Alex McNurlin  
CS121  
Programming assignment 6  
5/6/16

# Programming Log

Time:

\* May 5, 2:30 – 5:30, 9:30 – 11:30. **5 hours**

Things encountered/learned

* Trying to overwrite values outside an array can cause problems! One of the arrays in the code given was only 25 elements wide, but it needed to be 10,000
* A quicksort doesn't work very well when the data is already sorted.

# Program Design

The code was taken from the course website and modified as necessary.

selSort.cpp:

* SelectionSort: Initiates a selection sort.
* IndexOfSmallest: Finds the index of the smallest element in the array.
* PrintArray: Prints the contents of the array

mergeSort.cpp:

* Mergesort: Initiates the merge sort recursively. Breaks the array into smaller arrays and launches the merge function.
* Merge: Merges the given arrays into a bigger array.
* Swap: Not used
* PrintArray: Prints the contents of the array

quickSort.cpp:

* Quisort: Initiate the quicksort. Swaps values based on a pivot and partitions the array using recursion.
* Pivot: Swaps values less than the pivot with those above the pivot.
* Swap: Swaps the two given values
* PrintArray: Prints the contents of the array

Data files:

* numbers\_10k.txt: 10,000 random numbers generated on a random number generating website (www.random.org/integers)
* numbers\_1k.txt: A subset of 1,000 numbers taken from numbers\_10k.txt
* numbers\_100.txt: A subset of 100 numbers taken from numbers\_10k.txt
* low\_to\_high\_10k.txt: A file created by writing the sorted contents of numbers\_10k.txt to a file.
* low\_to\_high\_1k.txt: A subset of 1,000 numbers from low\_to\_high\_10k
* low\_to\_high\_100.txt: A subset of 1,00 numbers from low\_to\_high\_10k
* high\_to\_low\_10k.txt: A file created by modifying the code to sort in reverse and writing to a file.
* high\_to\_low\_1k.txt: A subset of 1,000 numbers from high\_to\_low\_10k.txt
* high\_to\_low\_100.txt: A subset of 100 numbers from high\_to\_low\_10k.txt

# Summary of findings

The chart shows the runtime, number of swaps, and number of comparisons for each of the functions.  Selections sort was relatively innefficient with the highest runtime and the highest number of comparisons. However, it used the lowest number of swaps. This would be useful in a situation where swapping values would take a lot of time/resources.

Merge sort was much faster than the selection sort, and had far less comparisons than selection sort. The number of comparisons was also less than a quick sort, but not by very much. This seems to have no particular advantages, but also no clear disadvantages.

Quick sort was by far the fastest sort. When the numbers were random, the time to sort was definitely faster than merge sort and much much faster than the selection sort. The number of comparisons and swaps were comparable to a merge sort, although slightly more. This sort method faced a clear disadvantage when sorting already sorted or reversed data, and was comparable to the selection sorts in that respect.

# Code – Ouput

Script started on Fri 06 May 2016 08:48:48 AM PDT

alexmcnurlin@Laquisha:~/Google Drive/schoolStuff/cs121/hw6$ ./selSort

Enter filename: numbers\_100.txt

Sorting...

SelectionSort() finished

------------------------

[ 83, 699, 748, 1999, 4057, 5506, 5834, 5937, 8481, 11917, 12967, 13004, 13384, 13467, 13574, 15206, 15243, 16535, 17265, 19014, 19436, 20550, 23987, 23997, 24302, 24750, 25523, 25661, 27139, 31387, 32282, 32285, 32820, 33260, 33546, 33763, 33830, 38305, 41169, 41779, 42096, 42363, 42612, 42624, 43018, 43489, 44108, 45352, 50284, 50580, 51468, 51701, 52528, 52563, 53433, 53798, 55307, 55821, 56197, 58982, 62096, 62578, 63422, 64079, 64308, 64934, 65847, 66223, 68289, 68354, 70128, 70552, 72397, 74343, 74638, 75904, 76095, 76161, 76167, 76624, 77317, 80681, 80706, 81583, 82175, 82945, 85540, 85917, 86301, 88075, 88477, 89507, 91101, 91152, 93171, 94780, 95845, 97042, 99253 ]

Number of comparisons: 5150

Number of swaps: 94

alexmcnurlin@Laquisha:~/Google Drive/schoolStuff/cs121/hw6$ ./mergeSort

Enter filename: numbers\_100.txt

Sorting...

