PA04 - Sorting

Generated by Doxygen 1.8.11

Contents

1 Class Index

1.1 Class List

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

mySort	**2**

2 File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Algorithm.h	**??**
main.cpp **Put the three algorithms to use**	**5**

3 Class Documentation

3.1 mySort Class Reference

Public Member Functions

- mySort (int)
- ∼mySort ()
- int getSize ()
- void **display** ()
- void readIn ()
- void readOut ()
- void RNG ()
- void insertSort ()
- void merge (int, int, int)
- void mergeSort (int, int)
- void bucketSort ()
- void prepData ()
- void getSortInfo (unsigned long long &, unsigned long long &)

Private Attributes

- int ∗ **myData**
- int **size**
- unsigned long long **numComps**
- unsigned long long **numSwaps**

3.1.1 Constructor & Destructor Documentation

3.1.1.1 mySort::mySort (int *howBig*)

Constructor for the class that stores and sorts data

Postcondition

An ADT is created, but empty

Parameters

howBig	The amount of data to be considered

3.1.1.2 mySort::∼mySort ( )

Deconstructor for the mySort class

Precondition

ADT is going out of scope

Postcondition

dynamically allocated array is deleted

3.1.2 Member Function Documentation

3.1.2.1 void mySort::bucketSort ( )

Sorts items in ascending order

Precondition

myData is an array of unsorted numbers

Postcondition

myData is sorted in ascending order

Note

This function will create 10 buckets, and arrange the array so the numbers that are x/10 of the max size (x being the bucket #, starting with 1) are placed together, and that segment of the array is sorted using merge sort

This only really works for data sets that are sizes of multiples of ten (anthing else might cut off a number)

3.1.2.2 int mySort::getSize ( )

Find the amount of data the ADT stores

Returns

the integer size

3.1.2.3 void mySort::getSortInfo (unsigned long long & *swaps,* unsigned long long & *comps*)

Gets the number of compares and swaps

Parameters

swaps	unsigned long long to hold the number of swaps
comps	unsigned long long to hold the number of compares

3.1.2.4 void mySort::insertSort ( )

Sorts the data using insertion sort algorithm

Precondition

Numbers in the myData array are unsorted

Postcondition

Numbers in the myData array are sorted

3.1.2.5 void mySort::merge (int *first,* int *mid,* int *last*)

Do the actual sorting in merge sort algorithm

Precondition

first <= mid <= last. The arrays are sorted in increasing order

Postcondition

theArray[first, last] is sorted

Parameters

first	index of beginning of segment in the array
mid	index of the end of the first segment in the array
last	index of last element in second array

Note

This function merges two subarrays into a temporary array

3.1.2.6 void mySort::mergeSort (int *first,* int *last*)

Sorts items in ascending order

Precondition

myData is an array of unsorted numbers

Postcondition

myData is sorted in ascending order

Parameters

first	first element to consider sorting
last	last element to consider sorting

3.1.2.7 void mySort::prepData ( )

Resets the counters for compares and swaps

Postcondition

the counters for compares and swaps is reset

3.1.2.8 void mySort::readIn ( )

Read in a list of numbers from the file "data"

Postcondition

Data is read into the myData array

3.1.2.9 void mySort::readOut ( )

Outputs the data in the myData array to file "data"

Postcondition

file "data" contains all data from the myData array

3.1.2.10 void mySort::RNG ( )

Generates a list of random numbers

Postcondition

The myData array and "data" file contain a list of the same randomly generated numbers between 0 and 1,000,000

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

- Algorithm.h
- Algorithm.cpp

4 File Documentation

4.1 main.cpp File Reference

Put the three algorithms to use.

```
#include "Algorithm.h"
```
Include dependency graph for main.cpp:

Index