Kendall Belles

CS-2413 Data Structures

Project 4

May 4, 2013

Contents

[Design Documents: 3](#_Toc355466517)

[Class Layout: 3](#_Toc355466518)

[Flow Charts: 4](#_Toc355466519)

[ReadMe File: 15](#_Toc355466520)

[Source Code: 16](#_Toc355466521)

[sorter.h 16](#_Toc355466522)

[sorter.cpp 16](#_Toc355466523)

[sortingAlgorithms.h 18](#_Toc355466524)

[sortingAlgorithms.cpp 19](#_Toc355466525)

[Compile: 25](#_Toc355466526)

[Output: 25](#_Toc355466527)

# Design Documents:

These are the initial design documents for the sorting module and driver. Some changes may have been made during implementation that are not reflected in these documents. Please refer to full code for current version.

## Class Layout:

|  |  |  |
| --- | --- | --- |
| class Sorter | |  |
| attributes | |  |
| int | size | number of elements in array |
| int | range | largest number allowed in array |
| int | digits | number of digits in largest allowable number |
| int\* | anArray | pointer to array |
| operations | |  |
|  | Sorter() | basic constructor |
| void | start() | conducts all other operations |
|  | ~Sorter | basic destructor |

|  |  |  |
| --- | --- | --- |
| class SortAlgorithm | |  |
| attributes | |  |
| int\* | myArray | pointer to array |
| int | swaps | number of write comands exectuded |
| int | compares | number of write comparisons exectuded |
| private operations | |  |
| void | medianOf3(int, int) | helper function for quickSort(int, int) |
| int | partionArray(int, int) | helper function for quickSort(int, int) |
| void | swap(int, int) | helper function for quickSort(int, int) |
| void | merge(int, int, int) | helper function for mergeSort(int, int) |
| public operations | |  |
|  | SortAlgorithm(int[], int) | constructor to initialize array, swaps, and compares |
| void | printArray() | print first 15 elements of array |
| void | printReport() | print compares, swaps |
| void | bubbleSort() | bubble sorts myArray |
| void | insertionSort() | insertion sorts myArray |
| void | selectionSort() | selection sorts myArray |
| void | radixSort(int) | radix sorts myArray |
| void | bucketSort(int, int) | bucket sorts myArray |
| void | mergeSort(int, int) | merge sorts myArray |
| void | quickSort(int, int) | quick sorts myArray |

## Flow Charts:



## 



          

# ReadMe File:

=== sorter.h ===

The sorting module consists of four files, sorter.h, sorter.cpp, sortingAlgorithms.h, and sortingAlgorithms.cpp. The sorting module creates an array of random values, sorts the array, and reports on the efficiency of the sorting algorithms.

=== Description ===

The sorting module takes an requests two integer values from the user, the data size and data range. The data size controls how many elements will be in the array, the data range controls how large the numbers generated can be. On initialization of the sorter object, the random array is generated. When the function start() is called, each sorting function is called and a report is generated.

=== Use ===

Extract the files to the desired directory and add #include “sorter.h” to the project file. Compile normally.

To implement the sorter object, create a sorter object. This will prompt the user for the needed input. To generate the report, call start() on the sorter object. The sorter class has a deconstructor that will free associated memory.

=== Example ===

Sorter mySort;

mySort.start();

mySort.~Sorter();

=== Author and Contact ===

Primary author: Ken Belles, kendall.belles@cameron.edu

Contributions by: Dr. Chao Zhao, chaoz@cameron.edu

# Source Code:

## sorter.h

#ifndef SORTER\_H

#define SORTER\_H

class Sorter {

private:

int size;

int range;

int digits;

int \*anArray;

public:

Sorter();

void start();

~Sorter();

};

#include "sorter.cpp"

#endif

## sorter.cpp

#include <iostream>

#include <iomanip>

#include <cmath>

#include <time.h>

#include "sortingAlgorithms.h"

using namespace std;

Sorter::Sorter() {

cout << "Please enter data size: ";

cin >> size;

cout << "Please enter maximum number: ";

cin >> range;

digits = 1;

for (int i=1; i<10; i++) {

// if the current power of ten is <= value

if (int(pow(10.0, i))<=range)

digits++;

else

break;

}

//create anArray of proper size

anArray = new int[size];

//set each element in anArray to random number [1, range]

srand((unsigned int)time(NULL));

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

anArray[i] = rand()%range + 1;

}

void Sorter::start() {

SortAlgorithm a(anArray, size);

SortAlgorithm b(anArray, size);

SortAlgorithm c(anArray, size);

SortAlgorithm d(anArray, size);

SortAlgorithm e(anArray, size);

SortAlgorithm f(anArray, size);

