**Table of Contents**

**Outputs**

**Class Examples………………………………………………………………………….. 2-3**

**Hypothesis………………………………………………………………………………. 4-13**

**Description of the Usage of Iterators with Deques………………………………… 14**

**Maps**

**Explanation and Output………………………………………………………………. 15**

**Graph of Values of Class Examples………………………………………………………… 16**

**Appendix**

**Class Example Program Codes……………………………………………….. 17-19**

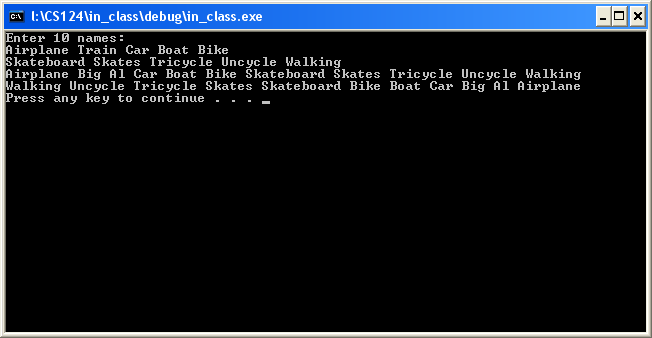
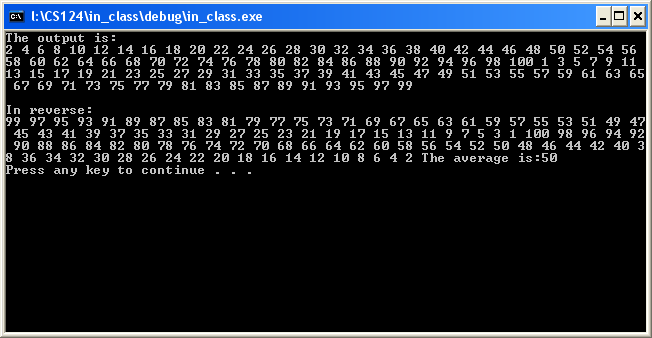
**Hypothesis Program Codes……………………………………………………. 19-26**

