INTRODUCTION

This week we cover the following subjects

9.1 HASHING

- The main reason we want to sort items in order is to speed up searching for them. While binary search is considerably faster than linear there are times when this is still too slow.

- An alternative to sorting is to use a *hash function* to store an item in a set position within a *hash table*. The hash function can then be used to retrieve the item from the table.

9.1.1 Hash Functions

- A hash function is any function that transforms any given value into a pseudo random number with a set hash range of 0 to *h*.

- Note, it is possible to have different values give the same hash value. This has implications for hash tables, which will be discussed below.

- In order to be useful a hash function needs a number of important properties.
  - The hash function should be quick to calculate.
  - The hash function should generate values that evenly spread across the given hash range.

Example:

- Student numbers are 8 digit numbers. We want a hashing function that will produce a hash value in the range 0 - 29999 from student numbers.

- In this example, our hash function takes the first four digits and multiplies it by the last four digits and then finds modulus 30000 of the result.

- If the student number = 01074838
  Then the hash value = (0107 × 4838) % 30000
  = 517666 % 30000
  = 7666

- Note, the student number 01824163 will also produce the hash value 7666. Most other student numbers will produce a different hash value.

- The following code implements the hash function

```
int hashfunc(int studnum)
{
   int result = studnum / 10000;

   result *= (studnum % 10000);
   return (result % 30000);
}
```

9.1.2 Hash Tables

- A Hash Table is data structure, usually an array, to store items in. The location within the table that an item will be placed is determined by a generated hash value.

- Consider the example of student numbers. A student number is an 8 digit number. We could declare an array of 100,000,000 in size to store all student

records. We simply place the student record in the position determined by their student number. E.g. `studentArray[1074838] = studentRecord;`

- Two problems exist though.
  - This size array would require memory beyond the capacity of most machines
  - There may only be 20000 student records. Therefore, 99.98% of the array would be unused.

- Instead, we could use an array of 30,000 in size and locate student records in the position determined by hashing their student numbers. In the case of student 01074838 we place their record in position 7666. E.g. `hashTable[7666] = studentRecord;`

- This way we use up to 67% of the array. The declared array is our hash table.

- What happens if two students have the same hash value, as in student 01824163? When this happens we have what is called a *collision* or a *clash* and it needs to be resolved.

9.1.3 Collision Resolution

- Two basic strategies exist for collision resolution.
  - Open Addressing
  - Chaining

- In Open Addressing, we seek unused locations in the table to store the item. We seek the locations in an

ordered fashion. For instance, every 3rd location after the initial hash location, till an empty slot is found.

- In Chaining, each hash location represents the start of a linked list containing entries. While a bit more complicated to program this mechanism provides a number of advantages.
  - Concerns about the Hash Table becoming full are eliminated.
  - A smaller Hash Table is required so it may be possible to reduce space taken up by the table.

9.1.4 Performance

- If there have been no collisions while placing items in the hash table then the speed of finding anything is O(1). In other words, regardless of the number of items in the table it only takes 1 operation to find them.

- Performance will degrade as the number of collisions mount up. Generally this only starts becoming serious when the table is > 85% full. Bad hashing functions can compound this problem.

