CS 310
Assignment 406

Arjan Regmi
ar4752

For the program in this analysis, we implement the `build_heap` and `heap_sort` functions to construct a max-heap and sort the items in a heap respectively. The values used in implementing these functions are random sets of integers from the input file.

For both methods, the value for n is taken to be the input size, that is, number of elements in the array. We use the same input for both methods as we are trying to analyse the two algorithms.

For the empirical analysis we measure the number of basic operations performed to construct a heap (build heap) for a given input size.

To analyze the `build_heap` function, we chose another function, `percolate_down`, on line 153 as our basic operation.

To analyze the `heap_sort` fucntion, we chose the two functions, `build_heap`, and `percolate_down` in lines 108 and 115 as our basic operations.

An empirical analysis of the `build_heap` running the algorithm for multiple values of n produces the results shown below. Standard functions $f(n) = n$ is also shown.

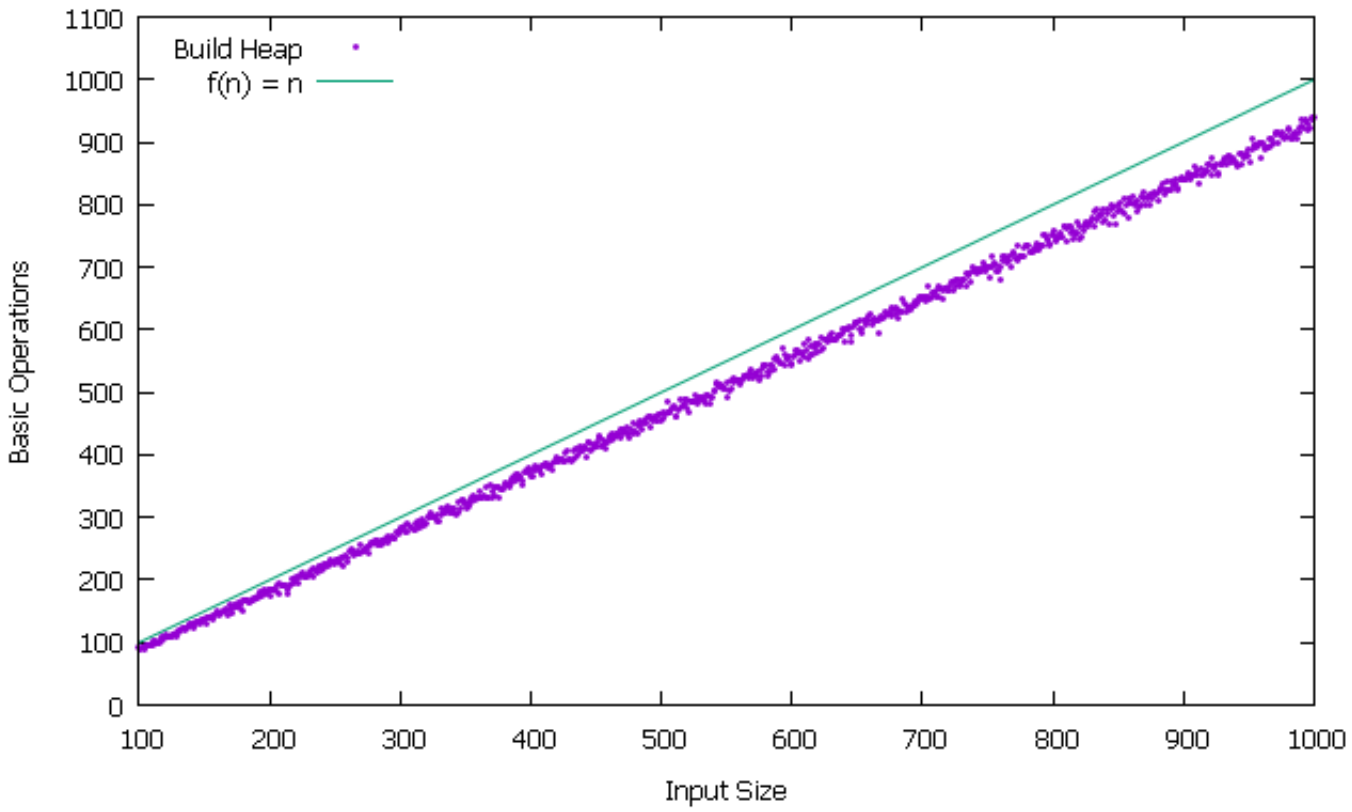

Figure: Basic Operation vs Input Size (`build_heap`)

An examination of the code itself explains the empirical results when we observe that the `build_heap` shows a linear nature when plotted for multiple values of n. Here, the standard function $f(n) = n$ asymtotically upper bounds the running time curve for this function.

