Implementing Heap Sort
Lab #3

By
Joshua Matthews

CS 303 Algorithms and Data Structures
September 22, 2015

1. Problem Specification

The goal of this Lab assignment was to implement the Heap Sort and test the difference in time it takes to sort the input from various sized files passed in to the program and also compare it to the Insertion Sort and Merge Sort.

2. Program Design

This program required three functions besides the main function to implement the Heap sort. It had to accept and extract values from a file, test the Heap sort, and display the amount of time it took to complete the Heap sort.

The following steps were needed to develop this program:

A. I needed to setup the environment with the correct c++ libraries included and main function to accept arguments that would be passed into the program.
B. Next I setup the code to take the file passed in and extract the values from the file and push them on to a vector to be used for later.
C. I then created a function which implemented the Heap sort which has one vector passed in as reference. This function handled calling the build_max_heap function to organize the array and then iterated through the array and called max_heapify function on the array.
D. I then create the function for build_max_heap which took in the array by reference. The function then goes through half the array and sends the array and the element number that it is on to the max_heapify.
E. The max_heapify function took in the vector by reference and the integer value of which element it was on. It figures out the left and right child nodes to the parent node and checks to see that they are in order and if they are not, puts them in order and will make sure the rest are in order if nodes are changed around.
F. I then setup code around where I am invoking the Heap sort function on the data from the file to calculate how long the Heap sort took to complete.

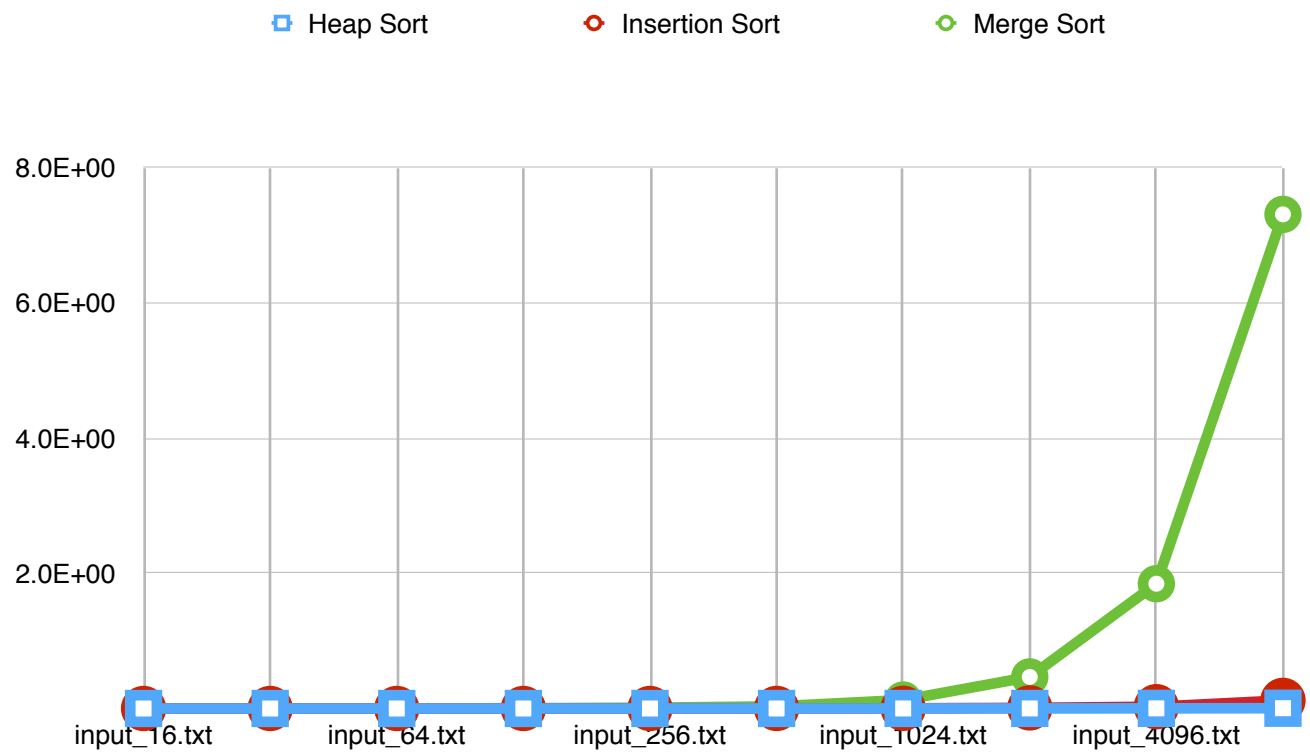
The included libraries were: iostream(for printing out to the screen), fstream(for reading in data from a file), vector(for having a dynamically sizing array), ctime(for calculating program running time in seconds).

3. Testing Plan

The Lab04.cpp file tests the Heap sort on the values from whatever file is passed in. The program accepts a file and parsers out each number in the file and pushes them back onto the vector object that was created to hold all of the values. The initial values we retrieve from the files are random numbers between 1 and the number of elements that are in the file inclusively. Once the vector is filled with every value from the file the current time is captured in a variable then the Heap sort is invoked. The vector is passed in by reference so there is no time being wasted on copying the values over to the function and allows the function to have direct manipulation over the object. Once the Heap sort is done we capture the current time minus the start time to calculate how long the function took in seconds, then display it to the screen for the user to see.

4. Test Cases

The test cases are shown below. The graph shows how long the sorts took from input_16.txt to input_8192.txt.



5. Analysis and Conclusion

The results show that the Heap sort has a time complexity of O(nlogn) in both best case and worst case scenarios. The Heap sort is faster than both the Insertion sort and the Merge sort. However, though the Heap sort is fast, once the array gets big enough you could experience cache misses with how the Heap sort traverses through the array.
For question four in our lab document, a good asymptotic upper bound for:

$T(n) = T(n-1) + n$

is T(n) = O(n^2) and for:

$T(n) = 1$, if $n = 1$
$= 2T(n/2) + n$

is T(n) = O(nlog(n)).

6. References

The www.cplusplus.com has references and documentation on all of the standard libraries that were used in this program. Input files were supplied by Abhishek Anand.