Matthew Wills

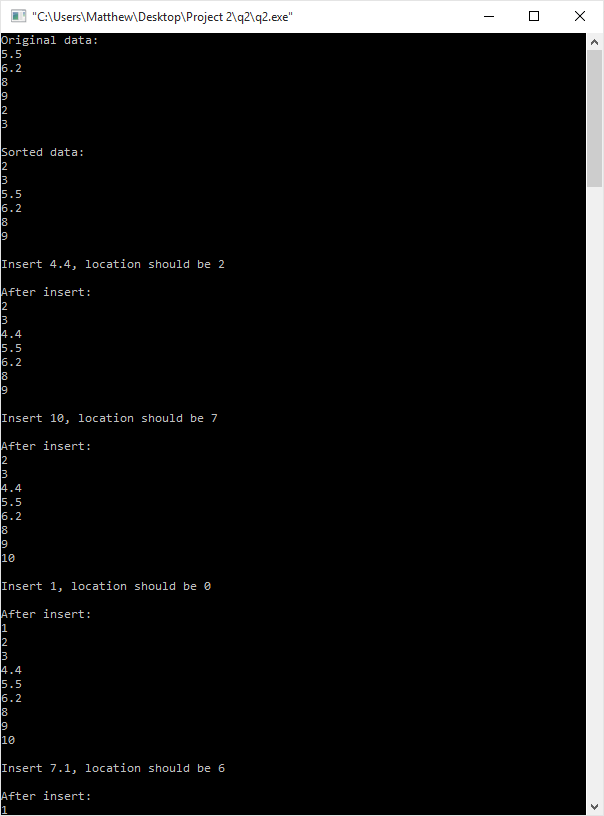
Compiler used: mingw

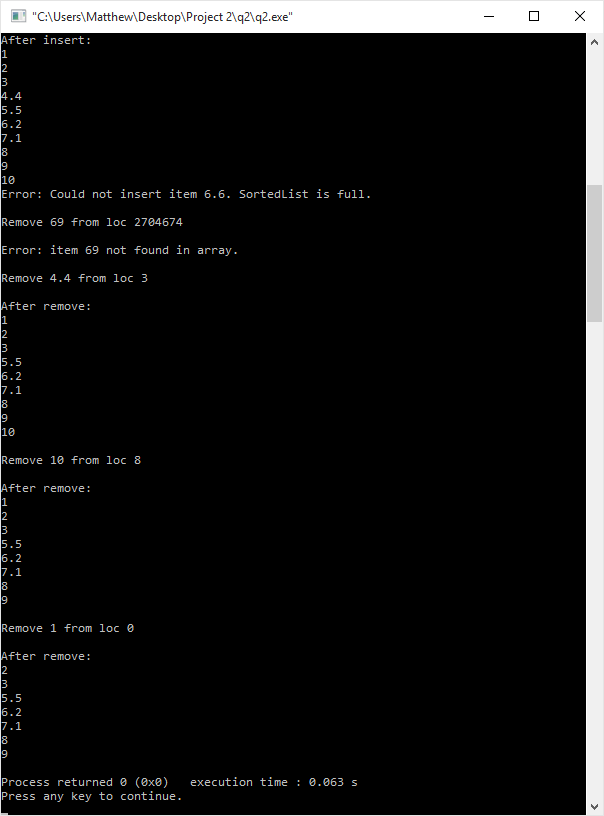
System Specs:

Windows 10 64bit

AMD Phenom II X4 970 Processor 3.50 GHz

10.0 GB RAM





**Source code:**

**q2.cpp**

#include <iostream>

#include <fstream>

#include "SortedList.h"

using namespace std;

int main()

{

SortedList floatList;

//Read in the data from float.dat and save it to the values[] array in floatList

float line;

ifstream myfile ("float.dat");

if (myfile.is\_open())

{

cout << "Original data: " << endl;

for(int i = 0; i < MAX\_ITEMS; i++)

