, myTable, NWords);
        t2.elapsedUserTime(t2Time);

        in.clear();
        in.seekg(0, ios::beg);
        t3.start();
        removeAll(in, myTable, NWords);
        t3.elapsedUserTime(t3Time);

        in.close();
        cout << "Using " << NWords << " words" << endl;
        cout << "min = " << myTable.min() << "; ";
        cout << "max = " << myTable.max() << "; ";
        cout << "average = " << myTable.avg() << "; ";
        cout << "std_dev = " << myTable.SD() << "; " << endl;
        //cout << "Using " << NWords << " words" << endl;
        cout << "insertAll: " << t1Time << endl;
        cout << "findAll: " << t2Time << endl;
        cout << "removeAll: " << t3Time << endl;
        cout << endl;
}
//overall tester function

void testHasher(char const* inputFileName, Hasher& hasher)
{

        //cout << inputFileName << hasher.hash("hello", 5000) << endl;
        testConstructor(hasher);
        testCopyConstructor(hasher);


        for(int i = 1; i <= NPartitions; ++i)
        {
                cout << "k = " << i << endl;
                measure(inputFileName, i*NSamples/NPartitions, hasher);
        }
        //hainedHashTable A(5000, hasher);

        //insertAll(inputFileName, A);
        //findAll(inputFileName, A);
        //removeAll(inputFileName, A);
        //call test functions

        //you may want to instantiate a ChainedHashTable
        //object to pass as a  reference to some of your
        //more advanced testing functions
}
                                                                    319,1         91%
```

Above is the measure and testHasher functions.

```
echo ------------compiling main.cpp to create executable program main--------------
------------compiling main.cpp to create executable program main--------------
g++ -ggdb -std=c++11 -Wpedantic -Wall -Wextra -Werror -Wzero-as-null-pointer-constant ChainedHashTable.cpp -o main
-bash-4.2$ valgrind main
==28751== Memcheck, a memory error detector
==28751== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==28751== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==28751== Command: main
==28751==
With text file random.txt
---------Using GeneralStringHasher----------
k = 1
Using 4539 words
min = 0; max = 8; average = 0.9078; std_dev = 1.14983;
insertAll: 0.106113
findAll: 0.073244
removeAll: 0.087571

k = 2
Using 9078 words
min = 0; max = 14; average = 1.8156; std_dev = 1.8629;
insertAll: 0.176181
findAll: 0.140347
removeAll: 0.176722

k = 3
Using 13617 words
min = 0; max = 19; average = 2.7234; std_dev = 2.56984;
insertAll: 0.253515
findAll: 0.22059
removeAll: 0.261219

k = 4
Using 18156 words
min = 0; max = 27; average = 3.6312; std_dev = 3.2445;
insertAll: 0.343324
findAll: 0.300578
removeAll: 0.362799

k = 5
Using 22696 words
min = 0; max = 34; average = 4.5392; std_dev = 3.91378;
insertAll: 0.427054
findAll: 0.38212
removeAll: 0.457722

k = 6
Using 27235 words
min = 0; max = 39; average = 5.447; std_dev = 4.59365;
insertAll: 0.511907
findAll: 0.467581
removeAll: 0.558999

k = 7
Using 31774 words
min = 0; max = 44; average = 6.3548; std_dev = 5.24154;
insertAll: 0.597694
findAll: 0.557876
removeAll: 0.667995

k = 8
Using 36313 words
min = 0; max = 53; average = 7.2626; std_dev = 5.89254;
insertAll: 0.680276
findAll: 0.648963
removeAll: 0.767981

k = 9
Using 40852 words
min = 0; max = 56; average = 8.1704; std_dev = 6.55849;
insertAll: 0.765029
findAll: 0.749785
removeAll: 0.878636

k = 10
Using 45392 words
min = 0; max = 61; average = 9.0784; std_dev = 7.23626;
insertAll: 0.845846
findAll: 0.847573
removeAll: 0.997764
```

