CS 302 Project 4 - Patrick Austin

Generated by Doxygen 1.8.6

Mon Mar 9 2015 22:43:27

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 bubbleSort Class Reference

`#include <sorts.h>`

Public Member Functions

- bubbleSort ()
- void sort (const int &)
- void bubbleUp (const int &, const int &, bool &)
- void readin (const int &)
- double getTime () const
- int getComparisons () const
- int getSwaps () const
- void resetCounts ()

3.1.1 Constructor & Destructor Documentation

3.1.1.1 bubbleSort::bubbleSort ( )

Bubble sort class constructor. Initializes count and time analysis values to 0.

Precondition

None.

Postcondition

Object created with time, comparisons, swaps = 0

3.1.2 Member Function Documentation

3.1.2.1 void bubbleSort::bubbleUp (const int & *startIndex,* const int & *endIndex,* bool & *sorted*)

bubbleSort bubbleUp function. Called by the sort function to loop through the array, conducting comparisons and swaps as needed. Tracks comparisons and swaps, as specified in the prompt. !!! ADAPTED FROM POWERPOINT SLIDE CODE !!!

Precondition

Comparable data between startIndex and endIndex.

Postcondition

One 'bubble up' completed, if possible.

Parameters

startIndex	The first value to be compared.
endIndex	The last value to be compared. sorted Bool used in internal bubble sort operations to help the sort terminate early if possible.

3.1.2.2 int bubbleSort::getComparisons ( ) const

Bubble sort getComparisons function. Returns comparisons data member, tracking number of comparisons carried out during sorting.

Precondition

None.

Postcondition

Comparisons returned; value unchanged.

3.1.2.3 int bubbleSort::getSwaps ( ) const

Bubble sort getSwaps function. Returns swaps data member, tracking number of swaps carried out during sorting.

Precondition

None.

Postcondition

Swaps returned; value unchanged.

3.1.2.4 double bubbleSort::getTime ( ) const

Bubble sort getTime function. Returns time data member, used to track time spent sorting.

Precondition

None.

Postcondition

Time returned; value unchanged.

3.1.2.5 void bubbleSort::readin (const int & *size*)

Bubble sort readin function. Reads in random data of specified size to sort.

Precondition

Extant and correctly formatted text files 1000.txt, 10000.txt, 100000.txt, 1000000.txt, size parameter = 1000, 10000, 100000, 1000000

Postcondition

Bubblesort array filled with read in values from 0 to size

Parameters

size	The size of the data to be read in. Only sizes of 1000, 10000, 100000, and 1000000 are supported in this implementation!

3.1.2.6 void bubbleSort::resetCounts ( )

Bubble sort resetCounts function. Resets time, comparisons, and swaps values to 0 if user wishes to reset the tracking of the sorting algorithm that has happened so far.

Precondition

None.

Postcondition

Time, comparisons, swaps set to 0.

3.1.2.7 void bubbleSort::sort (const int & *numValues*)

Sorts the items in an array into ascending order using bubble sort. Tracks time of operation, number of swaps, number of comparisons. Expect extremely long runtime with 1000000 values. Time operation coded for Linux. !!! ADAPTED FROM POWERPOINT SLIDE CODE !!!

Precondition

None.

Postcondition

theArray is sorted into ascending order; n is unchanged.

Parameters

n	The size of theArray.

The documentation for this class was generated from the following files:

- sorts.h
- bubble.cpp

3.2 countingSort Class Reference

`#include <sorts.h>`

Public Member Functions

- countingSort ()
- void sort (const int &)
- void readin (const int &)
- double getTime () const
- int getComparisons () const
- int getSwaps () const
- void resetCounts ()

3.2.1 Constructor & Destructor Documentation

3.2.1.1 countingSort::countingSort ( )

Counting sort class constructor. Initializes count and time analysis values to 0.

Precondition

None.

Postcondition

Object created with time, comparisons, swaps = 0

3.2.2 Member Function Documentation

3.2.2.1 int countingSort::getComparisons ( ) const

Counting sort getComparisons function. Returns comparisons data member, tracking number of comparisons carried out during sorting. Always returns 0, counting sort does no comparisons.

Precondition

None.

Postcondition

Comparisons returned; value unchanged.

3.2.2.2 int countingSort::getSwaps ( ) const

Bubble sort getSwaps function. Returns swaps data member, tracking number of swaps carried out during sorting. Always returns 0, counting sort does not sort by swapping values per se.

Precondition

None.

Postcondition

Swaps returned; value unchanged.