9.2 linhash.h

```
sally% cat dataobject.h

#ifndef DATAOBJECT_H_
#define DATAOBJECT_H_

#include <algorithm>
```

```
/*********************************************\
   class for holding data to be stored in
   container objects
\*********************************************/

struct dataObject
{
   int keyval;

   /*******************************************\
      place any other data declartions the data
      object may need here
   \*******************************************/

   /*******************************************\
      constructors
   \*******************************************/

   // constructor
   dataObject() : keyval(-1) {}
   dataObject(int val) : keyval(val) {}

   // copy constructor
   dataObject(const dataObject &other) :
      keyval(other.keyval) {
      // copy any data in other to this
   }

   /*******************************************\
      misc functions
   \*******************************************/

   void swapObjects(dataObject &other) {
      std::swap(keyval, other.keyval);
      // swap any other data in dataObject
   }

   bool isKeyval(int val) const {
      return (val == keyval);
   }

   int getKeyval() const {
      return keyval;
   }
```

```
int getHashval(int range) const {
   return (keyval % range);
}

bool empty() const {
   return (keyval == -1);
}

/*******************************************\
   overloaded operators
\*******************************************/

// overloaded assignment operator
dataObject& operator = (const dataObject
                        &other) {
   dataObject temp(other);
   swapObjects(temp);
   return *this;
}

// overloaded relational operators
bool operator ==
   (const dataObject &other) const {
   return (keyval == other.keyval);
}

bool operator !=
   (const dataObject &other) const {
   return (keyval != other.keyval);
}

bool
  operator > (const dataObject &other) const {
   return (keyval > other.keyval);
}

bool
  operator < (const dataObject &other) const {
   return (keyval < other.keyval);
}
```

```
   bool
    operator >= (const dataObject &other) const {
      return (keyval >= other.keyval);
   }

   bool
    operator <= (const dataObject &other) const {
      return (keyval <= other.keyval);
   }
};

#endif

// ###############################################

sally % cat linhash.h

#ifndef LINHASH_H_
#define LINHASH_H_

#include <stdexcept>

/*********************************************\
   template class for a linear hash table
\*********************************************/

template <typename dataType> class linHashtable
{
   private:
      dataType *theTable;
      int range, numItems;

   public:
      /*****************************************\
         constructors
      \*****************************************/

      // constructor

      linHashtable(int r) :
           range(r), numItems(0) {
         theTable = new dataType[r];
      }
```

```
/*****************************************\
   status functions
\*****************************************/

bool empty() const {
   return (numItems == 0);
}

bool full() const {
   // We ensure there is always 1 empty
   // space in table
   return (numItems == range-1);
}

int getRange() const {
   return range;
}

int size() const {
   return numItems;
}

/*****************************************\
   insertion erasure and find functions
\*****************************************/

void insert(const dataType& newData)
{
   if (full()) {
      throw std::out_of_range(
              "hash table is full");
   }

   int hashval = newData.getHashval(range);

   // starting with hashval,
   // find place for newData in table
   while (!theTable[hashval].empty())
      hashval = (hashval + 1) % range;

   theTable[hashval] = newData;
   numItems++;
}
```

```
void erase(const dataType& delData)
{
   int hashval = delData.getHashval(range);
   int next, start = hashval;

   // starting with hashval,
   // find place of deldata in table
   while (theTable[hashval] != delData) {
      if (theTable[hashval].empty())
         return;
      hashval = (hashval + 1) % range;
   }

   // find any other data in hashtable
   // with same hash value and move.
   start = hashval;
   next = (start + 1) % range;
   hashval = delData.getHashval(range);

   while (!theTable[next].empty()) {
      if (theTable[next].getHashval(range)
            == hashval) {
         theTable[start] = theTable[next];
         start = next;
      }
      next = (next + 1) % range;
   }

   // make final space empty
   dataType emptyData;
   theTable[start] = emptyData;
   numItems--;
}

bool find(const dataType &findData,
                dataType &foundData) const
{
   int hashval =
      findData.getHashval(range);

   // starting with position at hashval,
   // find findData in table
```

```
         while (theTable[hashval] != findData) {
            if (theTable[hashval].empty())
               return false;
            hashval = (hashval + 1) % range;
         }
         foundData = theTable[hashval];
         return true;
      }
};

#endif

// ###############################################

sally % cat testmain.cpp

/*********************************************\
   Test program for demonstrating container types
\*********************************************/
#include <sys/time.h>
#include <time.h>
#include <stdlib.h>

#include <iostream>
#include <algorithm>

#include "dataobject.h"
#include "linhash.h"

using namespace std;

double difUtime(struct timeval *first, struct
timeval *second);
int randNum();

double difUtime(struct timeval *first, struct
timeval *second)
{
   // return the difference in seconds, including
milli seconds

   double difsec =