**Map Program Codes……………………………………………………………………. 27**

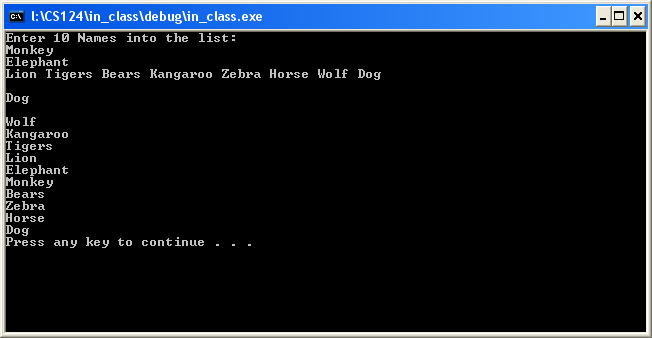
**Outputs**

**Class** **Examples**

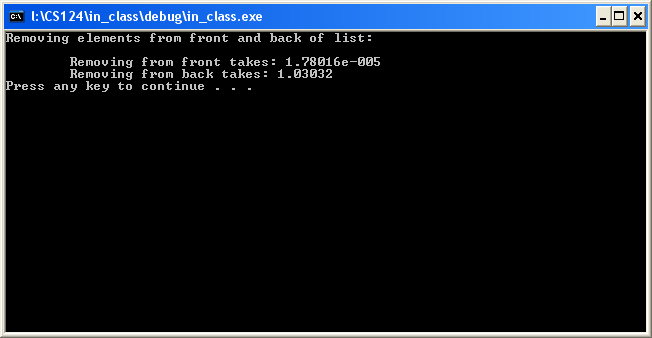
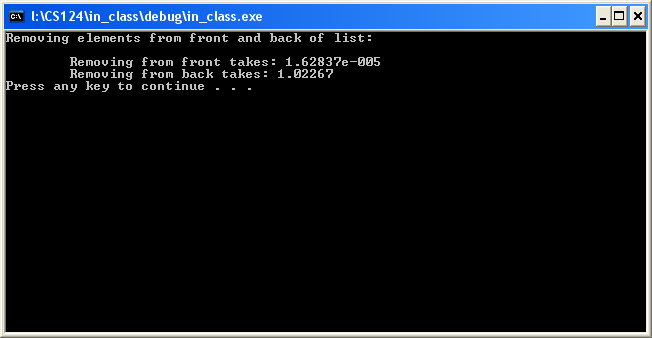
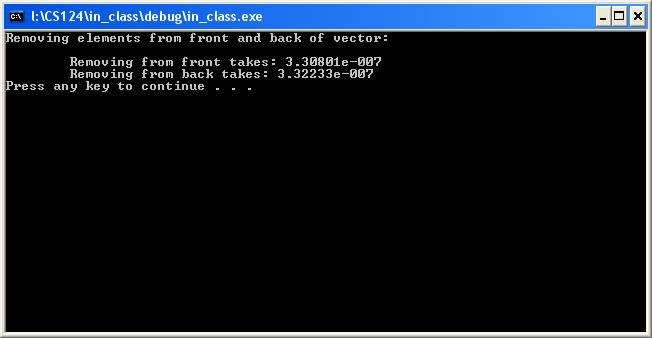
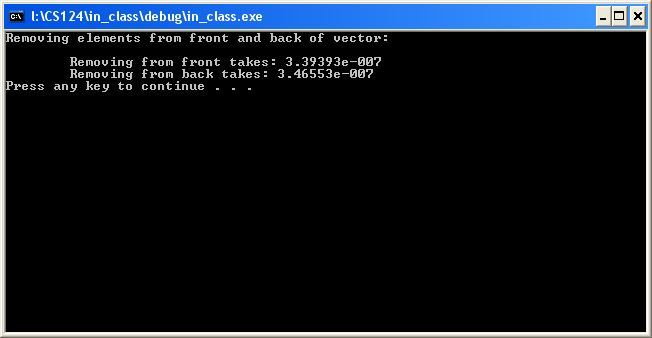
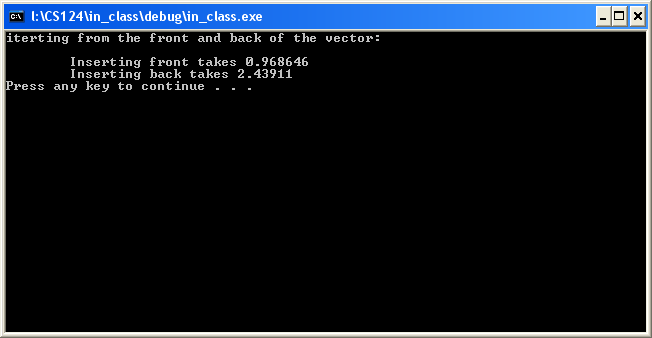
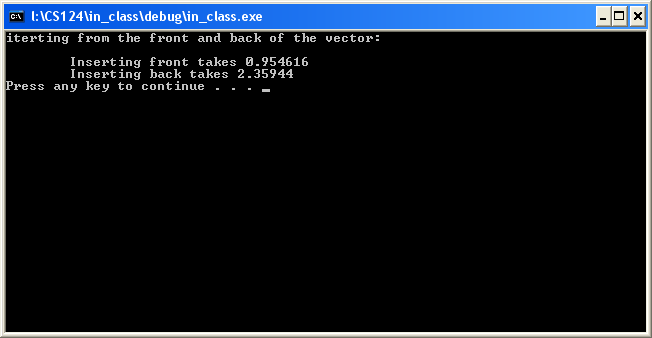