```
---------Using SumHasher-----------
k = 1
Using 4539 words
min = 0; max = 23; average = 0.9078; std_dev = 2.54953;
insertAll: 0.092969
findAll: 0.086076
removeAll: 0.097212

k = 2
Using 9078 words
min = 0; max = 39; average = 1.8156; std_dev = 4.91117;
insertAll: 0.170364
findAll: 0.210105
removeAll: 0.228005

k = 3
Using 13617 words
min = 0; max = 54; average = 2.7234; std_dev = 7.26849;
insertAll: 0.256833
findAll: 0.359415
removeAll: 0.39585

k = 4
Using 18156 words
min = 0; max = 71; average = 3.6312; std_dev = 9.5922;
insertAll: 0.343478
findAll: 0.540516
removeAll: 0.594377

k = 5
Using 22696 words
min = 0; max = 92; average = 4.5392; std_dev = 11.9613;
insertAll: 0.428455
findAll: 0.760826
removeAll: 0.817121

k = 6
Using 27235 words
min = 0; max = 104; average = 5.447; std_dev = 14.2929;
insertAll: 0.519782
findAll: 1.00611
removeAll: 1.08589

k = 7
Using 31774 words
min = 0; max = 116; average = 6.3548; std_dev = 16.6917;
insertAll: 0.607343
findAll: 1.29433
removeAll: 1.38176

k = 8
Using 36313 words
min = 0; max = 128; average = 7.2626; std_dev = 19.0569;
insertAll: 0.688646
findAll: 1.6186
removeAll: 1.72509

k = 9
Using 40852 words
min = 0; max = 149; average = 8.1704; std_dev = 21.4232;
insertAll: 0.77498
findAll: 1.96468
removeAll: 2.09794

k = 10
Using 45392 words
min = 0; max = 163; average = 9.0784; std_dev = 23.7996;
insertAll: 0.853166
findAll: 2.36339
removeAll: 2.49624
```

```
---------Using ProdHasher------------
k = 1
Using 4539 words
min = 0; max = 15; average = 0.9078; std_dev = 2.26912;
insertAll: 0.086184
findAll: 0.075591
removeAll: 0.090729

k = 2
Using 9078 words
min = 0; max = 30; average = 1.8156; std_dev = 4.35146;
insertAll: 0.176588
findAll: 0.165378
removeAll: 0.193472

k = 3
Using 13617 words
min = 0; max = 40; average = 2.7234; std_dev = 6.42184;
insertAll: 0.255782
findAll: 0.265622
removeAll: 0.318693

k = 4
Using 18156 words
min = 0; max = 56; average = 3.6312; std_dev = 8.50393;
insertAll: 0.341326
findAll: 0.378495
removeAll: 0.450559

k = 5
Using 22696 words
min = 0; max = 67; average = 4.5392; std_dev = 10.5885;
insertAll: 0.425533
findAll: 0.510605
removeAll: 0.585232

k = 6
Using 27235 words
min = 0; max = 80; average = 5.447; std_dev = 12.6574;
insertAll: 0.516888
findAll: 0.644178
removeAll: 0.750025

k = 7
Using 31774 words
min = 0; max = 93; average = 6.3548; std_dev = 14.722;
insertAll: 0.604447
findAll: 0.803053
removeAll: 0.920324

k = 8
Using 36313 words
min = 0; max = 109; average = 7.2626; std_dev = 16.7802;
insertAll: 0.680389
findAll: 0.963775
removeAll: 1.09909

k = 9
Using 40852 words
min = 0; max = 121; average = 8.1704; std_dev = 18.847;
insertAll: 0.77599
findAll: 1.14505
removeAll: 1.30439

k = 10
Using 45392 words
min = 0; max = 130; average = 9.0784; std_dev = 20.9346;
insertAll: 0.860605
findAll: 1.34834
removeAll: 1.52152
```

