Ben Haase

CS3130 Project 2

11/2/15

**Part C: Heap Sort**

Heap Sort's time efficiency is characterized mainly by two functions. The function max heapify compares elements and is a recursive function. Max heapify performs a constant amount of operations each time it is called, and works recursively looking at the element passed to it and the two elements below it in the heap. Max heapify ceases recursion at the worst case when it reaches the bottom of the heap. Max heapify moves in powers of two, since each time it is called it looks at i, 2i, and 2i+1. Therefore it will run until 2x = n, so lgn times. In the heap sort function this function along with some constants, are called n times. The function build max heap is called once at the beginning of the heap sort function, outside of the iteration, and builds a heap from the bottom up. It calls the function max heapify floor(n/2) times, since max heapify has a time efficiency of Θ(lgn), the time efficiency of build max heap will be ε Θ(n). For building a heap from the top down, each iteration of the insertion algorithm an element is placed in the bottom of the heap and moved to the top. Since h=floor(lgn), there will be at most h comparisons per element. With n elements undergoing floor(lgn) comparisons, the time efficiency for a building a heap from the top down ε O(nlgn). For heap sort with bottom up construction we have n + nlgn, so heapsortbu ε Θ(nlgn). For heapsort with top down construction we have nlgn +nlgn, so heapsorttd ε Θ(nlgn). We would expect heapsorttd to take roughly twice as long as heapsortbu.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **1000** | **10000** | **100000** | **1000000** |
| **Build Heap** | 0.0003 | 0.0018 | 0.0164 | 0.2073 |
| **Heap Sort Total** | 0.0018 | 0.0268 | 0.3562 | 4.1051 |
| **Build/Total** | 0.166666667 | 0.067164 | 0.046042 | 0.050498 |
| **1/(1 + lg(N))** | 0.09119275 | 0.06999 | 0.056787 | 0.047775 |

In the implementation of heap sort using a bottom up construction, very similar result to the theoretical time complexity were observed. The percentage of time used in heap construction is very close to theoretical expectations based on heap construction ε Θ(N) and heap sort ε Θ(NlgN). Theoretically dividing the time to build the heap by the total time should be roughly equal to N/(N+NlgN) = 1/(1+lgN). In the results very similar values to this were obtained. The slight variance from the actual value is most likely due to the lack of precision available in timing, specifically for the heap construction time due to its fast execution. In an effort to try to improve the accuracy of these times, the execution time of heap sort was determined by finding the average for ten arrays ran ten times each for each value of N. Through this heap construction time was able to be measured, but its accuracy for smaller values of N has a greater variance caused by low execution times.

Graphically both the total time of heap sort and the time spent in heap construction appear to be linear, which is what we would expect. As previously shown, bottom up heap construction occurs in linear time, and as N gets sufficiently large the total time of heap sort will be dominated by N giving it a linear appearance.

Looking at F(10N)/ F(N) we expect to see similar values to that of 10Nlg10N/NlgN with so we expect these values to be decreasing and leveling off around 10. Looking at the data this holds true for the implementation.

|  |  |  |  |
| --- | --- | --- | --- |
| **10N/N** | **10000/1000** | **100000/10000** | **1000000/100000** |
| **F(10N)/F(N)** | 14.8888889 | 13.29104478 | 11.52470522 |
| **10Nlg10N/NlgN** | 13.33333 | 12.5 | 12 |

As show in the graph above both functions are decreasing and appear to be leveling off as they approach ten. The line for the actual implementation does not exactly follow the expected graph but is fairly close and the difference may be attributed to the difficulty associated with timing precision.