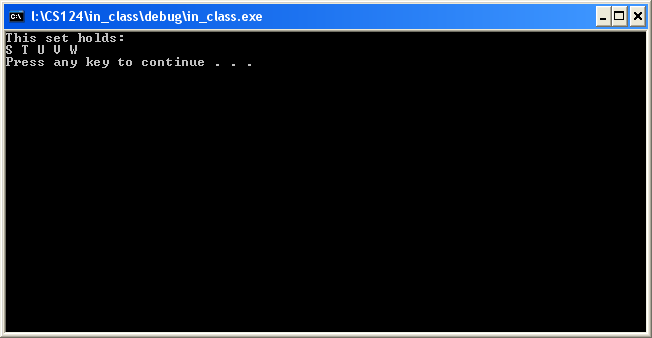
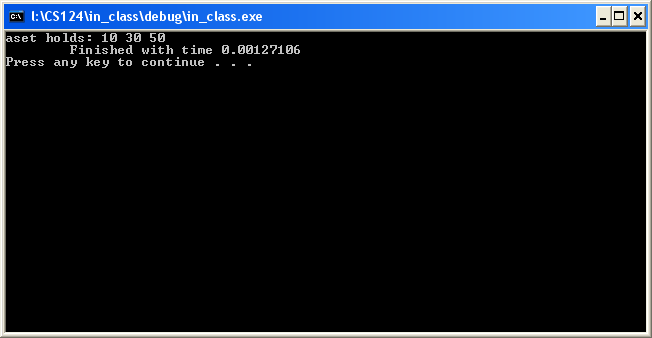
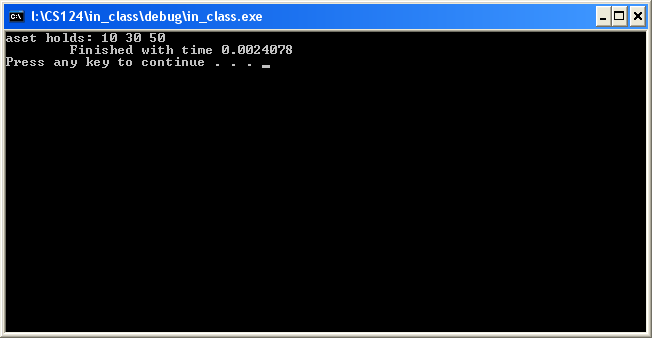
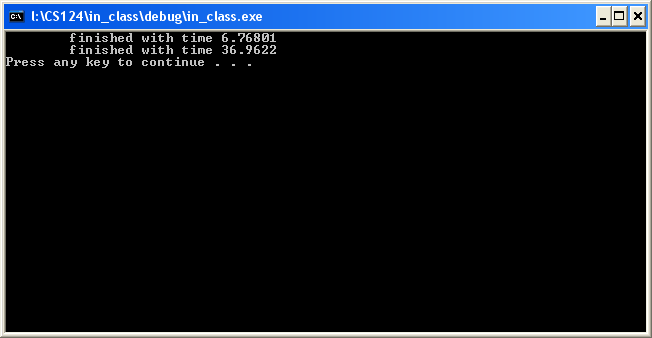
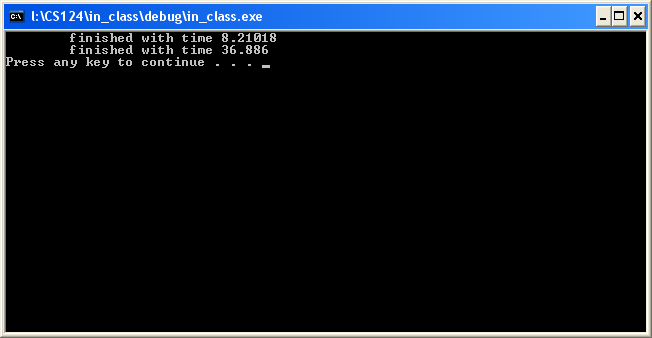
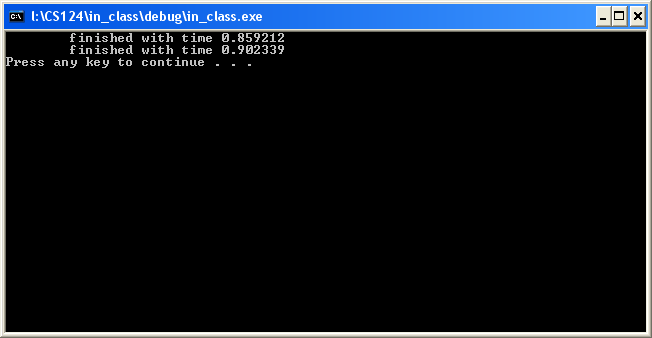
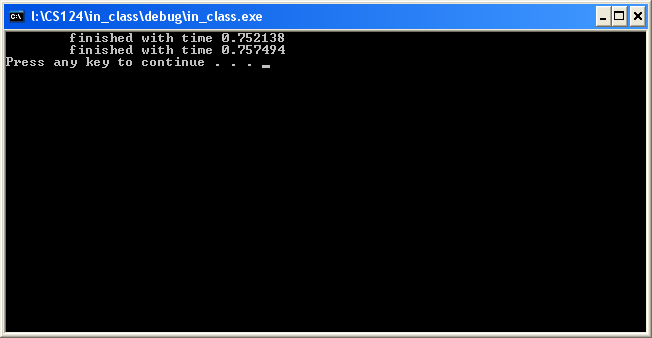
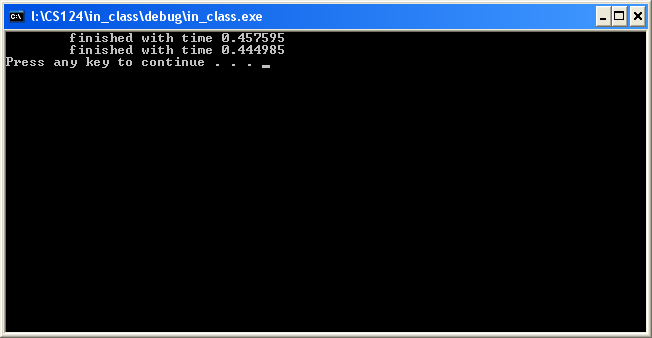
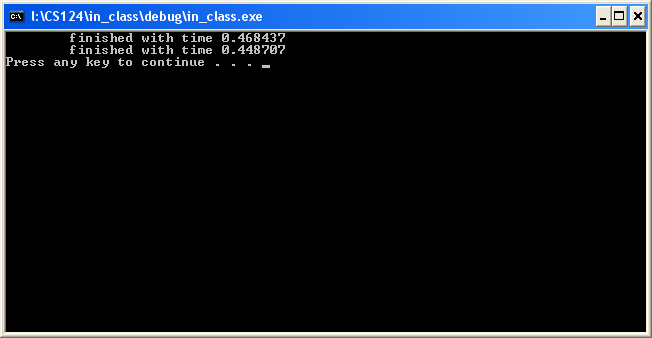
Class Exercise #1

