SortingLab

Lab7

Generated by Doxygen 1.8.6

Wed Oct 14 2015 12:31:22

Contents

1 Class Index

1.1 Class List

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

Sorts **1**

2 Class Documentation

2.1 Sorts Class Reference

Public Member Functions

- static< T > SList< T > take (SList< T > list, int pos)

Static Public Member Functions

- static< T > SList< T > append (SList< T > list1, SList< T > list2)
- static< T > SList< T > drop (SList< T > list, int pos)
- static< TextendsComparable< T >
  SList< T > mergeSort (SList< T > list)
- static< TextendsComparable< T >
  SList< T > qSort (SList< T > list)

2.1.1 Detailed Description

Common location to contain the quicksort and mergesort routines. Note, you will also create public methods that are required to support (implement) these sorts, such as "append," "merge," "take" and "drop." You may define other private methods as you deem fit.

Note that the purpose of this lab is for you to implement these two classic sorting routines over linked lists. These are recursive algorithms. The linked lists implementation has been provided for you and is available through the Jar file that is attached to your project. You should not need to use any iterative constructions here. Nor should you even consider using any of Java's Collection classes in place of the `SList` class that has been provided for you.

Important Considerations here!

Please read through these carefully:

- No iterative constructions should be used here.
- No exception handling will be permitted.
- Use the special "NULL" from the SList package in place of Java's `null`, or, better, return the original argument if it has been reduced to null.
- Note that no equals methods are defined for you. You should use, instead, the Comparable interface, which means that you should construct SLists of the correct types instead of relying upon equals methods for this lab. I advise reading how Strings compare!
- **DO NOT** import anything from java.util.∗ in this file.
- Do not mess with the imports at the top of this file, or in the StudentTests file.

Author

UMD CS Department.

2.1.2 Member Function Documentation

2.1.2.1 static <T> SList<T> append (SList< T > *list1*, SList< T > *list2*) `[static]`

append takes two lists and creates a new list containing the elements of the original lists in their original order. Append must handle empty lists as well. Empty lists, in the case of append, are "neutral elements" —similar to adding 0 to a sum.

Parameters

list1	
list2	

Returns

2.1.2.2 static <T> SList< T > drop (SList< T > *list*, int *pos*) `[static]`

Returns a new list that contains the remaining `pos` elements of `list` If the `list` has fewer elements than `pos` elements, an `IllegalStateException` if thrown.

For example: `drop( [1,2,3,4,5], 3)` gives `[4,5].`

Note: `drop( someList, 0 )` returns the original list.

Parameters

list	
pos	

Returns

2.1.2.3 static <TextendsComparable<T> SList<T> mergeSort (SList< T > *list*) [static]

Performs the classic "merge sort" by treating the empty list or the singleton list as already sorted and then recursively merging the results of reducing each sublist generated by a pivot.

Parameters

list	

Returns

2.1.2.4 static <TextendsComparable<T> SList<T> qSort (SList< T > *list*) [static]

Performs the classic quicksort —claimed to be the fastest in its class. Whereas mergesort is topological, quicksort is sensitive to the values in the collection to be sorted.

Parameters

list	

Returns

2.1.2.5 static< T > SList<T> take (SList< T > *list,* int *pos*)

Returns a new list that contains the first `pos` elements of `list`. If the `list` has fewer than `pos` elements, an `IllegalStateException` is thrown.

For example: `take( [1,2,3,4,5], 3 )` gives `[1,2,3]`

Note: `take( someList, 0 )` must return the empty list.

Parameters

list	
index	

Returns

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

- src/student_classes/Sorts.java

Index