SortAlgorithm g(anArray, size);

cout << endl;

cout << "Sorted List:" << endl;

cout << endl;

cout << "Selection Sort: ";

a.selectionSort();

a.printArray();

cout << "Bubble Sort: ";

b.bubbleSort();

b.printArray();

cout << "Insertion Sort: ";

c.insertionSort();

c.printArray();

cout << "Merge Sort: ";

d.mergeSort();

d.printArray();

cout << "Quick Sort: ";

e.quickSort();

e.printArray();

cout << "Radix Sort: ";

f.radixSort(digits);

f.printArray();

cout << "Bucket Sort: ";

g.bucketSort(1,range);

g.printArray();

cout << endl;

cout << "Sorting Results:" << endl;

cout << endl;

cout << "Algorithm " << setw(20) << "Comparisons" << setw(20) << "Data Moves" << endl;

cout << "Select ";

a.printReport();

cout << "Bubble ";

b.printReport();

cout << "Insert ";

c.printReport();

cout << "Merge ";

d.printReport();

cout << "Quick ";

e.printReport();

cout << "Radix ";

f.printReport();

cout << "Bucket ";

g.printReport();

cout << endl;

}

Sorter::~Sorter() {

delete [] anArray;

}

## sortingAlgorithms.h

#ifndef SORTINGALG\_H

#define SORTINGALG\_H

int MAX = 100;

class SortAlgorithm {

private:

int \*myArray;

int swaps;

int compares;

//private helper functions for quickSort(int, int)

void medianOf3(int, int);

int partitionArray(int, int);

void swap(int, int);

//private helper function for mergeSort(int, int)

void merge(int, int, int);

public:

SortAlgorithm(int[], int);

void printArray();

void printReport();

void bubbleSort();

void insertionSort();

void selectionSort();

void radixSort(int);

void bucketSort(int, int);

void mergeSort(int, int);

void quickSort(int, int);

};

#include "sortingAlgorithms.cpp"

#endif

## sortingAlgorithms.cpp

#include <iostream>

#include <iomanip>

#include <queue>

using namespace std;

SortAlgorithm::SortAlgorithm(int anArray[], int size) {

MAX = size;

myArray = new int[MAX];

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

myArray[i] = anArray[i];

swaps = 0;

compares = 0;

}

void SortAlgorithm::printArray() {

for (int i=0; i<15; i++)

cout << " " << myArray[i];

cout << "..." << endl;

}

void SortAlgorithm::printReport() {

cout << setw(20) << compares << setw(20) << swaps << endl;

}

void SortAlgorithm::bubbleSort() {

bool sorted = false;

int pass = 1;

//while not sorted and #passes < #elements

while (!sorted && (pass < MAX)) {

sorted = true;

//for each unsorted element

for (int i=0; i<MAX-pass; i++) {

compares++;

//if out of ourder, swap, still not sorted

if (myArray[i] > myArray[i+1]) {

swap(i, i+1);

sorted = false;

}

}

pass++;

}

}

void SortAlgorithm::insertionSort() {

//for each element, start from 2nd

for (int i=1; i<MAX; i++) {

int nextNum = myArray[i];

int loc = i;

//while loc still in array and nextNum < element, shift element

compares++;

while (loc>0 && myArray[loc-1]>nextNum) {

compares++;

swaps++;

myArray[loc] = myArray[loc-1];

loc--;

}

swaps++;

//move element at loc to correct position

myArray[loc] = nextNum;

}

}

void SortAlgorithm::selectionSort() {

int j = 0;

//for each element

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

int smallest = i;

//for all unsorted element, find smallest

for (j=i; j<MAX; j++) {

compares++;

if (myArray[smallest] > myArray[j])

smallest = j;

}

//swap smallest unsorted with first unsorted

swap(smallest, i);

}

}

void SortAlgorithm::radixSort(int digits = 3) {

queue<int> group[10];

int mod = 10;

int denom = 1;

//for each digit

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

//for every element of the array, isolate the desired digit

//and push the element onto the correct queue

for (int j=0; j<MAX; j++) {

group[(myArray[j]%mod)/denom].push(myArray[j]);

swaps++;

}

int index = 0;

//for each queue, push contents into array

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

while (!group[j].empty()) {

myArray[index] = group[j].front();

group[j].pop();

index++;

swaps++;

}

}

//increase modulus and denominator by a factor of 10

mod \*= 10;

denom \*= 10;

}

}

void SortAlgorithm::bucketSort(const int minNum, const int maxNum) {

//initialize tally array

int tMAX = maxNum - minNum + 1;

int \*tally;

tally = new int[tMAX];

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

tally[i] = 0;

}

//populate tally with myArray

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

tally[myArray[i]-minNum]++;

}

//populate myArray with tally

int index = 0;

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

while(tally[i] > 0) {

swaps++;

myArray[index] = i + minNum;

tally[i]--;

index++;

}

}

delete [] tally;