  
  
Class Exercise #2  


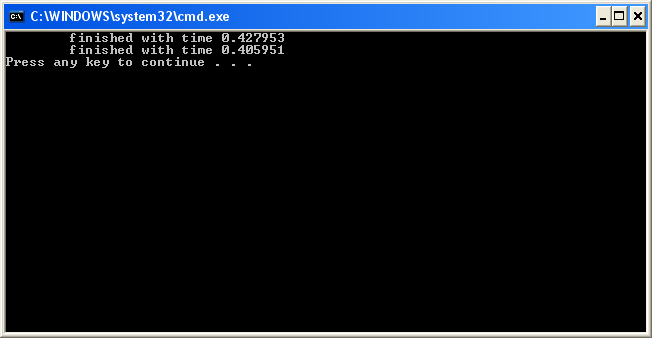
Class Exercise #3



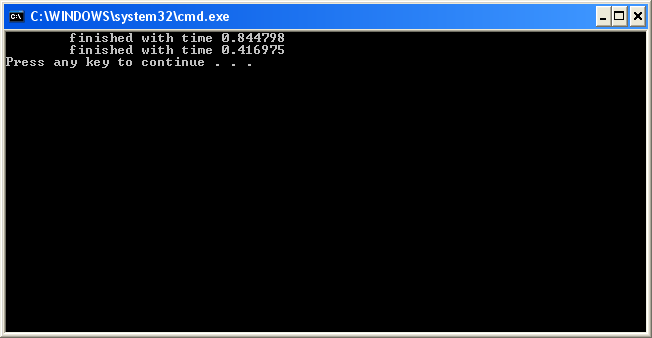
**Hypothesis**

1. Removing elements from the front of the list is quicker than removing elements from the back of the list.  
     
     
   
2. There is a significant difference when removing elements in a vector.  
     
     
   
3. It takes longer from the back to insert than in the front.  
     
     
   
4. A set will balance itself.  
   
5. The greater the number of N elements in a list, the longer the program will run.  
     
     
   
6. A) False   
     
     
   
7. B) False  
     
     
   
8. C) False  
     
     
   
9. It takes longer to remove from a set than a deque. This is because a set has to search in sequential order and a deque does not.