```
With text file words.txt
---------Using GeneralStringHasher-----------
k = 1
Using 4539 words
min = 0; max = 20; average = 0.9078; std_dev = 1.9499;
insertAll: 0.084202
findAll: 0.071722
removeAll: 0.087492

k = 2
Using 9078 words
min = 0; max = 30; average = 1.8156; std_dev = 2.69748;
insertAll: 0.167726
findAll: 0.144785
removeAll: 0.183454

k = 3
Using 13617 words
min = 0; max = 41; average = 2.7234; std_dev = 3.70196;
insertAll: 0.252994
findAll: 0.224518
removeAll: 0.272022

k = 4
Using 18156 words
min = 0; max = 57; average = 3.6312; std_dev = 4.33026;
insertAll: 0.335063
findAll: 0.30355
removeAll: 0.375271

k = 5
Using 22696 words
min = 0; max = 59; average = 4.5392; std_dev = 4.96726;
insertAll: 0.419462
findAll: 0.38588
removeAll: 0.468009

k = 6
Using 27235 words
min = 0; max = 60; average = 5.447; std_dev = 5.39181;
insertAll: 0.50979
findAll: 0.46702
removeAll: 0.566283

k = 7
Using 31774 words
min = 0; max = 61; average = 6.3548; std_dev = 5.8528;
insertAll: 0.59513
findAll: 0.555971
removeAll: 0.675679

k = 8
Using 36313 words
min = 0; max = 61; average = 7.2626; std_dev = 6.41074;
insertAll: 0.670684
findAll: 0.655106
removeAll: 0.776151

k = 9
Using 40852 words
min = 0; max = 61; average = 8.1704; std_dev = 6.80013;
insertAll: 0.753502
findAll: 0.740893
removeAll: 0.889593

k = 10
Using 45392 words
min = 0; max = 61; average = 9.0784; std_dev = 7.23786;
insertAll: 0.837832
findAll: 0.833231
removeAll: 0.993276
```

```
---------Using SumHasher-----------
k = 1
Using 4539 words
min = 0; max = 20; average = 0.9078; std_dev = 2.59185;
insertAll: 0.085205
findAll: 0.087028
removeAll: 0.09954

k = 2
Using 9078 words
min = 0; max = 45; average = 1.8156; std_dev = 5.17448;
insertAll: 0.175319
findAll: 0.208516
removeAll: 0.234139

k = 3
Using 13617 words
min = 0; max = 61; average = 2.7234; std_dev = 7.46322;
insertAll: 0.253724
findAll: 0.361015
removeAll: 0.410745

k = 4
Using 18156 words
min = 0; max = 77; average = 3.6312; std_dev = 9.96474;
insertAll: 0.338817
findAll: 0.560859
removeAll: 0.607139

k = 5
Using 22696 words
min = 0; max = 95; average = 4.5392; std_dev = 12.2103;
insertAll: 0.4272
findAll: 0.774594
removeAll: 0.842763

k = 6
Using 27235 words
min = 0; max = 109; average = 5.447; std_dev = 14.4851;
insertAll: 0.512264
findAll: 1.01541
removeAll: 1.09447

k = 7
Using 31774 words
min = 0; max = 127; average = 6.3548; std_dev = 16.7138;
insertAll: 0.603655
findAll: 1.28612
removeAll: 1.37613

k = 8
Using 36313 words
min = 0; max = 135; average = 7.2626; std_dev = 19.0819;
insertAll: 0.687579
findAll: 1.59814
removeAll: 1.71301

k = 9
Using 40852 words
min = 0; max = 153; average = 8.1704; std_dev = 21.4708;
insertAll: 0.767588
findAll: 1.95452
removeAll: 2.08347

k = 10
Using 45392 words
min = 0; max = 163; average = 9.0784; std_dev = 23.8005;
insertAll: 0.858653
findAll: 2.32776
removeAll: 2.47362
```

