HeapsLab

Generated by Doxygen 1.8.7

Mon Oct 26 2015 19:26:01

Contents

1 Class Index

1.1 Class List

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

student_classes.Heap< T extends Comparable< T > > **1**

2 Class Documentation

2.1 student_classes.Heap< T extends Comparable< T > > Class Reference

Public Member Functions

- Heap (Collection< Comparable< T >> elements)
- void sort ()
- int size ()
- List< T > asList ()
- String toString ()

Static Public Member Functions

- static< TextendsComparable< T >
  List< T > sort (List< T > lst)

2.1.1 Detailed Description

A heap is a complete binary tree that maintains the heap property, meaning that every *internal* node in the heap is greater than or equal to either of its children. (In the case of a "min heap," then the condition is less than or equal ....)

Unlike binary search trees, which you've implemented, no explicit ordering is required of the children; it's only the relationship between the children, i.e., the left and right, to their respective roots (internal nodes) that matters.

Now, we build heaps from the bottom up so as to ensure that the resulting tree is "complete," i.e., having no gaps. This is important because it allows us to traverse the nodes in left-right order and place them into a flat one-dimensional array. Once we've done this, then $2i+1$, where i is an in-range index gives the left child, and $2i+2$ gives the right. Oh, and $i/2-1$ gives the parent. (Recall that $i/2$ is integer division which is the floor of the quotient.)

As a proof of concept, we include the classic heapsort by calling `sort` on the inner heap. You should convince yourself that this algorithm is truly $\mathcal{O}(n\log(n))$.

Author

UMD CS Department.

Parameters

<T>	a `Comparable` object.

2.1.2 Constructor & Destructor Documentation

2.1.2.1 student_classes.Heap< T extends Comparable< T > >.Heap (Collection< Comparable< T >> *elements*)

Note: all of the work is done within this constructor. Given a Collection of Comparable objects, build a heap by calling the heapDown(index) on successive indices down to 0. Recall that index = elements.length/2 - 1, which is the parent node.

Parameters

elements	a `Collection` of `Comparable` objects.

2.1.3 Member Function Documentation

2.1.3.1 List<T> student_classes.Heap< T extends Comparable< T > >.asList ( )

Returns the contents of the heap as an object that implements List<T>, where T is a Comparable.

Returns

2.1.3.2 int student_classes.Heap< T extends Comparable< T > >.size ( )

Returns the size of the heap ... perhaps someone wants to know this.

Returns

`int` greater than or equal to 0.

2.1.3.3 void student_classes.Heap< T extends Comparable< T > >.sort ( )

Arguably, the "main event." A heap sort is a selection sort, which we've gone over in class, except that the heap is used to produce the next element in sort order.

Note: this method is rarely called on a heap that one intends to keep around because it modifies the heap by replacing it with a sorted collection of elements. Instead, it is usually called from within the `static sort` method, whose signature will interest you.

2.1.3.4 static <TextendsComparable<T> List<T> student_classes.Heap< T extends Comparable< T > >.sort (List< T > *lst*) `[static]`

This method is intended to be called by clients wishing to pass arbitrary collections of Comparable objects into the Heap sort logic. Note: it took longer to figure out the generic signatures for this method's signature than to write its actual logic, which is:

1. Use the `lst` parameter to create a `Heap`, that is, pass it directly to the constructor.
2. Invoke the `sort` method on the newly-created heap.
3. Use the `asList` method to return a `List` of the appropriate objects.

Parameters

lst	

Returns

2.1.3.5 String student_classes.Heap< T extends Comparable< T > >.toString ( )

Public override ... sanity check, just prints array style.

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

- Heap.java