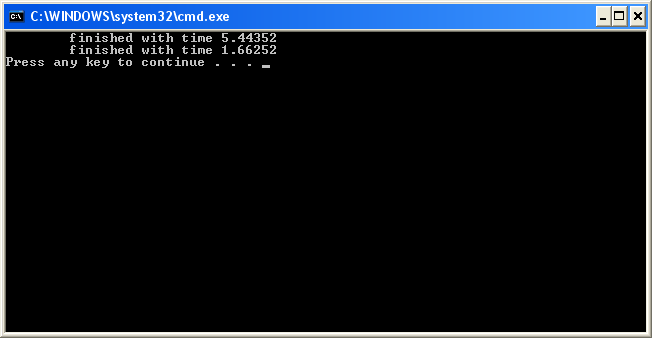
Deque: 1-100000 and 500-100000



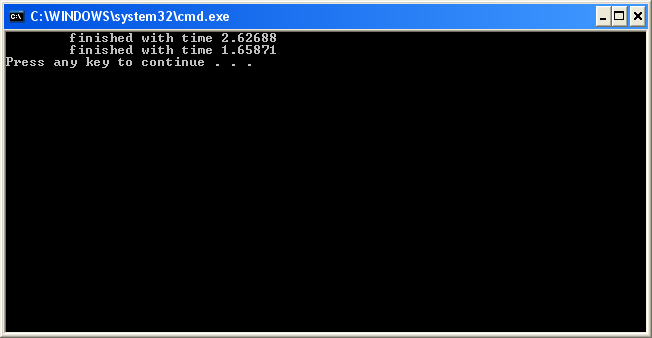
Deque: 0-200000 and 500-200000



Set: 1-100000 and 500-100000

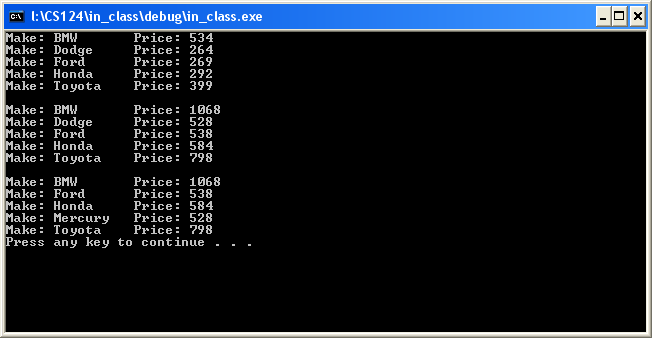


Set: 0-200000 and 500-200000



How iterators work with the Deque container: The elements contained within a deque can be accessed through random access iterators. By using a deque, the elements can be added and removed from the front or the back of a sequential list of elements. Deques have similar functionality as vectors, except that vectors cannot insert items into the front of sequence. Also, deques do not store the information contained within the sequence in sequential order in memory. Since the memory used for a deque is not store sequentially in memory, when a deque nears its capacity, the program can split the elements in memory into data chunks When compared to lists, deques perform the worse of the two.

Explanation of the “Map”: A map is another container class provided by STL to store things. It essentially works like an array, but differs greatly. A map is expandable which is only limited to how much memory you have on your device. A map also indexes your data automobile for you without you having to use any type of sort function. However, this will add some time to adding and sorting data so if time is a constraint then you will want to use your own code to perform theses action. You can also erase at any position in a map as well as between two iterators. Maps also have the ability to be compared and assigned values. Maps can be compared and assigned with the standard comparison operators. Individuals elements of a map can be examined with the [] operator.