      second->tv_sec - first->tv_sec;
```

```
   double difusec =
      second->tv_usec - first->tv_usec;

   return (difsec + (difusec) / 1000000.0);
}

int randNum()
{
   // rand() produces pseudo-random number in
   // range 0 to RAND_MAX
   // for most compilers RAND_MAX is 2^15-1
   // randNum() produces pseudo-random
   // number in range 0 to 2^31-1

   static bool initialised = false;

   if (!initialised) {
      srand(time(NULL));
      initialised  = true;
   }

   return ((rand() << 16) + rand());
}

int main()
{
   const int MAXDATA = 1000000;
   dataObject *doPtr, data;
   int i, keyvals[MAXDATA];
   bool found;

   // create hashtable, making sure it is only
   // 66% full at most
   linHashtable<dataObject>
      testContainer(MAXDATA*3/2);

   // data for calculating timing
   struct timeval first, second;
   double usecs;
```

```
try {
   /*****************************************\
      Initialise things to demonstrate the
      container
      - fill keyvals with number 2 digits
        greater than container size
      - scramble keyvals
   \*****************************************/

   for (i=0; i<MAXDATA; i++) keyvals[i] =
      (i * 100) + (randNum() % 100);
   for (i=0; i<MAXDATA; i++)
      swap(keyvals[i],
           keyvals[randNum() % MAXDATA]);

   /*****************************************\
      test inserting MAXDATA data pieces into
      the container with keyval 0 to MAXDATA-1
      in random order
   \*****************************************/

   gettimeofday(&first, NULL);
   for (i=0; i<MAXDATA; i++) {
      doPtr = new dataObject(keyvals[i]);
      testContainer.insert(*doPtr);
   }

   gettimeofday(&second, NULL);
   usecs = difUtime(&first, &second);
   cout << "\nContainer size = " <<
       testContainer.getRange() << "\n";
   cout << "MAXDATA = " << MAXDATA <<
       " items in keyvals in random order\n";
   cout << "time taken to insert data into
     container = " << usecs << " seconds\n\n";

   /*****************************************\
      test finding data in the container
   \*****************************************/

   gettimeofday(&first, NULL);
   found =
    testContainer.find(
        dataObject(keyvals[0]), data);
```

```
gettimeofday(&second, NULL);
usecs = difUtime(&first, &second);
if (!found) cout << "Not found\n";
else cout << "Found\n";
cout << "time taken to find keyvals[0] in
   container = " << usecs << " seconds\n";

gettimeofday(&first, NULL);
found = testContainer.find(
   dataObject(keyvals[MAXDATA/2]), data);
gettimeofday(&second, NULL);
usecs = difUtime(&first, &second);
if (!found) cout << "Not found\n";
else cout << "Found\n";
cout << "time taken to find
  keyvals[MAXDATA/2] in container = "
  << usecs << " seconds\n";

gettimeofday(&first, NULL);
found = testContainer.find(
   dataObject(keyvals[MAXDATA-1]), data);
gettimeofday(&second, NULL);
usecs = difUtime(&first, &second);
if (!found) cout << "Not found\n";
else cout << "Found\n";
cout << "time taken to find
   keyvals[MAXDATA-1] in container = "
   << usecs << " seconds\n";

gettimeofday(&first, NULL);
found = testContainer.find(
   dataObject(MAXDATA*100), data);
gettimeofday(&second, NULL);
usecs = difUtime(&first, &second);
if (!found) cout << "Not found\n";
else cout << "Found\n";
cout << "time taken to find a keyval
   doesn't exist in container = " << usecs
   << " seconds\n";

/*****************************************\
   test removing data from the container
\*****************************************/
```

```
      gettimeofday(&first, NULL);
      testContainer.erase(
         dataObject(keyvals[0]));
      gettimeofday(&second, NULL);
      usecs = difUtime(&first, &second);
      cout << "\ntime taken to erase keyvals[0]
         from container = " << usecs
         << " seconds\n";

      gettimeofday(&first, NULL);
      testContainer.erase(
         dataObject(keyvals[MAXDATA/2]));
      gettimeofday(&second, NULL);
      usecs = difUtime(&first, &second);
      cout << "time taken to erase
         keyvals[MAXDATA/2] from container = "
         << usecs << " seconds\n";

      gettimeofday(&first, NULL);
      testContainer.erase(
         dataObject(keyvals[MAXDATA-1]));
      gettimeofday(&second, NULL);
      usecs = difUtime(&first, &second);
      cout << "time taken to erase
         keyvals[MAXDATA-1] from container = "
         << usecs << " seconds\n";

   } catch (out_of_range &ex) {
      cout << "\nERROR - Out of Range Exception
         thrown\n" << ex.what() << "\n";
      exit(1);
   } catch (invalid_argument &ex) {
      cout << "\nERROR - Invalid Argument
         Exception thrown\n" << ex.what()
         << "\n";
      exit(1);
   } catch(...) {
      cout << "\nERROR - undefined Exception
         thrown\n";
      exit(1);
   }

   return 0;
}
```

```
// ##############################################

sally % cat makefile

CC = g++
prog: testmain.o
    $(CC) testmain.o -Wall -o testmain
testmain.o: testmain.cpp linhash.h dataobject.h
    $(CC) -Wall -c testmain.cpp

sally% testmain

Container size = 1500000
MAXDATA = 1000000 items in keyvals in random
order
time taken to insert data into container =
3.12856 seconds

Found
time taken to find keyvals[0] in container = 6e-
06 seconds
Found
time taken to find keyvals[MAXDATA/2] in
container = 6e-06 seconds
Found
time taken to find keyvals[MAXDATA-1] in
container = 6e-06 seconds
Not found
time taken to find a keyval doesn't exist in
container = 9e-06 seconds

time taken to erase keyvals[0] from container =
9e-06 seconds
time taken to erase keyvals[MAXDATA/2] from
container = 5e-06 seconds
time taken to erase keyvals[MAXDATA-1] from
container = 5e-06 seconds
```