Therefore, we conclude that the `build_heap` function in this program is described by

$$T(n) \in O(n)$$

Secondly, an empirical analysis of the `heap_sort` function running the algorithm for multiple values of n produces the results shown below. Standard functions $f(n) = n \lg(n)$ is also shown.

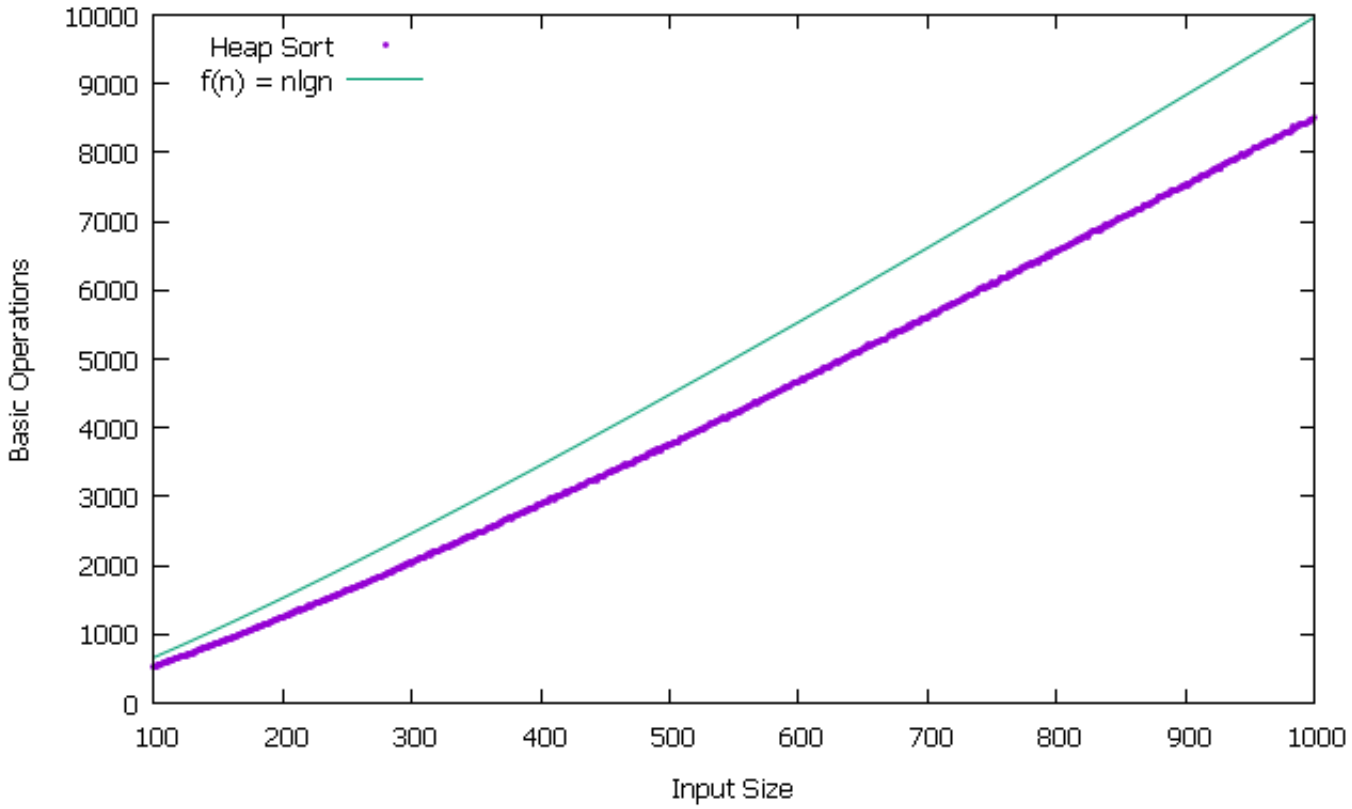

Figure: Basic Operation vs Input Size (`heap_sort`)

For the analysis of the `heap_sort` function, the above figure shows that the standard function $n \lg(n)$ upperbounds the number of basic operation as the input size increases. We can also know from the code and our `build_heap` analysis, that since `build_heap` takes $O(n)$ time and each `percolate_down` takes $O(\lg n)$ the array is sorted in $O(n \lg n)$ time.

With the help of the graph above we can conclude that the `heap_sort` has an analysis of:

$$T(n) \in O(n \lg n)$$