**Graph of Values from Class Examples**

**Appendix**

Code for Class Exercise #1

#include <iostream>

#include <vector>

#include <string>

using namespace std;

void main(){

string input;

vector <string> names;

vector <string>::iterator inames;

vector <string>::reverse\_iterator rnames;

int a=0;

cout << "Enter 10 names: " << endl;

while(a<10){

cin >> input;

names.push\_back(input);

a++;

}

names.at(1)="Big Al";

for(inames=names.begin(); inames!=names.end(); ++inames){

cout << \*inames << " ";

}

cout << endl;

for(rnames=names.rbegin(); rnames!=names.rend(); ++rnames){

cout << \*rnames << " ";

}

cout << endl;

system("pause");

};

Code for Class Exercise #2

#include <iostream>

#include <deque>

using namespace std;

int main(){

deque<int>nums;

deque<int>::iterator nums\_i;

deque<int>::reverse\_iterator rnums\_i;

for (int e=101;e>=3;e--){

e--;

nums.push\_front(e);

}

for (int k=0;k<=98;k++){

k++;

nums.push\_back(k);

}

cout<<"The output is:"<<endl;

for(nums\_i=nums.begin();nums\_i!=nums.end();nums\_i++){

cout<<\*nums\_i<<" ";

}

cout<<endl<<endl;

cout<<"In reverse:"<<endl;

for(rnums\_i=nums.rbegin();rnums\_i!=nums.rend();rnums\_i++){

cout<<\*rnums\_i<<" ";

}

int temp=0, divide=1;

for(nums\_i=nums.begin();nums\_i!=nums.end();nums\_i++){

temp=\*nums\_i+temp;

divide++;

}

temp=temp/divide;

cout<<"The average is:"<<temp<<endl;

return 0;

}

Class Exercise #3

#include<iostream>

#include<list>

#include<string>

using namespace std;

int main(){

list<string> s, copy;

list<string>::iterator si;

string name;

cout << "Enter 10 Names into the list: " << endl;

for(int i=0; i<10; i++){

cin >> name;

s.push\_back(name);

}

cout << endl;

for(si=s.begin(); si != s.end(); ++ si){

if((\*si).length()%2==0){

string temp=\*si;

copy.push\_back(temp);

}

else{

string temp=\*si;

copy.push\_front(temp);

}

}

s=copy;

cout << s.front() << endl << endl;

copy.clear();

for(si=s.begin(); si != s.end(); si++){

if(\*si != "BigAl" || "bigal"){

copy.push\_back(\*si);

}

}

s=copy;

s.reverse();

for(si=s.begin(); si != s.end(); si++){

cout << \*si << " " << endl;

}

system("pause");

return 0;

}

Code for Hypothesis #1

#include <iostream>

#include <windows.h> // include this header

#include <list>

using namespace std;

int main( )

{ LARGE\_INTEGER LI1,LI2,LIF;

QueryPerformanceFrequency(&LIF);

double number = (double)LIF.HighPart \* UINT\_MAX + (double)LIF.LowPart;

double dt, fq = 1.0/number;

int f = LIF.LowPart;

list<double> listed;

list<double>::iterator listed\_i;

double a;

for(a=1.25; a<100010; ++a){

listed.push\_back(a);

}

cout << "Removing elements from front and back of list: " << endl << endl;

QueryPerformanceCounter(&LI1); // here is where we start timing

listed.pop\_front();

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tRemoving from front takes: " << dt << endl;

QueryPerformanceCounter(&LI1); // here is where we start timing

for (int a=1; a<=50000; a++)

listed.pop\_back();

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tRemoving from back takes: " << dt << endl;

system("pause");

return 0;}

Code for Hypothesis #2

#include <iostream>

#include <windows.h> // include this header

#include <vector>

using namespace std;

int main( )

{ LARGE\_INTEGER LI1,LI2,LIF;

QueryPerformanceFrequency(&LIF);

double number = (double)LIF.HighPart \* UINT\_MAX + (double)LIF.LowPart;

double dt, fq = 1.0/number;

int f = LIF.LowPart;

vector<double> v;

vector<double>::iterator vi;

double a;

//for (a=0; a<1000; ++a) v.push\_back(a);

