Using Pointers and `malloc` in C.

Want to write another sort? Too bad! We want you to anyway.

For this week's lab, we want you to implement a sort that blends elements of the heap sort you implemented for Lab 4 and the tree sort that was presented as an example in class.

The heap sort we had you translate into MIPS assembly language did not use dynamic memory allocation. This is typical given the way that heap sort is traditionally implemented. The heap is a tree-like data structure, but the heap in heap sort is stored in a pre-allocated one-dimensional array rather than with dynamically allocated structures that point to their subtrees. Unfortunately, this means that the array must be preallocated so there is a fixed, maximum number of input elements that we can sort.

Our tree sort, on the other hand, was presented to illustrate dynamic memory allocation. The tree sort had no pre-specified limit on how many input values it could handle. As long as `malloc` continued giving it memory, the program could read and sort larger and larger input files.

Heap sort, however, can guarantee better performance. If the input values presented to our tree sort are already in order, it will build a tree that looks more like a linked list than a tree. Each insertion will take linear time in the current size of the tree rather than logarithmic time, so the sort will take quadratic time. Heap sort runs in logarithmic time regardless of the ordering of its input.

You have probably seen techniques that keep binary search trees balanced. Such techniques could be used to design a version of tree sort that always runs in logarithmic time, but they are complicated.

It is easier to keep a heap balanced than a search tree because there are fewer constraints on the structure of a heap. Within a heap, the value stored at the root of a subtree must simply be smaller than all the other values stored in the subtree. There are no constraints on whether values in the left subtree are less than or greater than those in the right subtree as there are in search trees. Recognizing this, we want you to implement a version of heap sort that uses a dynamically allocated and balanced heap rather than a heap stored in an array.

Each element of your heap will be stored in a separate struct allocated using `malloc`. Each struct will have components to store a value that will serve as the sort key (an int), pointers to left and right subtrees, and a count of the total number of values stored in the subtree for which the struct is the root.

When asked to add an element to a subtree, first check to see if the subtree is empty. If so, create a new tree consisting of one node holding the new value. If the subtree is not empty, compare the new element to what is stored in the root of the subtree. Whichever of these two values is smaller should become the value stored in the root and the other value should be added recursively to the smaller of the subtrees of the root.

There is a trick that can make coding this a bit easier. Make your trees left-handed. That is, when you add a value to a tree, always make sure that when you are done the left subtree contains at least as many values as the right subtree. If necessary, this can be accomplished by simply swapping the left and right subtrees after an insertion (or removal). If you do this, then when performing an insert you can ensure the heap remains balanced by performing any recursive inserts on the right subtree since it will never be larger than the left subtree.

You will need and use a remove function that removes the smallest value from a heap. Your main program will insert each element read from its input file. When the input ends, it will then repeatedly remove (and print) the smallest value from the heap until the heap is empty. The remove function you write should preserve the left-handedness of the tree and use the free function appropriately.

This should be a very short program. My version took about one hundred lines. Despite how small the program is, I would like you to break it up into two .c files named balancedHeapSort.c and balancedHeap.c and a .h file named balancedHeap.h. You should also create and use a makefile to control the compilation of your program. Your makefile should resemble the one used for the tree sort example in class.

Submitting Your Work

Within a terminal window, use the cd command to make the directory containing your .c and .h files your current working directory. Then type the command:

```
tar -cf balancedHeap.tar balancedHeapSort.c balancedHeap.c balancedHeap.h makefile
```

Next type the command:

```
turnin -c 237 balancedHeap.tar
```

Respond to the prompts appropriately and your work should be submitted.