3.2.2.3 double countingSort::getTime ( ) const

Counting sort getTime function. Returns time data member, used to track time spent sorting.

Precondition

None.

Postcondition

Time returned; value unchanged.

3.2.2.4 void countingSort::readin (const int & *size*)

Counting sort readin function. Reads in random data of specified size to sort.

Precondition

Extant and correctly formatted text files 1000.txt, 10000.txt, 100000.txt, 1000000.txt, size paramter = 1000, 10000, 100000, 1000000

Postcondition

Countingsort array filled with read in values from 0 to size

Parameters

size	The size of the data to be read in. Only sizes of 1000, 10000, 100000, and 1000000 are supported in this implementation!

3.2.2.5 void countingSort::resetCounts ( )

Counting sort resetCounts function. Resets time, comparisons, and swaps values to 0 if user wishes to reset the tracking of the sorting algorithm that has happened so far.

Precondition

None.

Postcondition

Time, comparisons, swaps set to 0.

3.2.2.6 void countingSort::sort (const int & *size*)

Counting sort sort function. Sorts values in arr data member from 0 to size into ascending order using counting sort. Tracks time of operation, number of comparisons, number of swaps. Time operation coded in Linux. !!! ADAPTED FROM POWERPOINT SLIDE CODE!!!

Precondition

size $>$ 0

Postcondition

Values sorted from 0 to size

Parameters

size	The size of the array to be sorted.

The documentation for this class was generated from the following files:

- sorts.h
- counting.cpp

3.3 mergeSort Class Reference

`#include <sorts.h>`

Public Member Functions

- mergeSort ()
- void timeSort (const int &, const int &)
- void sort (const int &, const int &)
- void merge (const int &, const int &, const int &)
- void readin (const int &)
- double getTime () const
- int getComparisons () const
- int getSwaps () const
- void resetCounts ()

3.3.1 Constructor & Destructor Documentation

3.3.1.1 mergeSort::mergeSort ( )

Merge sort class constructor. Initializes count and time analysis values to 0.

Precondition

None.

Postcondition

Object created with time, comparisons, swaps = 0

3.3.2 Member Function Documentation

3.3.2.1 int mergeSort::getComparisons ( ) const

Merge sort getComparisons function. Returns comparisons data member, tracking number of comparisons carried out during sorting.

Precondition

None.

Postcondition

Comparisons returned; value unchanged.

3.3.2.2 int mergeSort::getSwaps ( ) const

Merge sort getSwaps function. Returns swaps data member, tracking number of swaps carried out during sorting.

Precondition

None.

Postcondition

Swaps returned; value unchanged.

3.3.2.3 double mergeSort::getTime ( ) const

Merge sort getTime function. Returns time data member, used to track time spent sorting.

Precondition

None.

Postcondition

Time returned; value unchanged.

3.3.2.4 void mergeSort::merge (const int & *first,* const int & *mid,* const int & *last*)

Merges two sorted array segments theArray[first..mid] and theArray[mid+1..last] into one sorted array. Tracks time of operation, number of swaps, number of comparisons. There seems to be a bug in this method when run with 1000000 values where a small number of values are erroneously swapped when sorting an already sorted array. I was not able to ascertain why this happens, but even if I could I am not allowed to modify the sorting algorithm's code for this project. For 1000, 10000, and 100000 values the algorithm appears to work correctly in the specified circumstances. In order to prevent running out of memory on my system when analyzing 1 000 000 values, temp-Array had to be changed from a static array to a dynamic one. This does not strike me as an unreasonable adaptation, but if it strikes you as unreasonable then please show mercy. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

first <= mid <= last. The subarrays theArray[first..mid] and theArray[mid+1..last] are each sorted in increasing order.

Postcondition

arr[first..last] is sorted.

Parameters

first	The index of the beginning of the first segment in theArray.
mid	The index of the end of the first segment in theArray; mid + 1 marks the beginning of the second segment.

last	The index of the last element in the second segment in theArray.

Note

This function merges the two subarrays into a temporary array and copies the result into the original array theArray.

3.3.2.5 void mergeSort::readin (const int & *size*)

Merge sort readin function. Reads in random data of specified size to sort.

Precondition

Extant and correctly formatted text files 1000.txt, 10000.txt, 100000.txt, 1000000.txt, size paramter = 1000, 10000, 100000, 1000000

Postcondition

Mergesort array filled with read in values from 0 to size

Parameters

size	The size of the data to be read in. Only sizes of 1000, 10000, 100000, and 1000000 are supported in this implementation!