}

void SortAlgorithm::mergeSort(int first=0, int last=MAX-1) {

if (first < last) {

int mid = first + (last - first) / 2;

mergeSort(first, mid);

mergeSort(mid+1, last);

merge(first, mid, last);

}

}

void SortAlgorithm::merge(int first, int mid, int last) {

//create tempArray for sorting

int \*tempArray;

tempArray = new int[MAX];

//initializations

int sub1Start = first;

int sub1End = mid;

int sub2Start = mid + 1;

int sub2End = last;

int index = sub1Start;

//while niether subArray empty, write smallest element to tempArray, update

while (sub1Start <= sub1End && sub2Start <= sub2End) {

compares++;

if (myArray[sub1Start] < myArray[sub2Start]) {

tempArray[index] = myArray[sub1Start];

sub1Start++;

swaps++;

}

else {

tempArray[index] = myArray[sub2Start];

sub2Start++;

swaps++;

}

index++;

}

//while subArray1 not empty, write contents to tempArray, update

while (sub1Start <= sub1End) {

tempArray[index] = myArray[sub1Start];

sub1Start++;

index++;

swaps++;

}

//while subArray2 not empty, write contents to tempArray, update

while (sub2Start <= sub2End) {

tempArray[index] = myArray[sub2Start];

sub2Start++;

index++;

swaps++;

}

//write contents of tempArray to myArray

for(index=first; index<=last; index++) {

myArray[index] = tempArray[index];

swaps++;

}

delete [] tempArray;

}

void SortAlgorithm::quickSort (int front=0, int back=MAX-1) {

//if array larger than 1 element

if (front < back) {

//if array has at least 4 elements, select good pivot

if (back-front > 2)

medianOf3(front, back);

//partion array into S1 and S2

int partitionIndex = partitionArray (front, back);

//sort S1

quickSort(front, partitionIndex-1);

//sort S2

quickSort(partitionIndex+1, back);

}

}

void SortAlgorithm::medianOf3 (int front, int back) {

//select mid point

int mid = front + (back - front) / 2;

compares++;

//sort front, back, and mid in increasing order

if (myArray[front] > myArray[back])

swap(front, back);

compares++;

if (myArray[mid] > myArray[back])

swap(mid, back);

compares++;

if (myArray[front] > myArray[mid])

swap(front, mid);

compares++;

if (myArray[mid] > myArray[back])

swap(mid, back);

//move mid, a good pivot, to the pivot position

swap(back, mid);

}

int SortAlgorithm::partitionArray(int front, int back) {

//last S1

int indexFromLeft = front;

//first S2

int indexFromRight = back - 1;

int pivot = myArray[back];

bool done = false;

while (!done) {

//while indexFromLeft in array & array element member of S1

//update indexFromLeft

compares++;

while (indexFromLeft < back && myArray[indexFromLeft] < pivot) {

indexFromLeft++;

compares++;

}

//while indexFromRight in array & array element member of S2

//update indexFromRight

compares++;

while (indexFromRight > front && myArray[indexFromRight] > pivot) {

indexFromRight--;

compares++;

}

//if last S1 left of first S2, swap and update

if (indexFromLeft < indexFromRight) {

swap(indexFromLeft, indexFromRight);

indexFromLeft++;

indexFromRight--;

}

else

done = true;

}

//avoid unnecessary swap

if (indexFromLeft != back)

swap(indexFromLeft, back);

return indexFromLeft;

}

void SortAlgorithm::swap(int first, int second) {

swaps += 3;

int temp = myArray[first];

myArray[first] = myArray[second];

myArray[second] = temp;

}

# Compile:

C:\Users\Ken\Documents\Data Struct\Project4>cl project4.cpp /EHsc

Microsoft (R) C/C++ Optimizing Compiler Version 15.00.30729.01 for x64

Copyright (C) Microsoft Corporation. All rights reserved.

project4.cpp

Microsoft (R) Incremental Linker Version 9.00.30729.01

Copyright (C) Microsoft Corporation. All rights reserved.

/out:project4.exe

project4.obj

C:\Users\Ken\Documents\Data Struct\Project4>

# Output:

C:\Users\Ken\Documents\Data Struct\Project4>project4.exe

Please enter data size: 100

Please enter maximum number: 10

Sorted List:

Selection Sort: 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2...

Bubble Sort: 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2...

Insertion Sort: 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2...

Merge Sort: 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2...

Quick Sort: 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2...

Radix Sort: 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2...

Bucket Sort: 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2...

Sorting Results:

Algorithm Comparisons Data Moves

Select 5049 297

Bubble 4697 6552

Insert 2283 2283

Merge 540 1344

Quick 743 744

Radix 0 400

Bucket 0 100

Press any key to continue . . .

C:\Users\Ken\Documents\Data Struct\Project4>