```
---------Using ProdHasher------------
k = 1
Using 4539 words
min = 0; max = 17; average = 0.9078; std_dev = 2.22749;
insertAll: 0.085119
findAll: 0.075372
removeAll: 0.097576

k = 2
Using 9078 words
min = 0; max = 26; average = 1.8156; std_dev = 4.29525;
insertAll: 0.169394
findAll: 0.163905
removeAll: 0.194803

k = 3
Using 13617 words
min = 0; max = 42; average = 2.7234; std_dev = 6.46876;
insertAll: 0.255742
findAll: 0.26672
removeAll: 0.312973

k = 4
Using 18156 words
min = 0; max = 53; average = 3.6312; std_dev = 8.55395;
insertAll: 0.347817
findAll: 0.38034
removeAll: 0.446526

k = 5
Using 22696 words
min = 0; max = 63; average = 4.5392; std_dev = 10.7169;
insertAll: 0.427494
findAll: 0.514621
removeAll: 0.591916

k = 6
Using 27235 words
min = 0; max = 75; average = 5.447; std_dev = 12.5622;
insertAll: 0.513565
findAll: 0.648993
removeAll: 0.742148

k = 7
Using 31774 words
min = 0; max = 87; average = 6.3548; std_dev = 14.7713;
insertAll: 0.602667
findAll: 0.799117
removeAll: 0.924974

k = 8
Using 36313 words
min = 0; max = 99; average = 7.2626; std_dev = 16.846;
insertAll: 0.683326
findAll: 0.971874
removeAll: 1.10797

k = 9
Using 40852 words
min = 0; max = 122; average = 8.1704; std_dev = 18.8447;
insertAll: 0.768755
findAll: 1.14719
removeAll: 1.30977

k = 10
Using 45392 words
min = 0; max = 130; average = 9.0784; std_dev = 20.9344;
insertAll: 0.850919
findAll: 1.34389
removeAll: 1.52418
```

```
==28751==
==28751== HEAP SUMMARY:
==28751==     in use at exit: 0 bytes in 0 blocks
==28751==   total heap usage: 1,571,101 allocs, 1,571,101 frees, 78,480,824 bytes allocated
==28751==
==28751== All heap blocks were freed -- no leaks are possible
==28751==
==28751== For counts of detected and suppressed errors, rerun with: -v
==28751== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
-bash-4.2$
```

Above is the code ran with valgrind and all of its output using 3 different hash functions.

```
With text file random.txt
---------Using GeneralStringHasher-----------
k = 1
Using 4539 words
min = 0; max = 8; average = 0.9078; std_dev = 1.14983;
insertAll: 0.00201
findAll: 0.001817
removeAll: 0.001856

k = 2
Using 9078 words
min = 0; max = 14; average = 1.8156; std_dev = 1.8629;
insertAll: 0.004249
findAll: 0.00385
removeAll: 0.00409

k = 3
Using 13617 words
min = 0; max = 19; average = 2.7234; std_dev = 2.56984;
insertAll: 0.006531
findAll: 0.006104
removeAll: 0.006211

k = 4
Using 18156 words
min = 0; max = 27; average = 3.6312; std_dev = 3.2445;
insertAll: 0.007829
findAll: 0.008642
removeAll: 0.008302

k = 5
Using 22696 words
min = 0; max = 34; average = 4.5392; std_dev = 3.91378;
insertAll: 0.010716
findAll: 0.01116
removeAll: 0.01157

k = 6
Using 27235 words
min = 0; max = 39; average = 5.447; std_dev = 4.59365;
insertAll: 0.011942
findAll: 0.014178
removeAll: 0.014427

k = 7
Using 31774 words
min = 0; max = 44; average = 6.3548; std_dev = 5.24154;
insertAll: 0.0142
findAll: 0.016924
removeAll: 0.017481

k = 8
Using 36313 words
min = 0; max = 53; average = 7.2626; std_dev = 5.89254;
insertAll: 0.012784
findAll: 0.019188
removeAll: 0.020733

k = 9
Using 40852 words
min = 0; max = 56; average = 8.1704; std_dev = 6.55849;
insertAll: 0.020006
findAll: 0.022533
removeAll: 0.022015

k = 10
Using 45392 words
min = 0; max = 61; average = 9.0784; std_dev = 7.23626;
insertAll: 0.022029
findAll: 0.027001
removeAll: 0.027869
```

