Assignment 1

Introduction

For your first Algorithm’s programming assignment, you are to implement a heapsort algorithm using a binary tree structure. Since a heap can be efficiently stored in an array, there is never a need to implement a heap as a binary tree, even though heaps are often drawn that way. However, implementing a heap as a tree will give you needed practice in manipulating trees.

To make this task more challenging, tree nodes will be restricted to left, right, and parent pointers, plus a field to store the integer value of the node. You may, however, use auxilliary data structures. Full credit will be reserved for implementations that use the fewest and most appropriate auxilliary data structures. All tree-based heap operations must be as efficient as those for a heap implemented as an array:

- reading in a value and adding it to the heap (constant time)
- ordering the heap (linear time)
- extracting the extreme value (logarithmic time)

You may not use an array anywhere in your implementation.

I/O

Your executable must be named:

```
heapsort
```

The executable will take a file name as a command-line argument and will produce, on stdout, the integers found in the given file in sorted order. The output must be all integers on a single line, each integer separated from the next by a single space.

Here is an example invocation:

```
$ for i in {1..5}; do echo $RANDOM; done > integers
$ heapsort integers
4106 6986 9883 19357 27330
$ heapsort -d integers
27330 19357 9883 6986 4106
$
```

where $ is the system prompt. The file to be sorted is a free-format text file. That is, the integers within are separated by arbitrary amounts of whitespace and every line ends with a newline.

The executable must handle the following options:

<table>
<thead>
<tr>
<th>option</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-v</td>
<td>give author’s name and explanation on how the implementation performs the sorting in $\theta(n \log n)$ time – In particular explain how reading in a value and adding it to the heap takes constant time, how ordering the heap takes linear time, and how extracting the extreme value takes logarithmic time. — also give some empirical evidence (such as a table of input sizes versus time) that indicates your sort runs in $\theta(n \log n)$ time. — then immediately exit</td>
</tr>
<tr>
<td>-d</td>
<td>sort in decreasing order (default is increasing order)</td>
</tr>
</tbody>
</table>

Here is a program that features some handy-dandy option-handling code that you may use verbatim without credit: options.c

Implementation details

You must implement your heapsort algorithm in C99. You must provide a makefile which responds properly to the commands:

```
make heapsort
```

Your program must compile cleanly and with no warnings or errors and it must be compiled with the highest level of error checking (the -Wall option for gcc). It must name the executable heapsort (not heapsort.exe for you Cygwin users). You may develop on any system you wish but your program will be compiled and tested on a Linux system. Only the most foolish students would not thoroughly test their implementations on a Linux system before submission.

Note: depending on where you develop your code, uninitialized variables may have a tendency to start with a value of zero. Thus, a program with uninitialized variables may work on your system but fail when I run it. I won’t care, as you are mature enough not to have uninitialized variables. You may have other errors as well that do not reveal themselves until run on my system. Again, that’s not my problem. If I am feeling generous and have the time, I may figure out where your error is and, perhaps, give you a few meager points back, but don’t depend on it.
Documentation
All code you hand in must be attributed to its authors. Comment sparingly but well. Do explain the purpose of your program. Do not explain obvious code. If code is not obvious, consider rewriting the code rather than explaining what is going on through comments.

Grading
Severe deductions will be made for poor performance or not following the specification (e.g. extra output).

Here are some additional deductions which might apply:
- bad style, lack of appropriate modularizations (20 point deduction max)
- include file names that do not match (in a case-sensitive sense) their include statements (10 point deduction)
- variables that are defined, but not used (5 point deduction)
- variables that are defined and set, but not used (5 point deduction)
- a makefile that does not use -Wall or -std=c99 for all compilations (5 point deduction)
- a makefile without a rule for each module (10 point deduction)
- a makefile without a clean rule or a clean rule that does not remove all object files and the executable (5 point deduction)

This list is not inclusive; other deductions may apply.
Note: you are required to use the standard read pattern for reading in your data:

```
read a value
while not at end of file
{
    process the value that was read
    read a value
}
```

Failure to do so will result in a 60 point deduction.

A request will a regrade will result in a 20 point deduction, unless the problems your implementation has are deemed to be not your fault.

Submission
To submit assignments, you need to install the submit system.

- linux and cygwin instructions
- mac instructions

You will hand in (electronically) the final source code with the command:

```
submit cs201 lusth assign1
```

The submit program will bundle up all the files in your current directory and ship them to me. Thus it is very important that only the source code and any testing files be in your directory. This includes subdirectories as well since all the files in any subdirectories will also be shipped to me. I will deduct points for intermediate files (like object files), so be careful. You may submit as many times as you want before the deadline; new submissions replace old submissions.