Comparing Sorting Algorithms

Generated by Doxygen 1.9.0
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Data Structure Index</td>
<td>1</td>
</tr>
<tr>
<td>2 File Index</td>
<td>3</td>
</tr>
<tr>
<td>3 Data Structure Documentation</td>
<td>5</td>
</tr>
<tr>
<td>4 File Documentation</td>
<td>7</td>
</tr>
</tbody>
</table>

Index 13
Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

sorts .................................................. 5
Chapter 2

File Index

2.1 File List

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

sort-comparisons.c ................................................................. 7
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

```c
#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 ()`
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 algorithms, including * quicksort and improved quicksort, * merge sort and heap sort *

·

Author

Henry M. Walker *

Remarks

Assignment Comparison of Sorting Algorithms *

·

Date

July 23, 2022 *

·

Remarks

References *


·

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

·

4.1.2 Typedef Documentation

4.1.2.1 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. *

4.1.3 Function Documentation

4.1.3.1 checkAscending()

char* checkAscending ( 
    int a[],
    int n)
check all array elements are in non-descending order *

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>the array to be sorted *</td>
</tr>
<tr>
<td>n</td>
<td>the size of the array * returns &quot;ok&quot; if array elements in non-descending order; &quot;NO&quot; otherwise *</td>
</tr>
</tbody>
</table>
4.1.3.2  checkAscValues()

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

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

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td>the array to be sorted *</td>
</tr>
<tr>
<td><code>n</code></td>
<td>the size of the array *</td>
</tr>
</tbody>
</table>

4.1.3.3  heapSort()

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

heap sort, main function *

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td>the array to be sorted *</td>
</tr>
<tr>
<td><code>n</code></td>
<td>the size of the array *</td>
</tr>
</tbody>
</table>

Postcondition

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

4.1.3.4  hybridQuicksort()

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

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

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td>the array to be sorted *</td>
</tr>
<tr>
<td><code>n</code></td>
<td>the size of the array *</td>
</tr>
</tbody>
</table>

Postcondition

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

4.1.3.5  hybridQuicksortHelper()

```c
void hybridQuicksortHelper ( int a[], int size, int
```

Generated by Doxygen
Quicksort helper function * algorithmic elements * quicksort used when array segments > variable breakQuicksort ← TolInsertion * insertion sort used for small array segments *

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td>the array to be processed</td>
</tr>
<tr>
<td><code>size</code></td>
<td>the size of the array</td>
</tr>
<tr>
<td><code>left</code></td>
<td>the lower index for items to be processed</td>
</tr>
<tr>
<td><code>right</code></td>
<td>the upper index for items to be processed</td>
</tr>
</tbody>
</table>

Postcondition

sorts elements of `a` between `left` and `right`

4.1.3.6 `impPartition()`

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

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

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td>the array to be processed</td>
</tr>
<tr>
<td><code>size</code></td>
<td>the size of the array</td>
</tr>
<tr>
<td><code>left</code></td>
<td>the lower index for items to be processed</td>
</tr>
<tr>
<td><code>right</code></td>
<td>the upper index for items to be processed</td>
</tr>
</tbody>
</table>

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`

4.1.3.7 `insertionSort()`

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

insertion sort *

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td>the array to be sorted</td>
</tr>
<tr>
<td><code>n</code></td>
<td>the size of the array</td>
</tr>
</tbody>
</table>
4.1.3.8 main()

int main ( )

driver program for testing and timing sorting algorithms

4.1.3.9 merge()

void merge ( )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aInit</td>
<td>source array for merging</td>
</tr>
<tr>
<td>aRes</td>
<td>target array for merging</td>
</tr>
<tr>
<td>aInitLength</td>
<td>the size of the array segment to be merged</td>
</tr>
<tr>
<td>start1</td>
<td>the first index of the first array segment to be merged</td>
</tr>
<tr>
<td>start2</td>
<td>the first index of the second array segment to be merged</td>
</tr>
<tr>
<td>end2</td>
<td>the last index of the second array segment to be merged</td>
</tr>
</tbody>
</table>

Postcondition

elements aInit[start1]..aInit[start1+mergeSize] merged with aInit[start2]..aInit[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

4.1.3.10 mergeSort()

void mergeSort ( )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>initArr</td>
<td>the array to be sorted</td>
</tr>
<tr>
<td>n</td>
<td>the size of the array</td>
</tr>
</tbody>
</table>

Postcondition

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

4.1.3.11 percDown()

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

**percDown function**

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>the array to be made into a heap, starting at hold *</td>
</tr>
<tr>
<td>hole</td>
<td>base of subtree for start of processing *</td>
</tr>
<tr>
<td>size</td>
<td>the size of the array *</td>
</tr>
</tbody>
</table>

**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 heap *

### 4.1.3.12 selectionSort()

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

**Straight selection sort +**

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>the array to be sorted *</td>
</tr>
<tr>
<td>n</td>
<td>the size of the array *</td>
</tr>
</tbody>
</table>

**Postcondition**

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

checkAscending
    sort-comparisons-6.c, 10
checkAscValues
    sort-comparisons-6.c, 10

heapSort
    sort-comparisons-6.c, 10

impPartition
    sort-comparisons-6.c, 11
impQuicksort
    sort-comparisons-6.c, 11
impQuicksorthelper
    sort-comparisons-6.c, 11
insertionSort
    sort-comparisons-6.c, 12

main
    sort-comparisons-6.c, 12
merge
    sort-comparisons-6.c, 12
mergeSort
    sort-comparisons-6.c, 13

name
    sorts, 7

partition
    sort-comparisons-6.c, 13
percDown
    sort-comparisons-6.c, 13

quicksort
    sort-comparisons-6.c, 14
quicksorthelper
    sort-comparisons-6.c, 14

selectionSort
    sort-comparisons-6.c, 15
sort-comparisons-6.c, 9
    checkAscending, 10
    checkAscValues, 10
    heapSort, 10
    impPartition, 11
    impQuicksort, 11
    impQuicksorthelper, 11
    insertionSort, 12
    main, 12
    merge, 12
    mergeSort, 13
    partition, 13

percDown, 13
quicksort, 14
quicksorthelper, 14
selectionSort, 15
sorts, 10
sortProc
    sorts, 7
sorts, 7
name, 7
sort-comparisons-6.c, 10
sortProc, 7