```
---------Using SumHasher-----------
k = 1
Using 4539 words
min = 0; max = 23; average = 0.9078; std_dev = 2.54953;
insertAll: 0.001134
findAll: 0.002369
removeAll: 0.002434

k = 2
Using 9078 words
min = 0; max = 39; average = 1.8156; std_dev = 4.91117;
insertAll: 0.002377
findAll: 0.005951
removeAll: 0.006044

k = 3
Using 13617 words
min = 0; max = 54; average = 2.7234; std_dev = 7.26849;
insertAll: 0.006505
findAll: 0.010666
removeAll: 0.010816

k = 4
Using 18156 words
min = 0; max = 71; average = 3.6312; std_dev = 9.5922;
insertAll: 0.009067
findAll: 0.016549
removeAll: 0.016776

k = 5
Using 22696 words
min = 0; max = 92; average = 4.5392; std_dev = 11.9613;
insertAll: 0.009921
findAll: 0.023865
removeAll: 0.024426

k = 6
Using 27235 words
min = 0; max = 104; average = 5.447; std_dev = 14.2929;
insertAll: 0.011305
findAll: 0.032875
removeAll: 0.033632

k = 7
Using 31774 words
min = 0; max = 116; average = 6.3548; std_dev = 16.6917;
insertAll: 0.014246
findAll: 0.042091
removeAll: 0.043341

k = 8
Using 36313 words
min = 0; max = 128; average = 7.2626; std_dev = 19.0569;
insertAll: 0.014521
findAll: 0.054997
removeAll: 0.05611

k = 9
Using 40852 words
min = 0; max = 149; average = 8.1704; std_dev = 21.4232;
insertAll: 0.019134
findAll: 0.067481
removeAll: 0.069165

k = 10
Using 45392 words
min = 0; max = 163; average = 9.0784; std_dev = 23.7996;
insertAll: 0.022375
findAll: 0.081852
removeAll: 0.083721
```

```
---------Using ProdHasher------------
k = 1
Using 4539 words
min = 0; max = 15; average = 0.9078; std_dev = 2.26912;
insertAll: 0.000296
findAll: 0.002236
removeAll: 0.002293

k = 2
Using 9078 words
min = 0; max = 30; average = 1.8156; std_dev = 4.35146;
insertAll: 0.004563
findAll: 0.005108
removeAll: 0.00526

k = 3
Using 13617 words
min = 0; max = 40; average = 2.7234; std_dev = 6.42184;
insertAll: 0.00675
findAll: 0.008322
removeAll: 0.008636

k = 4
Using 18156 words
min = 0; max = 56; average = 3.6312; std_dev = 8.50393;
insertAll: 0.008573
findAll: 0.012457
removeAll: 0.013026

k = 5
Using 22696 words
min = 0; max = 67; average = 4.5392; std_dev = 10.5885;
insertAll: 0.010616
findAll: 0.01711
removeAll: 0.017962

k = 6
Using 27235 words
min = 0; max = 80; average = 5.447; std_dev = 12.6574;
insertAll: 0.012513
findAll: 0.022413
removeAll: 0.024021

k = 7
Using 31774 words
min = 0; max = 93; average = 6.3548; std_dev = 14.722;
insertAll: 0.014095
findAll: 0.028405
removeAll: 0.029962

k = 8
Using 36313 words
min = 0; max = 109; average = 7.2626; std_dev = 16.7802;
insertAll: 0.017424
findAll: 0.035382
removeAll: 0.037048

k = 9
Using 40852 words
min = 0; max = 121; average = 8.1704; std_dev = 18.847;
insertAll: 0.017941
findAll: 0.042419
removeAll: 0.045232

k = 10
Using 45392 words
min = 0; max = 130; average = 9.0784; std_dev = 20.9346;
insertAll: 0.022112
findAll: 0.050354
removeAll: 0.054151
```