/\*for (vi=v.begin(); vi!=v.end();++vi)

cout << \*vi << " "\*/

cout << "Removing elements from front and back of vector: " << endl << endl;

QueryPerformanceCounter(&LI1); // here is where we start timing

//v.erase(55);

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tRemoving from front takes: " << dt << endl;

QueryPerformanceCounter(&LI1); // here is where we start timing

//v.erase(899);

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tRemoving from back takes: " << dt << endl;

system("pause");

return 0;

}

Code for Hypothesis #3

//Hypothesis code one

#include <iostream>

#include <windows.h> // include this header

#include <vector>

using namespace std;

int main( )

{ LARGE\_INTEGER LI1,LI2,LIF;

QueryPerformanceFrequency(&LIF);

double number = (double)LIF.HighPart \* UINT\_MAX + (double)LIF.LowPart;

double dt, fq = 1.0/number;

int f = LIF.LowPart;

vector<double> v;

cout<<"iterting from the front and back of the vector: "<<endl<<endl;

QueryPerformanceCounter(&LI1); // here is where we start timing

for (double a=1.1; a<=25000; a++)

v.insert(v.begin(),a);

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tInserting front takes " << dt << endl;

QueryPerformanceCounter(&LI1); // here is where we start timing

for (double a=1.1; a<=25000; a++)

v.insert(v.begin(),a);

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tInserting back takes " << dt << endl;

system(“pause”);

return 0;

}

Code for Hypothesis #4

#include <iostream>

#include <set>

using namespace std;

int main()

{

set<char> s;

set<char>::iterator i;

s.insert('S');

s.insert('T');

s.insert('U');

s.insert('V');

s.insert('S');

s.insert('V');

s.insert('T');

s.insert('W');

cout << "This set holds:\n";

for (i = s.begin(); i != s.end(); i++)

cout << \*i << " ";

cout << endl;

system("pause");

return 0;

}

Code for Hypothesis #5

#include <iostream>

#include <set>

#include <windows.h>

using namespace std;

int main( )

{

LARGE\_INTEGER LI1,LI2,LIF;

QueryPerformanceFrequency(&LIF);

double number = (double)LIF.HighPart \* UINT\_MAX + (double)LIF.LowPart;

double dt, fq = 1.0/number;

int f = LIF.LowPart;

set<int> aset;

set<int>::iterator j;

QueryPerformanceCounter(&LI1); // here is where we start timing

// set some initial values:

for (int i=1; i<=5; i++) aset.insert(i\*10); // set: 10 20 30 40 50

j=aset.find(20);

aset.erase (j);

aset.erase (aset.find(40));

cout << "aset holds:";

for (j=aset.begin(); j!=aset.end(); j++)

cout << " " << \*j;

cout << endl;

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tFinished with time " << dt << endl;

system("pause");

return 0;}

}

Code for Hypothesis #6A

#include <iostream>

#include <windows.h> // include this header

#include <vector>

using namespace std;

int main( )

{

LARGE\_INTEGER LI1,LI2,LIF;

QueryPerformanceFrequency(&LIF);

double number = (double)LIF.HighPart \* UINT\_MAX + (double)LIF.LowPart;

double dt, fq = 1.0/number;

int f = LIF.LowPart;

vector<int> x;

QueryPerformanceCounter(&LI1); // here is where we start timing

for (int a=1; a<=100000; a++)

x.insert(x.begin(),a);

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tfinished with time " << dt << endl;

QueryPerformanceCounter(&LI1); // here is where we start timing

for (int a=100000; a>=1; a--)

x.insert(x.begin(),a);

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tfinished with time " << dt << endl;

system("pause");

return 0;

}

Code for Hypothesis #6B

#include <iostream>

#include <windows.h> // include this header

#include <list>

using namespace std;

int main( )

{

LARGE\_INTEGER LI1,LI2,LIF;

QueryPerformanceFrequency(&LIF);

double number = (double)LIF.HighPart \* UINT\_MAX + (double)LIF.LowPart;

double dt, fq = 1.0/number;

int f = LIF.LowPart;

list<int> x;

QueryPerformanceCounter(&LI1); // here is where we start timing

for (int i=1; i<=100000; i++)

x.insert(x.begin(),i);