9.3 chainhash.h

```
sally% cat chainhash.h
```

```
ifndef CHAINHASH_H_
#define CHAINHASH_H_

#include <stdexcept>
#include <list>

/*********************************************\
   template class for a hash table using chaining
\*********************************************/

template <typename dataType> class chainHashtable
{
   private:
      std::list<dataType> *theTable;
      int range, numItems;

   public:
      /*****************************************\
         constructors
      \*****************************************/

      // constructor

      chainHashtable(int r) :
            range(r), numItems(0) {
         theTable = new std::list<dataType>[r];
      }

      /*****************************************\
         status functions
      \*****************************************/

      bool empty() const {
         return (numItems == 0);
      }

      int getRange() const {
         return range;
      }

      int size() const {
         return numItems;
      }
```

```
/*****************************************\
   insertion erasure and find functions
\*****************************************/

void insert(const dataType& newData)
{
   int hashval =
      newData.getHashval(range);

   // starting with hashval,
   // find place for newData in table
   theTable[hashval].push_front(newData);
   numItems++;
}

void erase(const dataType& delData)
{
   dataType temp;

   if (find(delData, temp)) {
      int hashval =
         delData.getHashval(range);
      theTable[hashval].remove(delData);
      numItems--;
   }
}

bool find(
   const dataType &findData,
   dataType &foundData) const
{
   int hashval =
      findData.getHashval(range);
   typename std::list<dataType>::iterator
      listIter = theTable[hashval].begin();

   for( ; listIter !=
          theTable[hashval].end();
          listIter++) {
      if (*listIter == findData) {
         foundData = *listIter;
         return true;
      }
   }
```

```
         return false;
      }
};

#endif

// ##############################################

sally% cat testmain.cpp

/*********************************************\
   Test program for demonstrating container types
\*********************************************/
#include <sys/time.h>
#include <time.h>
#include <stdlib.h>

#include <iostream>
#include <algorithm>

#include "dataobject.h"
#include "chainhash.h"

using namespace std;

double difUtime(struct timeval *first, struct
timeval *second);
int randNum();

double difUtime(struct timeval *first, struct
timeval *second)
{
   // return the difference in seconds,
   // including milli seconds

   double difsec =
      second->tv_sec - first->tv_sec;
   double difusec =
      second->tv_usec - first->tv_usec;

   return (difsec + (difusec) / 1000000.0);
}
```

```
int randNum()
{
   // rand() produces pseudo-random number in
   // range 0 to RAND_MAX
   // for most compilers RAND_MAX is 2^15-1
   // randNum() produces pseudo-random number
   // in range 0 to 2^31-1

   static bool initialised = false;

   if (!initialised) {
      srand(time(NULL));
      initialised  = true;
   }

   return ((rand() << 16) + rand());
}

int main()
{
   const int MAXDATA = 1000000;
   dataObject *doPtr, data;
   int i, keyvals[MAXDATA];
   bool found;

   // create hashtable
   chainHashtable<dataObject>
      testContainer(MAXDATA/2);

   // data for calculating timing
   struct timeval first, second;
   double usecs;

   try {
      /*****************************************\
         Initialise things to demonstrate the
         container
         - fill keyvals with number 2 digits
           greater than container size
         - scramble keyvals
      \*****************************************/
```

```
for (i=0; i<MAXDATA; i++) keyvals[i] =
        (i * 100) + (randNum() % 100);
for (i=0; i<MAXDATA; i++)
    swap(keyvals[i],
         keyvals[randNum() % MAXDATA]);

/*****************************************\
   test inserting MAXDATA data pieces into
   the container with keyval 0 to MAXDATA-1
   in random order
\*****************************************/

gettimeofday(&first, NULL);
for (i=0; i<MAXDATA; i++) {