```
With text file words.txt
---------Using GeneralStringHasher-----------
k = 1
Using 4539 words
min = 0; max = 20; average = 0.9078; std_dev = 1.9499;
insertAll: 0.002111
findAll: 0.001951
removeAll: 0.002003

k = 2
Using 9078 words
min = 0; max = 30; average = 1.8156; std_dev = 2.69748;
insertAll: 0.004215
findAll: 0.003969
removeAll: 0.004077

k = 3
Using 13617 words
min = 0; max = 41; average = 2.7234; std_dev = 3.70196;
insertAll: 0.005452
findAll: 0.006258
removeAll: 0.006592

k = 4
Using 18156 words
min = 0; max = 57; average = 3.6312; std_dev = 4.33026;
insertAll: 0.008509
findAll: 0.008619
removeAll: 0.009012

k = 5
Using 22696 words
min = 0; max = 59; average = 4.5392; std_dev = 4.96726;
insertAll: 0.008697
findAll: 0.009976
removeAll: 0.011251

k = 6
Using 27235 words
min = 0; max = 60; average = 5.447; std_dev = 5.39181;
insertAll: 0.012895
findAll: 0.014211
removeAll: 0.014156

k = 7
Using 31774 words
min = 0; max = 61; average = 6.3548; std_dev = 5.8528;
insertAll: 0.013973
findAll: 0.016372
removeAll: 0.01693

k = 8
Using 36313 words
min = 0; max = 61; average = 7.2626; std_dev = 6.41074;
insertAll: 0.014993
findAll: 0.019196
removeAll: 0.018815

k = 9
Using 40852 words
min = 0; max = 61; average = 8.1704; std_dev = 6.80013;
insertAll: 0.01772
findAll: 0.022335
removeAll: 0.022783

k = 10
Using 45392 words
min = 0; max = 61; average = 9.0784; std_dev = 7.23786;
insertAll: 0.019784
findAll: 0.025362
removeAll: 0.025863
```

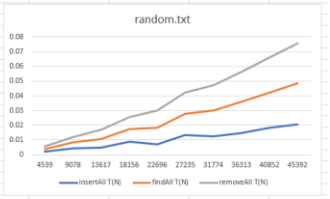
```
---------Using SumHasher----------
k = 1
Using 4539 words
min = 0; max = 20; average = 0.9078; std_dev = 2.59185;
insertAll: 0.002153
findAll: 0.002884
removeAll: 0.002535

k = 2
Using 9078 words
min = 0; max = 45; average = 1.8156; std_dev = 5.17448;
insertAll: 0.003279
findAll: 0.006428
removeAll: 0.006547

k = 3
Using 13617 words
min = 0; max = 61; average = 2.7234; std_dev = 7.46322;
insertAll: 0.005484
findAll: 0.011353
removeAll: 0.011509

k = 4
Using 18156 words
min = 0; max = 77; average = 3.6312; std_dev = 9.96474;
insertAll: 0.007676
findAll: 0.017662
removeAll: 0.018146

k = 5
Using 22696 words
min = 0; max = 95; average = 4.5392; std_dev = 12.2103;
insertAll: 0.009877
findAll: 0.024738
removeAll: 0.025776

k = 6
Using 27235 words
min = 0; max = 109; average = 5.447; std_dev = 14.4851;
insertAll: 0.012077
findAll: 0.032192
removeAll: 0.033952

k = 7
Using 31774 words
min = 0; max = 127; average = 6.3548; std_dev = 16.7138;
insertAll: 0.011971
findAll: 0.042751
removeAll: 0.042397

k = 8
Using 36313 words
min = 0; max = 135; average = 7.2626; std_dev = 19.0819;
insertAll: 0.015366
findAll: 0.053832
removeAll: 0.054638

k = 9
Using 40852 words
min = 0; max = 153; average = 8.1704; std_dev = 21.4708;
insertAll: 0.017408
findAll: 0.065875
removeAll: 0.067228

k = 10
Using 45392 words
min = 0; max = 163; average = 9.0784; std_dev = 23.8005;
insertAll: 0.021858
findAll: 0.079003
removeAll: 0.080032
```

