```
//HeapSort.h
//Template functions to perform a heap sort in ascending order
//Programmer: Randy Miller
//Date last modified: April 8, 2015

/*For an exposition of this code and the data structure that it implements, cf.
http://www.personal.psu.edu/jum385/pdfs/HeapSort.pdf */


#ifndef HEAPSORT_H
#define HEAPSORT_H

#include <algorithm>

/*In the following code the typename 'I' represents a random-access iterator.
Besides supporting an iterator of this category, the type of the underlying
container template must be completely ordered by a (possibly overloaded) operator
'<', according to which this algorithm will sort in an ascending order.*/

template<typename I>
void heapSort(I lower, I upper);;

template<typename I>
void makeHeap(I lower, I upper);

template<typename I>
void undoHeap(I lower, I upper);

template<typename I>
void insertIntoHeap(I lower, I upper);

template<typename I>
void removeFromHeap(I lower, I upper);

template<typename I>
```

```
void descendThruHeap(I lower, I upper, I mover);


//sort the sequence
template<typename I>
void heapSort(I lower, I upper)
{
    makeHeap(lower, upper);
    undoHeap(lower, upper);
}

//create a heap in the range >= lower and < upper
template<typename I>
void makeHeap(I lower, I upper)
{
    if (--upper == lower)
        return;
    else
    {
        makeHeap(lower, upper);
        insertIntoHeap(lower, upper);
    }
}

//replace a heap in the range >= lower and < upper with a sorted sequence
template<typename I>
void undoHeap(I lower, I upper)
{
    if (--upper == lower)
        return;
    else
    {
        removeFromHeap(lower, upper);
        undoHeap(lower, upper);
    }
```

```
}

//insert the element at upper into a heap in the range >= lower and < upper
template<typename I>
void insertIntoHeap(I lower, I upper)
{
    I   child = upper,
        parent = lower + ((child - lower) - 1) / 2;
    if (parent < lower || child == parent || *child <= *parent)
        return;
    else
    {
        std::iter_swap(child, parent);
        insertIntoHeap(lower, parent);
    }
}

/*remove the element now at lower to the front of the sorted sequence at upper,
 and restore the properties of a heap in the range >= lower and < upper*/
template<typename I>
void removeFromHeap(I lower, I upper)
{
    std::iter_swap(lower, upper);
    descendThruHeap(lower, upper, lower);
}

/*reposition the element now at mover to its proper place in a heap
 in the range >= lower and < upper*/
template<typename I>
void descendThruHeap(I lower, I upper, I mover)
{
    I   parent = mover,
        child1 = (((2 * (parent - lower) + 1) <= (upper - lower)) ?
           (lower + 2 * (parent - lower) + 1) : upper);
    if (child1 == upper)
```

```
        return;
    I   child2 = child1 + 1,
        childx = ((child2 == upper || *child1 >= *child2) ? child1 : child2);
    if (*parent >= *childx)
        return;
    else
    {
        std::iter_swap(parent, childx);
        descendThruHeap(lower, upper, childx);
    }
}

#endif
```