[ 83, 699, 748, 1999, 4057, 5506, 5834, 5937, 8481, 11917, 12967, 13004, 13384, 13467, 13574, 15206, 15243, 16535, 17265, 19014, 19436, 20550, 23987, 23997, 24302, 24750, 25523, 25661, 27139, 31387, 32282, 32285, 32820, 33260, 33546, 33763, 33830, 38305, 41169, 41779, 42096, 42363, 42612, 42624, 43018, 43489, 44108, 45352, 50284, 50580, 51468, 51701, 52528, 52563, 53433, 53798, 55307, 55821, 56197, 58982, 62096, 62578, 63422, 64079, 64308, 64934, 65847, 66223, 68289, 68354, 70128, 70552, 72397, 74343, 74638, 75904, 76095, 76161, 76167, 76624, 77317, 80681, 80706, 81583, 82175, 82945, 85540, 85917, 86301, 88075, 88477, 89507, 91101, 91152, 93171, 94780, 95845, 97042, 99253 ]

Number of comparisons: 544

Number of swaps: 257

alexmcnurlin@Laquisha:~/Google Drive/schoolStuff/cs121/hw6$ ./quickSort

Enter filename: numbers\_100.txt

Sorting...

[ 83, 699, 748, 1999, 4057, 5506, 5834, 5937, 8481, 11917, 12967, 13004, 13384, 13467, 13574, 15206, 15243, 16535, 17265, 19014, 19436, 20550, 23987, 23997, 24302, 24750, 25523, 25661, 27139, 31387, 32282, 32285, 32820, 33260, 33546, 33763, 33830, 38305, 41169, 41779, 42096, 42363, 42612, 42624, 43018, 43489, 44108, 45352, 50284, 50580, 51468, 51701, 52528, 52563, 53433, 53798, 55307, 55821, 56197, 58982, 62096, 62578, 63422, 64079, 64308, 64934, 65847, 66223, 68289, 68354, 70128, 70552, 72397, 74343, 74638, 75904, 76095, 76161, 76167, 76624, 77317, 80681, 80706, 81583, 82175, 82945, 85540, 85917, 86301, 88075, 88477, 89507, 91101, 91152, 93171, 94780, 95845, 97042, 99253 ]

Number of comparisons: 651

Number of swaps: 373

]0;alexmcnurlin@Laquisha: ~/Google Drive/schoolStuff/cs121/hw6alexmcnurlin@Laquisha:~/Google Drive/schoolStuff/cs121/hw6$ exit

Script done on Fri 06 May 2016 08:49:56 AM PDT

# Code – selSort.cpp

/\* selSort.cpp

\*

\* Selection Sort test program.

\*

\* Bruce M. Bolden

\* November 16, 1998

\*

\* Edited by Alex McNurlin

\* May 5, 2016

\* Most lines edited are commented with 4 slashes (////)

\* and a comment on what was changed

\*/

#include <iostream>

#include <fstream> //// Added library for main

#include <cstdlib> //// Added library for main

using namespace std; ////

#define aType int

// prototypes

void SelectionSort( aType A[], int nElements );

int IndexOfSmallest( aType A[], int iStart, int iEnd );

void PrintArray( aType A[], int nElements );

int c = 0; //// c is the number of comparisons, s is the number of swaps

int s = 0; //// Added these lines

int main()

{

//// Begin edited code

char filename[32];

cout << "Enter filename: ";

cin >> filename;

ifstream in\_file(filename);

if ( in\_file.fail() ) {

cout << "File " << filename << " is invalid "

<< endl << "Exiting..." << endl;

exit(1);

}

// Read in lines until end of file

char in\_char[32];

int in\_int = -1;

// The size of array\_to\_sort will always be 10000,

// but the size will be recorded in size. Also

int array\_to\_sort[10001];

int size = 0;

do {

size++;

in\_file.getline( in\_char, 32 );

in\_int = atoi(in\_char);

array\_to\_sort[size-1] = in\_int;

} while ( !in\_file.eof() );

// for some reason, the program always reads in one more line thats empty.

// This fixes that

size--;

SelectionSort( array\_to\_sort, size );

cout << "Number of comparisons: " << c << endl;

cout << "Number of swaps: " << s << endl;

//// End edited code

}

/\* SelectionSort: Sort and array, A, using a selection

\* sort algorithm.