   doPtr = new dataObject(keyvals[i]);
   testContainer.insert(*doPtr);
}

gettimeofday(&second, NULL);
usecs = difUtime(&first, &second);
cout << "\nContainer size = " <<
    testContainer.getRange() << "\n";
cout << "MAXDATA = " << MAXDATA <<
     " items in keyvals in random order\n";
cout << "time taken to insert data into
  container = " << usecs << " seconds\n\n";

/*****************************************\
   test finding data in the container
\*****************************************/

gettimeofday(&first, NULL);
found = testContainer.
        find(dataObject(keyvals[0]), data);
gettimeofday(&second, NULL);
usecs = difUtime(&first, &second);
if (!found) cout << "Not found\n";
else cout << "Found\n";
cout << "time taken to find keyvals[0] in
    container = " << usecs << " seconds\n";

gettimeofday(&first, NULL);
found = testContainer.
       find(dataObject(keyvals[MAXDATA/2]),
```

```
            data);
gettimeofday(&second, NULL);
usecs = difUtime(&first, &second);
if (!found) cout << "Not found\n";
else cout << "Found\n";
cout << "time taken to find
   keyvals[MAXDATA/2] in container = "
   << usecs << " seconds\n";

gettimeofday(&first, NULL);
found = testContainer.
       find(dataObject(keyvals[MAXDATA-1]),
            data);
gettimeofday(&second, NULL);
usecs = difUtime(&first, &second);
if (!found) cout << "Not found\n";
else cout << "Found\n";
cout << "time taken to find
    keyvals[MAXDATA-1] in container = "
    << usecs << " seconds\n";

gettimeofday(&first, NULL);
found = testContainer.
       find(dataObject(MAXDATA*100), data);
gettimeofday(&second, NULL);
usecs = difUtime(&first, &second);
if (!found) cout << "Not found\n";
else cout << "Found\n";
cout << "time taken to find a keyval
    doesn't exist in container = "
    << usecs << " seconds\n";

/*****************************************\
   test removing data from the container
\*****************************************/

gettimeofday(&first, NULL);
testContainer.
      erase(dataObject(keyvals[0]));
gettimeofday(&second, NULL);
usecs = difUtime(&first, &second);
cout << "\ntime taken to erase keyvals[0]
    from container = "
    << usecs << " seconds\n";
```

```
        gettimeofday(&first, NULL);
        testContainer.
             erase(dataObject(keyvals[MAXDATA/2]));
        gettimeofday(&second, NULL);
        usecs = difUtime(&first, &second);
        cout << "time taken to erase
             keyvals[MAXDATA/2] from container = "
              << usecs << " seconds\n";

        gettimeofday(&first, NULL);
        testContainer.
             erase(dataObject(keyvals[MAXDATA-1]));
        gettimeofday(&second, NULL);
        usecs = difUtime(&first, &second);
        cout << "time taken to erase
             keyvals[MAXDATA-1] from container = "
             << usecs << " seconds\n";

    } catch (out_of_range &ex) {
        cout << "\nERROR - Out of Range Exception
              thrown\n" << ex.what() << "\n";
        exit(1);
    } catch (invalid_argument &ex) {
        cout << "\nERROR - Invalid Argument
          Exception thrown\n" << ex.what() << "\n";
        exit(1);
    } catch(...) {
        cout << "\nERROR - undefined Exception
           thrown\n";
        exit(1);
    }

    return 0;
}

// ###############################################

sally% cat makefile

CC = g++
prog: testmain.o
     $(CC) testmain.o -Wall -o testmain
testmain.o: testmain.cpp chainhash.h dataobject.h
```

```
    $(CC) -Wall -c testmain.cpp

sally% ./testmain

Container size = 500000
MAXDATA = 1000000 items in keyvals in random
order
time taken to insert data into container =
6.90041 seconds

Found
time taken to find keyvals[0] in
container = 9e-06 seconds
Found
time taken to find keyvals[MAXDATA/2] in
container = 7e-06 seconds
Found
time taken to find keyvals[MAXDATA-1] in
container = 4e-06 seconds
Not found
time taken to find a keyval doesn't exist in
container = 9e-06 seconds

time taken to erase keyvals[0] from container =
6.4e-05 seconds
time taken to erase keyvals[MAXDATA/2] from
container = 1.3e-05 seconds
time taken to erase keyvals[MAXDATA-1] from
container = 1.2e-05 seconds
```