{

myfile >> line;

floatList.values[i] = line;

floatList.length++;

cout << line << endl;

if (myfile.eof()) break;

}

cout << endl << "Sorted data: " << endl;

floatList.selectionSort(floatList.values);

floatList.insertItem(4.4);

floatList.insertItem(10.0);

floatList.insertItem(1.0);

floatList.insertItem(7.1);

floatList.insertItem(6.6);

floatList.deleteItem(69);

floatList.deleteItem(4.4);

floatList.deleteItem(10.0);

floatList.deleteItem(1.0);

}

else cout << "Unable to open file." << endl;

ofstream outputFile;

outputFile.open("output.dat");

if (outputFile.is\_open())

{

for(int i = 0; i < floatList.length; i++)

{

//write each value into the output file output.dat

outputFile << floatList.values[i];

//makes new line after writing each value, excluding the last.

if(i != floatList.length - 1)

outputFile << '\n';

}

}

outputFile.close();

return 0;

}

**SortedList.h**

#include <iostream>

#define MAX\_ITEMS 10

using namespace std;

typedef float ItemType;

class SortedList

{

public:

int length;

ItemType values[MAX\_ITEMS];

int currentPos;

bool found;

SortedList();

void makeEmpty();

void insertItem(ItemType x);

void deleteItem(ItemType x);

bool isFull();

int lengthIs();

void retrieveItem(ItemType &x, bool &found);

void resetList();

void getNextItem(ItemType &x);

void selectionSort(float x[]);

void shiftRight(ItemType x[], int loc);

void shiftLeft(ItemType x[], int loc);

};

//returns the location where a given number should reside in the array

int binarySearch(ItemType x[], ItemType item, int startPoint, int endPoint)

{

int midPoint = startPoint + (endPoint - startPoint)/2;

if (item == x[midPoint])

return midPoint;

if (item < x[midPoint])

{

endPoint = midPoint - 1;

if (endPoint < startPoint)

return midPoint;

return binarySearch(x, item, startPoint, endPoint);

}

if (item > x[midPoint])

{

startPoint = midPoint + 1;

if (startPoint > endPoint)

return midPoint;

return binarySearch(x, item, startPoint, endPoint);

}

}

//makes the constructor initialize the length to 0 and currentPos to -1 for every new SortedList object

SortedList::SortedList()

{

length = 0;

currentPos = -1;

}

//shifts all values in the array to the right beginning at a specified location.

void SortedList::shiftRight(ItemType x[], int loc)

{

for(int i = MAX\_ITEMS-1; i > loc; i--)

{

ItemType temp = x[i];

x[i] = x[i-1];

x[i-1] = temp;

}

}

//shifts all values in the array to the left beginning at a specified location.

void SortedList::shiftLeft(ItemType x[], int loc)

{

for(int i = loc; i < MAX\_ITEMS-1; i++)

{

x[i] = x[i+1];

}

}

//Makes a list empty by setting its length to 0.

void SortedList::makeEmpty()

{

length = 0;

}

//insert x into list and make use binary search to find where it should go

void SortedList::insertItem(ItemType x)

{

//if values[] is full, output error message.

if (isFull())

cout << "Error: Could not insert item " << x << ". SortedList is full." << endl;

//if values is not empty, use binary search to find the position in the array where x belongs, then set values[thatposition] = x.

else

{

int loc = binarySearch(values, x, 0, length);

cout << endl << "Insert " << x << ", location should be " << loc << endl;

shiftRight(values, loc);

values[binarySearch(values, x, 0, length)] = x;

length++;

//print the array after the insert

cout << endl << "After insert: " << endl;

for(int i = 0; i < length; i++)

{

cout << values[i] << endl;

}

}

}

//Remove an item from the list. Use binary search to find it.

void SortedList::deleteItem(ItemType x)

{

//if values[] is empty, output error message.

if (length == 0)

cout << "Error: Could not remove item, array is empty." << endl;

else

{

int loc = binarySearch(values, x, 0, length);

cout << endl << "Remove " << x << " from loc " << loc << endl;