3.3.2.6 void mergeSort::resetCounts ( )

Merge sort resetCounts function. Resets time, comparisons, and swaps values to 0 if user wishes to reset the tracking of the sorting algorithm that has happened so far.

Precondition

None.

Postcondition

Time, comparisons, swaps set to 0.

3.3.2.7 void mergeSort::sort (const int & *first,* const int & *last*)

Recursively merge sorts arr data member of mergeSort class, from first element to last. Tracks time of operation, number of swaps, number of comparisons. !!! ADAPTED FROM POWERPOINT SLIDE CODE !!!

Precondition

first <= last

Postcondition

Array values[first..last] sorted into ascending order.

Parameters

first	The index of the first element to be sorted.
last	The index of the last element to be sorted.

3.3.2.8 void mergeSort::timeSort (const int & *first,* const int & *last*)

Merge sort timeSort function. Sorts the values in the arr data member of the class by calling the sort function. Tracks the time the sort operation takes. Timing operation coded for Linux.

Precondition

first <= last

Postcondition

Array values [first..last] sorted into ascending order, time of operation stored in time double. data member.

Parameters

first	The index of the first element to be sorted.
last	The index of the last element to be sorted.

The documentation for this class was generated from the following files:

- sorts.h
- merge.cpp

Chapter 4

File Documentation

4.1 bubble.cpp File Reference

```
#include <algorithm>
#include <ctime>
#include <fstream>
#include "sorts.h"
```

4.1.1 Detailed Description

Author

Patrick Austin

Date

3/9/2015

4.2 counting.cpp File Reference

```
#include <algorithm>
#include <ctime>
#include <fstream>
#include "sorts.h"
```

4.2.1 Detailed Description

Author

Patrick Austin

Date

3/9/2015

4.3 main.cpp File Reference

```
#include <iostream>
#include "sorts.h"
```

Functions

- void analyzeBubble (const int &size)
- void analyzeMerge (const int &size)
- void analyzeCounting (const int &size)
- int main ()

4.3.1 Detailed Description

Author

Patrick Austin

Date

3/9/2015

4.3.2 Function Documentation

4.3.2.1 void analyzeBubble (const int & *size*)

Conducts sorting operation for bubble sort - 10 iterations for values $<$ 1 000 000, 1 iteration for values $>$ 1 000 000. Conducts readin, sorts, and displays info to console. For reasons I was unable to resolve before submission, swaps and comparisons for the 100000 value analysis will return negative values due to integer overload. Short of including a third party BigInt library I was unable to resolve this issue, which seems to be partly OS/compiler based. My apologies; mercy please.

Precondition

Text files 1000.txt, 10000.txt, 100000.txt, 1000000.txt must exist and be formatted as specified.

Postcondition

Sort info displayed to console.

Parameters

size	The size of the data to be sorted - only 1000, 10000, 100000, 1000000 supported.

4.3.2.2 void analyzeCounting (const int & *size*)

Conducts sorting operation for counting sort - 10 iterations for all values. Conducts readin, sorts, and displays info to console.

Precondition

Text files 1000.txt, 10000.txt, 100000.txt, 1000000.txt must exist and be formatted as specified.

Postcondition

Sort info displayed to console.

Parameters

size	The size of the data to be sorted - only 1000, 10000, 100000, 1000000 supported.

4.3.2.3 void analyzeMerge (const int & *size*)

Conducts sorting operation for merge sort - 10 iterations for all values. Conducts readin, sorts, and displays info to console.

Precondition

Text files 1000.txt, 10000.txt, 100000.txt, 1000000.txt must exist and be formatted as specified.

Postcondition

Sort info displayed to console.

Parameters

size	The size of the data to be sorted - only 1000, 10000, 100000, 1000000 supported.

4.3.2.4 int main ( )

4.4 merge.cpp File Reference

```
#include <algorithm>
#include <ctime>
#include <fstream>
#include "sorts.h"
```

4.4.1 Detailed Description

Author

Patrick Austin

Date

3/9/2015

4.5 sorts.h File Reference

Classes

- class bubbleSort
- class mergeSort
- class countingSort

Variables

- const int MAX_VALUES = 1000000
- const int MAX_SIZE = 500000

4.5.1 Detailed Description

Author

Patrick Austin

Date

3/9/2015

4.5.2 Variable Documentation

4.5.2.1 const int MAX_SIZE = 500000

4.5.2.2 const int MAX_VALUES = 1000000

Index