9.4 DATA STRUCTURE TIME COMPARISONS

VECTOR

```
MAXDATA = 1000000 items in keyvals in random
order
time taken to insert data into container = 3.0579
seconds

Found
time taken to find keyvals[0] in container = 5e-
06 seconds
```

```
Found
time taken to find keyvals[MAXDATA/2] in
container = 0.095845 seconds
Found
time taken to find keyvals[MAXDATA-1] in
container = 0.186671 seconds
Not found
time taken to find a keyval doesn't exist in
container = 0.192016 seconds

time taken to erase first item from container =
0.447668 seconds
time taken to erase middle item from container =
0.242449 seconds
time taken to erase last item from container =
3e-06 seconds
```

DLIST

```
MAXDATA = 1000000 items in keyvals in random
order
time taken to insert data into container =
2.95134 seconds

Found
time taken to find keyvals[0] in container = 8e-
06 seconds
Found
time taken to find keyvals[MAXDATA/2] in
container = 0.631783 seconds
Found
time taken to find keyvals[MAXDATA-1] in
container = 1.27088 seconds
Not found
time taken to find a keyval doesn't exist in
container = 1.27736 seconds

time taken to erase keyvals[0] from container =
8.1e-05 seconds
time taken to erase keyvals[MAXDATA/2] from
container = 8e-06 seconds
time taken to erase keyvals[MAXDATA-1] from
container = 4e-06 seconds
```

