Programs Illustrating the C/C++ Style Guide

Generated by Doxygen 1.9.6

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 sorts Struct Reference

Data Fields

- char ∗ name
- void(∗ sortProc)(int[ ], int)

3.1.1 Detailed Description

structure to identify both the name of a sorting algorithm and ∗ a pointer to the function that performs the sort ∗ the main function utilizes this struct to define an array of ∗ the sorting algorithms to be timed by this program. ∗

3.1.2 Field Documentation

3.1.2.1 name

```
char* name
```

the name of a sorting algorithm as text

3.1.2.2 sortProc

```
void(* sortProc) (int[], int)
```

the procedure name of a sorting function

The documentation for this struct was generated from the following file:

- sort-comparisons.c

Chapter 4

File Documentation

4.1 sort-comparisons.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

Data Structures

- struct sorts

Macros

- #define numAlgs 5

Typedefs

- typedef struct sorts sorts

Functions

- void selectionSort (int a[ ], int n)
- void insertionSort (int a[ ], int n)
- int impPartition (int a[ ], int size, int left, int right)
- void hybridQuicksortHelper (int a[ ], int size, int left, int right)
- void hybridQuicksort (int a[ ], int n)
- void merge (int aInit[ ], int aRes[ ], int aInitLength, int start1, int start2, int end2)
- void mergeSort (int initArr[ ], int n)
- void percDown (int array[ ], int hole, int size)
- void heapSort (int a[ ], int n)
- char $*$ checkAscValues (int a[ ], int n)
- char $*$ checkAscending (int a[ ], int n)
- int main ()

4.1.1 Detailed Description

Remarks

program times several sorting algorithms on data sets of various sizes ∗

-

this version includes code for straight selection insertion sorts ∗ stubbs are provided for other sorting algoritms, including ∗ hybrid quicksort, merge sort and heap sort ∗

-

Author

Henry M. Walker ∗

Remarks

Assignment Comparison of Sorting Algorithms ∗

-

Date

August 15, 2022 ∗

-

Remarks

References ∗

Dynamic Programming: Anany Levitin, "The Design and ∗ and Analysis of Algorithms", Second Edition, ∗ Sections 3.1 (Selectino Sort), 4.1 (Insertion Sort), ∗ 5.1 (Mergesort), 5.2 (Quicksort), 6.4 (Heapsort) ∗

-

People participating with Problem/Progra Discussions: ∗ Marcia Watts ∗

-

4.1.2 Macro Definition Documentation

4.1.2.1 numAlgs

```
#define numAlgs 5
```

4.1.3 Typedef Documentation

4.1.3.1 sorts

```
typedef struct sorts sorts
```

structure to identify both the name of a sorting algorithm and ∗ a pointer to the function that performs the sort ∗ the main function utilizes this struct to define an array of ∗ the sorting algorithms to be timed by this program. ∗

4.1.4 Function Documentation

4.1.4.1 checkAscending()

```
char * checkAscending (
              int a[ ],
              int n )
```

check all array elements are in non-descending order ∗

Parameters

a	the array to be sorted *
n	the size of the array * returns "ok" if array elements in non-descending order; "NO" otherwise *

4.1.4.2 checkAscValues()

```
char * checkAscValues (
             int a[],
             int n )
```

check all array elements have values 0, 2, 4, . . ., 2(n-1) *

Parameters

a	the array to be sorted *
n	the size of the array * returns "ok" if array contains required elements; "NO" if not *

4.1.4.3 heapSort()

```
void heapSort (
             int a[],
             int n )
```

heap sort, main function *

Parameters

a	the array to be sorted *
n	the size of the array *

Postcondition

the first n elements of a are sorted in non-descending order *

4.1.4.4 hybridQuicksort()

```
void hybridQuicksort (
             int a[],
             int n )
```

hybrid quicksort, main function * algoithmic elements * random pivot used in partition function * insertion used for small array segments *

Parameters

a	the array to be sorted *
n	the size of the array *

Postcondition

the first n elements of a are sorted in non-descending order *

4.1.4.5 hybridQuicksortHelper()

```
void hybridQuicksortHelper (
             int a[],
             int size,
             int left,
             int right )
```

Quicksort helper function ∗ algoithmic elements ∗ quicksort used when array segments > variable breakQuicksortToInsertion ∗ insertion sort used for small array segments ∗

Parameters

a	the array to be processed ∗
size	the size of the array ∗
left	the lower index for items to be processed ∗
right	the upper index for items to be processed ∗

Postcondition

sorts elements of a between left and right ∗

4.1.4.6 impPartition()

```
int impPartition (
             int a[],
             int size,
             int left,
             int right )
```

Improved Partition function ∗ uses a[left] as pivot value in processing ∗ algoithmic elements ∗ random pivot utilized ∗ swaps only when required by finding misplaced large and small elements ∗

Parameters

a	the array to be processed ∗
size	the size of the array ∗
left	the lower index for items to be processed ∗
right	the upper index for items to be processed ∗

Postcondition

elements of a are rearranged, so that ∗ items between left and index mid are <= a[mid] ∗ items between dex mid and right are >= a[mid] ∗

Returns

mid ∗

4.1.4.7 insertionSort()

```
void insertionSort (
             int a[],
             int n )
```

insertion sort ∗

Parameters

a	the array to be sorted ∗
n	the size of the array ∗

Postcondition

the first n elements of a are sorted in non-descending order ∗

4.1.4.8 main()

```
int main ( )
```

driver program for testing and timing sorting algorithms ∗

4.1.4.9 merge()

```
void merge (
             int aInit[],
             int aRes[],
             int aInitLength,
             int start1,
             int start2,
             int end2 )
```

merge sort helper function ∗

Parameters

aInit	source array for merging ∗
aRes	target array for merging ∗
aInitLength	the size of the array segment to be merged ∗
start1	the first index of the first array segment to be merged ∗
start2	the first index of the second array segment to be merged ∗
end2	the last index of the second array segment to be merged ∗

Postcondition

elements aInit[start1]..aInit[start1+mergeSize] merged with ∗ aInit[start2]..Init[end2] ∗ with the result placed in aRes ∗ Note: it may be that start2 >= aInit.length, in which case, only the ∗ valid part of aInit[start1] is copied ∗

4.1.4.10 mergeSort()

```
void mergeSort (
             int initArr[],
             int n )
```

merge sort helper function ∗

Parameters

initArr	the array to be sorted ∗
n	the size of the array ∗

Postcondition

the first n elements of a are sorted in non-descending order ∗

4.1.4.11 percDown()

```
void percDown (
             int array[],
             int hole,
             int size )
```

percDown function *

Parameters

array	the array to be made into a heap, starting at hold *
hole	base of subtree for start of processing *
size	the size of the array *

Precondition

all nodes in left and right subtrees of the hole node are heaps *

Postcondition

all nodes in the tree from the hole node downward form a hea *

4.1.4.12 selectionSort()

```
void selectionSort (
            int a[],
            int n )
```

straight selection sort *

Parameters

a	the array to be sorted *
n	the size of the array *

Postcondition

the first n elements of a are sorted in non-descending order *

Index