```
---------Using ProdHasher------------
k = 1
Using 4539 words
min = 0; max = 17; average = 0.9078; std_dev = 2.22749;
insertAll: 0.002228
findAll: 0.00257
removeAll: 0.002251

k = 2
Using 9078 words
min = 0; max = 26; average = 1.8156; std_dev = 4.29525;
insertAll: 0.003513
findAll: 0.004953
removeAll: 0.005141

k = 3
Using 13617 words
min = 0; max = 42; average = 2.7234; std_dev = 6.46876;
insertAll: 0.005377
findAll: 0.008381
removeAll: 0.008751

k = 4
Using 18156 words
min = 0; max = 53; average = 3.6312; std_dev = 8.55395;
insertAll: 0.008037
findAll: 0.011337
removeAll: 0.012997

k = 5
Using 22696 words
min = 0; max = 63; average = 4.5392; std_dev = 10.7169;
insertAll: 0.009565
findAll: 0.017257
removeAll: 0.018111

k = 6
Using 27235 words
min = 0; max = 75; average = 5.447; std_dev = 12.5622;
insertAll: 0.014201
findAll: 0.022214
removeAll: 0.022429

k = 7
Using 31774 words
min = 0; max = 87; average = 6.3548; std_dev = 14.7713;
insertAll: 0.014877
findAll: 0.027217
removeAll: 0.029827

k = 8
Using 36313 words
min = 0; max = 99; average = 7.2626; std_dev = 16.846;
insertAll: 0.017967
findAll: 0.033841
removeAll: 0.035893

k = 9
Using 40852 words
min = 0; max = 122; average = 8.1704; std_dev = 18.8447;
insertAll: 0.017784
findAll: 0.042361
removeAll: 0.044496

k = 10
Using 45392 words
min = 0; max = 130; average = 9.0784; std_dev = 20.9344;
insertAll: 0.021024
findAll: 0.049776
removeAll: 0.053262
```

Above is the code ran with the random.txt and words.txt text files using the 3 different hash functions, no valgrind.

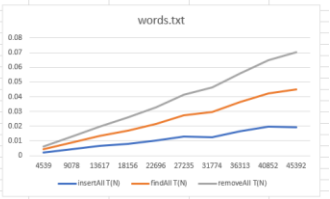
random.txt

	N (number of inputs)									
	4539	9078	13617	18156	22696	27235	31774	36313	40852	45392
insertAll T(N)	0.002088	0.004323	0.004675	0.008701	0.006974	0.013246	0.012589	0.014653	0.018231	0.020524
findAll T(N)	0.001851	0.00388	0.006145	0.008892	0.011564	0.014497	0.017469	0.021147	0.024192	0.028257
removeAll T(N)	0.001823	0.003839	0.006095	0.00782	0.011445	0.014636	0.017194	0.020368	0.023649	0.027125
min	0	0	0	0	0	0	0	0	0	0
max	8	14	19	27	34	39	44	53	56	61
avg	0.9078	1.8156	2.7234	3.6312	4.5392	5.447	6.3548	7.2626	8.1704	9.0784
std_dev	1.14983	1.8629	2.56984	3.2445	3.91378	4.59365	5.24154	5.89254	6.55849	7.23626



words.txt

	N (number of inputs)									
	4539	9078	13617	18156	22696	27235	31774	36313	40852	45392
insertAll T(N)	0.002181	0.004438	0.006591	0.007861	0.010001	0.013046	0.012233	0.01674	0.01967	0.019258
findAll T(N)	0.002071	0.004214	0.006608	0.00903	0.011583	0.014181	0.017594	0.019805	0.022757	0.02586
removeAll T(N)	0.00204	0.004144	0.006499	0.009251	0.011414	0.013946	0.016648	0.019492	0.022373	0.025375
min	0	0	0	0	0	0	0	0	0	0
max	20	30	41	57	59	60	61	61	61	61
avg	0.9078	1.8156	2.7234	3.6312	4.5392	5.447	6.3548	7.2626	8.1704	9.0784
std_dev	1.9499	2.69748	3.70196	4.33026	4.96726	5.39181	5.8528	6.41074	6.80013	7.23786



Here is my data of each time the functions took using different N of inputs.

End of Report