ORDERED VECTOR

```
MAXDATA = 10000 items in keyvals in random order
time taken to insert data into container =
10.4604 seconds

Found
time taken to find keyvals[0] in container =
1.6e-05 seconds
Found
time taken to find keyvals[MAXDATA/2] in
container = 1.5e-05 seconds
Found
time taken to find keyvals[MAXDATA-1] in
container = 1.4e-05 seconds
Not found
time taken to find a keyval doesn't exist in
container = 1.4e-05 seconds

time taken to erase keyvals[0] from container =
0.001005 seconds
time taken to erase keyvals[MAXDATA/2] from
container = 0.00375 seconds
time taken to erase keyvals[MAXDATA-1] from
container = 0.002726 seconds
```

TREE

```
MAXDATA = 1000000 items in keyvals in random
order
time taken to insert data into container =
25.3858 seconds

Found
time taken to find keyvals[0] in container = 4e-
06 seconds
Found
time taken to find keyvals[MAXDATA/2] in
container = 3.1e-05 seconds
Found
time taken to find keyvals[MAXDATA-1] in
container = 3e-05 seconds
```

```
Not found
time taken to find a keyval doesn't exist in
container = 1.4e-05 seconds

time taken to erase keyvals[0] from container =
4.9e-05 seconds
time taken to erase keyvals[MAXDATA/2] from
container = 3.1e-05 seconds
time taken to erase keyvals[MAXDATA-1] from
container = 2.8e-05 seconds
```

LINHASH

```
Container size = 1500000
MAXDATA = 1000000 items in keyvals in random
order
time taken to insert data into container =
3.12856 seconds

Found
time taken to find keyvals[0] in container = 6e-
06 seconds
Found
time taken to find keyvals[MAXDATA/2] in
container = 6e-06 seconds
Found
time taken to find keyvals[MAXDATA-1] in
container = 6e-06 seconds
Not found
time taken to find a keyval doesn't exist in
container = 9e-06 seconds

time taken to erase keyvals[0] from container =
9e-06 seconds
time taken to erase keyvals[MAXDATA/2] from
container = 5e-06 seconds
time taken to erase keyvals[MAXDATA-1] from
container = 5e-06 seconds
```

CHAINHASH

```
Container size = 500000
MAXDATA = 1000000 items in keyvals in random
order
time taken to insert data into container =
6.90041 seconds

Found
time taken to find keyvals[0] in container = 9e-
06 seconds
Found
time taken to find keyvals[MAXDATA/2] in
container = 7e-06 seconds
Found
time taken to find keyvals[MAXDATA-1] in
container = 4e-06 seconds
Not found
time taken to find a keyval doesn't exist in
container = 9e-06 seconds

time taken to erase keyvals[0] from container =
6.4e-05 seconds
time taken to erase keyvals[MAXDATA/2] from
container = 1.3e-05 seconds
time taken to erase keyvals[MAXDATA-1] from
container = 1.2e-05 seconds
```

	Insertion	Push_back	Searching	Deletion
Vector	$O(n)$	$O(1)$	$O(n)$	$O(n)$
DList	$O(1)$	$O(1)$	$O(n)$	$O(1)$
Ordered Vector	$O(n \lg_2(n))$	N/A	$O(\lg_2(n))$	$O(n)$
Binary Tree	$O(\lg_2(n))$	N/A	$O(\lg_2(n))$	$O(1)$
Linear Hashing	$O(1)$	N/A	$O(1)$	$O(1)$
Chain Hash	$O(1)$	N/A	$O(1)$	$O(1)$