\*/

void SelectionSort( aType A[], int nElements )

{

int iSmallest;

aType tmp;

cout << "------------------------" << endl;

cout << "In SelectionSort():" << endl;

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

{

cout << " Pass: " << i << endl;

iSmallest = IndexOfSmallest( A, i, nElements-1 );

// swap

c++; //// Add counter for comparisons

if( iSmallest > i )

{

s++; //// Added this line

tmp = A[i];

A[i] = A[iSmallest];

A[iSmallest] = tmp;

}

//PrintArray( A, nElements ); //// Removed this line

}

cout << "SelectionSort() finished" << endl;

cout << "------------------------" << endl;

//PrintArray( A, nElements );

}

/\* IndexOfSmallest: Find index of smallest value.

\*/

int IndexOfSmallest( aType A[], int iStart, int iEnd )

{

int index = -1;

aType aMin = A[iStart];

for( int i = iStart ; i <= iEnd ; i++ )

{

c++;

if( A[i] < aMin )

{

aMin = A[i];

index = i;

}

}

cout << "IndexOfSmallest: " << index << endl;

return index;

}

/\* Print contents of an array.

\*/

void PrintArray( aType A[], int nElements )

{

cout << "[ ";

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

{

cout << A[i] ;

if( i < nElements-1 )

cout << ", ";

}

cout << " ] " << endl;

}

# Code – mergeSort.cpp

/\* mergeSort.cpp

\*

\* Bruce M. Bolden

\* November 18, 1998

\*

\* Edited by Alex McNurlin

\* May 5, 2016

\* Most lines edited are commented with 4 slashes (////)

\* and a comment on what was changed

\*/

#include <iostream>

#include <fstream> //// Added library for main

#include <cstdlib> //// Added library for main

using namespace std; ////

typedef int aType; // array type

const int MAX\_ARRAY = 10000; //// This was bumped up to be big enough

// prototypes

void Mergesort( aType a[], int first, int last );

void Merge( aType a[],

int firstLeft, int lastLeft,

int firstRight, int lastRight );

void Swap( aType &v1, aType &v2 );

void PrintArray( aType A[], int nElements );

int c = 0; //// c is the number of comparisons, s is the number of swaps

int s = 0; //// Added these lines

int main()

{

//// Begin edited code

char filename[32];

cout << "Enter filename: ";

cin >> filename;

ifstream in\_file(filename);

if ( in\_file.fail() ) {

cout << "File " << filename << " is invalid " << endl << "Exiting..." << endl;

exit(1);

}

// Read in lines until end of file

char in\_char[32];

int in\_int = -1;

// The size of array\_to\_sort will always be 10000,

// but the size will be recorded in size. Also

int array\_to\_sort[10001];

int size = 0;

do {

size++;

in\_file.getline( in\_char, 32 );

in\_int = atoi(in\_char);

array\_to\_sort[size-1] = in\_int;

} while ( !in\_file.eof() );

// for some reason, the program always reads in one more line thats empty.

// This fixes that

size--;

Mergesort( array\_to\_sort, 0, size-1 );

PrintArray( array\_to\_sort, size-1 );

cout << "Number of comparisons: " << c << endl;

cout << "Number of swaps: " << s << endl;

//// End edited code

////cout << "\nFinal array contents:" << endl;

////PrintArray( testArray, nA );

}

/\* Mergesort: Do a mergesort on array a.

\* Note: This is a recursive routine.

\*/

void Mergesort( aType a[], int first, int last )

{

int middle;

////cout << "\nMergesort:" << endl;

////cout << "Mergesort::first: " << first << endl;

////cout << "Mergesort::last: " << last << endl;

if( first < last ) {

middle = (first + last)/2;

cout << "Mergesort::middle: " << middle << endl;

Mergesort( a, first, middle);

Mergesort( a, middle+1, last );

Merge( a, first, middle, middle+1, last );

}

}

/\* Merge: Merge two segments of an array together.