//Check to make sure the value is in the list

for(int i = 0; i < MAX\_ITEMS-1; i++)

{

//if it is in the list, shift everything to the right of it one space to the left and decrement length.

if(loc == i)

{

shiftLeft(values, loc);

length--;

//print the array after the removal

cout << endl << "After remove: " << endl;

for(int i = 0; i < length; i++)

{

cout << values[i] << endl;

}

return;

}

}

//If the item is not found in the array, print error message.

cout << endl << "Error: item " << x << " not found in array." << endl;

}

}

//if length of list = MAX\_ITEMS, the list must be full

bool SortedList::isFull()

{

if (length == MAX\_ITEMS)

return true;

else

return false;

}

int SortedList::lengthIs()

{

return length;

}

//binary search list for x, return found. bool result is stored in found.

void SortedList::retrieveItem(ItemType &x, bool &found)

{

int loc = binarySearch(values, x, 0, length);

found = false;

//check if the value is in the array

for(int i = 0; i < MAX\_ITEMS-1; i++)

{

//if it is in the list, set found to true;

if(loc == i)

{

found = true;

cout << endl << x << " was found at location " << loc << endl;

return;

}

}

//If the item is not found in the array, print error message.

cout << endl << x << " was not found in array." << endl;

}

//resets your position in the list

void SortedList::resetList()

{

currentPos = -1;

}

//get the next element in the list with respect to currentPos.

void SortedList::getNextItem(ItemType &x)

{

x = values[currentPos + 1];

}

void SortedList::selectionSort(float x[])

{

float temp;

for (int i = 0; i < length; i++)

{

for(int j = i + 1; j < length; j++)

{

if (x[i] > x[j])

{

temp = x[i];

x[i] = x[j];

x[j] = temp;

}

}

cout << x[i] << endl;

}

}

**Data Structure and Algorithm:**

Data Structure, only one class is used.

**Instance variables are**

int length;

ItemType values[MAX\_ITEMS];

int currentPos;

bool found;

**functions used are**

void makeEmpty();

void insertItem(ItemType x);

void deleteItem(ItemType x);

bool isFull();

int lengthIs();

void retrieveItem(ItemType &x, bool &found);

void resetList();

void getNextItem(ItemType &x);

void selectionSort(float x[]);

void shiftRight(ItemType x[], int loc);

void shiftLeft(ItemType x[], int loc);

For algorithms, I wrote binary search, selection sort, and functions to shift values left and right in the array.

**Binary search**

//returns the location where a given number should reside in the array

int binarySearch(ItemType x[], ItemType item, int startPoint, int endPoint)

{

int midPoint = startPoint + (endPoint - startPoint)/2;

if (item == x[midPoint])

return midPoint;

if (item < x[midPoint])

{

endPoint = midPoint - 1;

if (endPoint < startPoint)

return midPoint;

return binarySearch(x, item, startPoint, endPoint);

}

if (item > x[midPoint])

{

startPoint = midPoint + 1;

if (startPoint > endPoint)

return midPoint;

return binarySearch(x, item, startPoint, endPoint);

}

}

**Selection Sort**

void SortedList::selectionSort(float x[])

{

float temp;

for (int i = 0; i < length; i++)

{

for(int j = i + 1; j < length; j++)

{

if (x[i] > x[j])

{

temp = x[i];

x[i] = x[j];

x[j] = temp;

}

}

cout << x[i] << endl;

}

}

**Shift Right**

void SortedList::shiftRight(ItemType x[], int loc)

{

for(int i = MAX\_ITEMS-1; i > loc; i--)

{

ItemType temp = x[i];

x[i] = x[i-1];

x[i-1] = temp;

}

}

**Shift Left**

void SortedList::shiftLeft(ItemType x[], int loc)

{

for(int i = loc; i < MAX\_ITEMS-1; i++)

{

x[i] = x[i+1];

}

}