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tfinished with time " << dt << endl;

QueryPerformanceCounter(&LI1); // here is where we start timing

for (int a=100000; a>=1; a--)

x.insert(x.begin(),a);

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tfinished with time " << dt << endl;

system("pause");

return 0;

}

Code for Hypothesis #6C

#include <iostream>

#include <windows.h>

#include <deque>

using namespace std;

int main( )

{

LARGE\_INTEGER LI1,LI2,LIF;

QueryPerformanceFrequency(&LIF);

double number = (double)LIF.HighPart \* UINT\_MAX + (double)LIF.LowPart;

double dt, fq = 1.0/number;

int f = LIF.LowPart;

deque<int> d;

QueryPerformanceCounter(&LI1); // here is where we start timing

for (int a=1; a<=100000; a++)

d.insert(d.begin(),a);

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tfinished with time " << dt << endl;

QueryPerformanceCounter(&LI1); // here is where we start timing

for (int a=100000; a>=1; a--)

d.insert(d.begin(),a);

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tfinished with time " << dt << endl;

system("pause");

return 0;

}

Code for Hypothesis #7 Deque

#include <iostream>

#include <windows.h> // include this header

#include <deque>

using namespace std;

int main( )

{ LARGE\_INTEGER LI1,LI2,LIF;

QueryPerformanceFrequency(&LIF);

double number = (double)LIF.HighPart \* UINT\_MAX + (double)LIF.LowPart;

double dt, fq = 1.0/number;

int f = LIF.LowPart;

deque<int> d;

deque<int>::iterator di;

QueryPerformanceCounter(&LI1); // here is where we start timing

for (int a=0; a<=200000; a++)

d.insert(d.begin(),a);

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tfinished with time " << dt << endl;

QueryPerformanceCounter(&LI1); // here is where we start timing

for (int a=500; a<=100000; a++)

di.remove(600);

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tfinished with time " << dt << endl;

return 0;

}

Code for Hypothesis #7 Set

#include <iostream>

#include <windows.h> // include this header

#include <set>

using namespace std;

int main( )

{ LARGE\_INTEGER LI1,LI2,LIF;

QueryPerformanceFrequency(&LIF);

double number = (double)LIF.HighPart \* UINT\_MAX + (double)LIF.LowPart;

double dt, fq = 1.0/number;

int f = LIF.LowPart;

set<int> s;

set<int>::iterator si;

QueryPerformanceCounter(&LI1); // here is where we start timing

for (int a=0; a<=200000; a++)

s.insert(s.begin(),a);

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tfinished with time " << dt << endl;

QueryPerformanceCounter(&LI1); // here is where we start timing

for (int a=500; a<=100000; a++)

si.erase(600);

QueryPerformanceCounter(&LI2); // here is where we finish timing

dt = ( ( (double)(LI2.HighPart)\*UINT\_MAX + (double)LI2.LowPart )

- ( (double)(LI1.HighPart)\*UINT\_MAX + (double)LI1.LowPart ) )\*fq;

cout << "\tfinished with time " << dt << endl;

return 0;

Code for Map

#include <iostream>

#include <map>

#include <string>

using namespace std;

int main( ){

typedef map<string, float> mymap;

mymap::iterator pos;

mymap s;

s["Ford"]=269;

s["Dodge"]=264;

s["Toyota"]=399;

s["BMW"]=534;

s["Honda"]=292;

for(pos = s.begin(); pos != s.end(); ++pos){

cout << "Make: " << pos->first << "\t" << "Price: " << pos->second << endl;

}

cout << endl;

for (pos = s.begin(); pos != s.end(); ++ pos){

pos ->second \*=2;

}

for (pos = s.begin(); pos != s.end(); ++pos){

cout << "Make: " << pos->first << "\t" <<

"Price: " << pos->second << endl;

}

cout << endl;

s["Mercury"]=s["Dodge"];

s.erase("Dodge");

for (pos = s.begin(); pos != s.end(); ++pos){

cout << "Make: " << pos->first << "\t" <<

"Price: " << pos->second << endl;

}

system("pause");

return 0;

}