\*/

void Merge( aType a[],

int firstLeft, int lastLeft,

int firstRight, int lastRight )

{

aType tempArray[MAX\_ARRAY];

int index = firstLeft;

int firstSave = firstLeft;

////cout << "Merge::firstLeft: " << firstLeft << endl;

////cout << "Merge::lastLeft: " << lastLeft << endl;

////cout << "Merge::firstRight: " << firstRight << endl;

////cout << "Merge::lastRight: " << lastRight << endl;

// Merge segments (array subsets)

while( (firstLeft <= lastLeft) && (firstRight <= lastRight) )

{

c++;//// Added comparison count

if( a[firstLeft] < a[firstRight] )

{

tempArray[index] = a[firstLeft];

firstLeft++;

}

else

{

s++; //// Added swap count

tempArray[index] = a[firstRight];

firstRight++;

}

index++;

}

// Copy remainder of left array into tempArray

while( firstLeft <= lastLeft )

{

tempArray[index] = a[firstLeft];

firstLeft++;

index++;

}

// Copy remainder of right array into tempArray

while( firstRight <= lastRight )

{

tempArray[index] = a[firstRight];

firstRight++;

index++;

}

// Copy back into original array

for( index = firstSave ; index <= lastRight ; index++ )

a[index] = tempArray[index];

}

/\* Swap: Swap two items (by reference).

\*/

void Swap( aType &v1, aType &v2 )

{

aType tmpVal;

tmpVal = v1;

v1 = v2;

v2 = tmpVal;

}

/\* PrintArray: Print contents of an array.

\*/

void PrintArray( aType A[], int nElements )

{

cout << "[ ";

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

{

cout << A[i] ;

if( i < nElements-1 )

cout << ", ";

}

cout << " ] " << endl;

}

# Code – quickSort.cpp

/\* quickSort.cpp

\*

\* Bruce M. Bolden

\* November 18, 1998

\*

\* Edited by Alex McNurlin

\* May 5, 2016

\* Most lines edited are commented with 4 slashes (////)

\* and a comment on what was changed

\*/

#include <iostream>

#include <fstream> //// Added library for main

#include <cstdlib> //// Added library for main

using namespace std; ////

typedef int aType;

// prototypes

void Quicksort( aType a[], int first, int last );

int Pivot( aType a[], int first, int last );

void Swap( aType &v1, aType &v2 );

void PrintArray( aType A[], int nElements );

int c = 0; //// c is the number of comparisons, s is the number of swaps

int s = 0; //// Added these lines

int main()

{

//// Begin edited code

char filename[32];

cout << "Enter filename: ";

cin >> filename;

ifstream in\_file(filename);

if ( in\_file.fail() ) {

cout << "File " << filename << " is invalid "

<< endl << "Exiting..." << endl;

exit(1);

}

// Read in lines until end of file

char in\_char[32];

int in\_int = -1;

// The size of array\_to\_sort will always be 10000,

// but the size will be recorded in size. Also

int array\_to\_sort[10001];

int size = 0;

do {

size++;

in\_file.getline( in\_char, 32 );

in\_int = atoi(in\_char);

//cout << in\_int << " " << size-1 << endl;

array\_to\_sort[size-1] = in\_int;

} while ( !in\_file.eof() );

// for some reason, the program always reads in one more line thats empty.

// This fixes that

size--;

Quicksort( array\_to\_sort, 0, size-1 );

PrintArray( array\_to\_sort, size-1 );

cout << "Number of comparisons: " << c << endl;

cout << "Number of swaps: " << s << endl;

//// End edited code

}

/\* Quicksort: Sort an array a, using the quicksort

\* algorithm.

\*/

void Quicksort( aType a[], int first, int last )

{

int pivot;

if( first < last ) {

pivot = Pivot( a, first, last );

Quicksort( a, first, pivot-1 );

Quicksort( a, pivot+1, last );

}

}

/\* Pivot: Find and return the index of pivot element.

\*/

int Pivot( aType a[], int first, int last )

{

int p = first;

aType pivot = a[first];

for( int i = first+1 ; i <= last ; i++ ) {

c++;//// Added comparison count

if( a[i] <= pivot ) {

p++;

Swap( a[i], a[p] );

}

}

Swap( a[p], a[first] );

return p;

}

/\* Swap: Swap two item (by reference).

\*/

void Swap( aType &v1, aType &v2 )

{

aType tmpVal;

s++; //// Added to swap count

tmpVal = v1;

v1 = v2;

v2 = tmpVal;

}

/\* PrintArray: Print contents of an array.

\*/

void PrintArray( aType A[], int nElements )

{

cout << "[ ";

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

{

cout << A[i] ;

if( i < nElements-1 )

cout << ", ";

}

cout << " ] " << endl;

}